FPT-Approximations for k-Min-Sum-Radii and Practical Advances for k-Means

Inaugural-Dissertation

zur Erlangung des Doktorgrades der

Mathematisch-Naturwissenschaftlichen Fakultät der

Heinrich-Heine-Universität Düsseldorf

vorgelegt von

Lukas Roland Drexler

aus Aschaffenburg

Düsseldorf, Januar 2026

aus dem Institut für Informatik
der Heinrich-Heine-Universität Düsseldorf

Gedruckt mit der Genehmigung der
Mathematisch-Naturwissenschaftlichen Fakultät der
Heinrich-Heine-Universität Düsseldorf

Berichterstatter:

1. Prof. Dr. Melanie Schmidt

2. Prof. Dr. Heiko Röglin

Tag der mündlichen Prüfung:

Ich versichere an Eides statt, dass die Dissertation von mir selbstständig und ohne unzulässige fremde Hilfe unter Beachtung der „Grundsätze zur Sicherung guter wissenschaftlicher Praxis an der Heinrich-Heine-Universität Düsseldorf“ erstellt worden ist.

Abstract

In a world of ever-growing amounts of data, the automatic discovery of meaningful structure has become increasingly important. Clustering addresses this challenge by grouping data points into subsets based on their similarity, without the need for labeled data. It can therefore be viewed as a form of unsupervised machine learning. Another way of looking at it is through the lens of combinatorial optimization and algorithmics. One way of formulating clustering as an optimization problem is the following: given a point set P and a distance function d on this set, find a set of at most k centers and an assignment of points to those centers, so as to minimize some objective function. In some applications, solutions are additionally required to satisfy further constraints. For instance, a center may only serve a certain number of points, or the points may belong to different protected groups, with the requirement that no group is underrepresented in any cluster. In this thesis, we mainly study two objectives: k-Means and k-Min-Sum-Radii. It is structured into three parts.

Part I provides an introduction and establishes the formal definitions, concepts, and notation used throughout the thesis.

Part II is devoted to k-Min-Sum-Radii clustering and the more theoretical aspects of clustering. This objective is of special interest, as it can exhibit counterintuitive behavior and differs significantly from other, seemingly related objectives. As a result, many standard algorithmic techniques that are successful for objectives such as k-Center or k-Median fail to apply. After investigating several structural properties of k-Min-Sum-Radii, we present two FPT approximation algorithms for different variants. The first is a $(1 + \varepsilon)$-approximation for Euclidean k-Min-Sum-Radii. The second is a $(6 + \varepsilon)$-approximation for k-Min-Sum-Radii in general metrics, and with mergeable constraints, a class of constraints that share a common structural property.

Part III focuses on the k-Means objective from a more practical perspective. We introduce a new algorithm, which is extensively tested against other commonly used algorithms. While it can be shown to yield a constant-factor approximation, its main strength lies in its practical performance. Across most of the experiments, it outperforms the other algorithms in terms of cost, while remaining competitive in terms of runtime. Furthermore, we explore the algorithm's ability to approach the globally optimum solution on a selection of datasets for which the optima are known. Although the algorithm typically comes very close to optimality, it consistently fails to achieve the final step. This observation motivates a brief discussion of emerging questions concerning dataset clusterability and the ability of this class of algorithms to reach globally optimal solutions.

Contents

Part I.

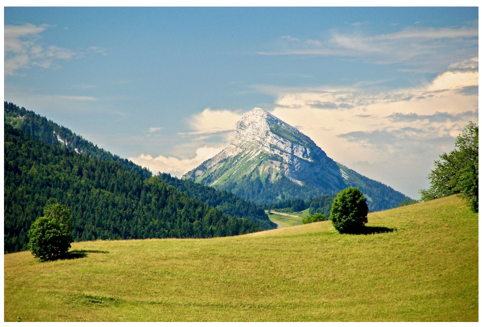
Introduction

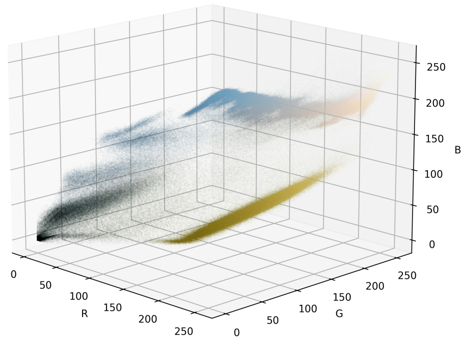
The concept of *clustering* is something that, even without any formal training, can easily be understood intuitively. In their book *Cluster Analysis* [51], Everitt, Landau, Leese and Stahl describe it as “the grouping of similar things to produce a classification”. So given a set of different individual objects, we want to group *similar* objects together while separating *dissimilar* ones, and in the process uncover some meaningful structure in our data. With the ever-larger amounts of data, being able to automatically detect structures and organize the data has become increasingly important. Documents need to be grouped by topic, customers by spending behavior, patients by symptoms and risk factors. To be able to do this, we have to overcome two central challenges. Firstly, we need to define what similar means. Depending on the use case at hand, this can be more or less obvious. Secondly, we need a way of judging the quality of a clustering. Even if we have decided on a similarity measure, there might exist multiple, equally sensible clusterings. Additionally, in many modern applications, the data are high-dimensional or otherwise too complex to visualize directly, making it impossible to simply look at the data and conclude whether the discovered clustering is meaningful. In this thesis, however, we adopt a theoretical perspective on clustering, where both the similarity measure and the quality measure for clusterings are treated as part of the problem specification. Instead of arguing whether a given measure is appropriate for a certain real-world application, we accept it as fixed and study the properties of the algorithmic problem.

To take a step towards a more formal framework, we describe what we call *center-based k-clustering*. Given a set of data points, a distance function (which serves as measure of dissimilarity) on these points, and a natural number k, the goal is to (1) find a set of k *centers*, and (2) an assignment of points to these centers. This assignment partitions the points into k *clusters*. One can think about the centers as representatives of their cluster. From this viewpoint, clustering can also be interpreted as reduction of complexity. If the original dataset is very large, and the computational task to be performed is quite costly, identifying a set of adequate representatives and then performing the task on these representatives can be advantageous. Of course, the question arises what “adequate” even means in this context. We have not yet addressed the topic of judging the quality of such a center set. Usually, this is done via an objective function that is based on the underlying distance function in some way.

For a concrete example, we will consider k-means clustering. In this problem setting, the goal is to place the k centers in such a way that the sum of squared distances of points to their closest centers is minimized. A classic application of k-means is image compression[1]. Given an image (e.g. as PNG, JPG or PDF file), the goal is to reduce the file size while maintaining some visual fidelity. To be able to perform clustering on such a file, we first need to transform it into a suitable input. Each pixel of an image has three values between 0 and 255, indicating the amount of red, blue and green color in that pixel (known as RGB values). For example, an image with a resolution of 1600x900 can be interpreted as a set of $1600 \cdot 900 = 1440000$ 3-dimensional points, with the RGB values denoting the coordinates. Figure 1 shows this for an exemplary image[2]. On the

[1]This is one of the application examples used in [28].


(a) Original image



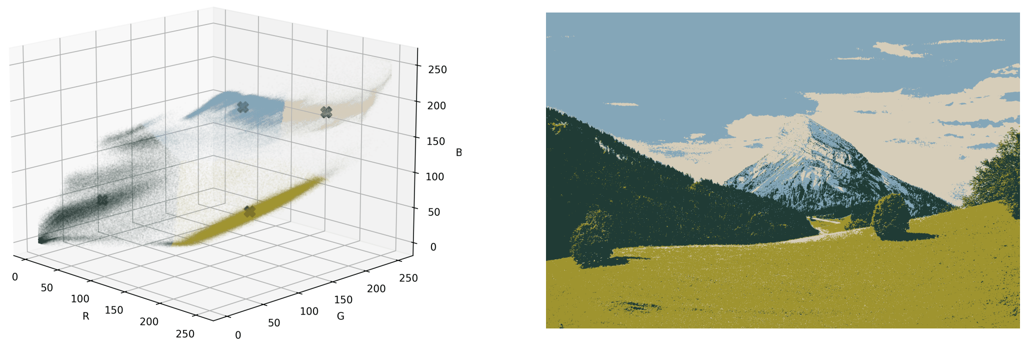
(b) RGB values

Figure 1.: The image on the left can be interpreted as 3-dimensional dataset by identifying each pixel by its RGB-values. Subfigure (b) shows the resulting dataset, where each point has the color of its corresponding pixel in the original image.

dataset in Figure 1(b), we can already see some structure. The original image has large portions of light blue and white pixels for the sky, darker green shades for trees, and lighter, brownish and yellowish shades of green for the meadow. In the $3d$-dataset, this is reflected by a higher concentration of points in the areas that correspond to RGB values of these color shades. As the distance function, we can simply use the Euclidean distance. This is now a dataset on which we can perform k-means clustering. If, for instance, we choose $k = 4$, this will result in 4 centers being chosen. Then each point can be assigned to its closest center, partitioning the set into 4 clusters. A possible result is shown in Figure 2 (a). Note that the coordinates of each of the centers now also correspond to RGB values a color. The actual compression lies in assigning each pixel the color of the closest center of its corresponding 3-dimensional point. Figure 2(a) shows the resulting compressed image. While it naturally cannot preserve all details of the original, the main scenery and features are still clearly visible. At the same time, saved as a PDF file, the size is 87 Kilobytes instead of roughly 8 Megabytes of the original.

Clustering can be viewed (and is often described) as a form of machine learning, called unsupervised learning. It differs from supervised learning in that the data is not labelled beforehand. In supervised learning, the goal is to train a model with labelled data so that it performs well on unseen data. Unsupervised learning can, in a sense, be interpreted as trying to *find* the labels – i.e., the clusters – in the data. Another way of looking at clustering is through the lens of combinatorial optimization problems. Such problems are characterized by a discrete (and often finite) set of solutions – whose number is usually exponential in the input size – and some objective function that assigns a value to every solution. The goal is to find a feasible solution that minimizes (or, depending on the exact problem, maximizes) this objective. In our case, the number of possible partitions

[2]This specific image is used as illustration in our paper [41].



(a) Solution for $k = 4$ (b) The corresponding image

Figure 2.: Performing k-means clustering with $k = 4$ centers (i.e. colors). Subfigure (a) shows a possible solution for 4 centers. Each point gets the color of its closest center. The resulting image, depicted in Subfigure (b), uses only these four colors, reducing the file size by a factor of roughly 100.

can be thought of as such a finite set of solutions, or, if centers are restricted to a finite set of choices, all possible ways of choosing k centers. In the above example of k-means clustering, the objective to be minimized is the sum of squared distances of points to their closest centers. There are, of course, other objectives that can be applied, some of which will be explored later on. Many combinatorial optimization problems are computationally hard, and we cannot expect to find algorithms that always compute optimal solutions efficiently. This is also the case for most clustering problems. It is therefore of high interest to design algorithms whose runtimes do not depend exponentially on the input size while still maintaining some guarantees for solution quality. A central aspect in the design of such algorithms is judging them by *worst-case analysis*. This means that we want to derive guarantees (be it with respect to runtime or solution quality) that hold universally, i.e., even in the worst possible case. This approach differs fundamentally from the machine learning point of view described above. In machine learning, these types of guarantees are usually absent (and often impossible to obtain). Instead, the underlying assumption is that the data is generated by some probability distribution, and that the sample used for learning is sufficiently representative. In approximation algorithm design, there are classically no such assumptions about input instances.

There are, of course, clustering approaches that do not work with centers and do not seek to minimize some objective function. For example, density based methods like DBSCAN or OPTICS aim at identifying dense regions in the dataset and then construct the clusters by exploring the neighborhoods of these regions. These methods can excel on instances where many center- and objective-based clustering approaches fail, e.g. datasets with non-convex or elongated cluster shapes. Some methods compute *hierarchical clusterings*, where the number of clusters is not an input parameter. Instead, they produce nested clusterings for all values of k from 1 to n (with n being the number of input points). These can be viewed as levels of granularity: in the n-clustering, which is the finest level of granularity, every point is in its own cluster; for the 1-clustering, all points are in one large cluster; and for all clusterings in between, every cluster at a "finer" level is entirely contained within a cluster at a "coarser" level. These methods usually work in an agglomerative manner, starting with every point in its own cluster and then consecutively merging clusters to obtain coarser clusterings. Typical algorithms for this are Single Linkage, Complete Linkage, Average Linkage or Ward's Method. Some strategies work with similarity measures instead of dissimilarity measures, e.g. spectral clustering or correlation clustering. Yet other methods try to judge clusters by their silhouette.

However, the main focus of this thesis is on center-based k-clustering, viewed as an optimization problem. It is split into three parts. In Part I, we establish a formal framework for center-based k-custering that enables us to make formal mathematical statements. Furthermore, we introduce concepts that are important for the remainder of the thesis. Part II explores the theoretical side of algorithm design, discussing some properties of a specific objective function for center-based k-clustering and then presenting two approximation algorithms for this objective. In Part III, we look at k-means clustering from a more practical angle. We present an algorithm that performs very well in practice, even though its theoretical guarantees are not the best. We compare it to other popular algorithms and explore its ability (or lack thereof) of finding optimum solutions on selected instances.

I.1. Center-Based k-Clustering

In the following, we define a formal framework that lets us talk about instances and solutions to clustering problems. Furthermore, we define all terms and concepts that are needed throughout this thesis.

Definition 1. *An instance $I = (P, d, k)$ of the* Center-Based k-Clustering Problem *consists of*

- *a point set P with $|P| = n$*
- *a distance function $d : P \times P \to \mathbb{R}_{\geq 0}$*
- *a natural number $k \in \{1, \dots, n\}$*

A feasible solution $S = (\mathcal{C}, \sigma)$ consists of a set of centers *$\mathcal{C} \subseteq P$ with $|\mathcal{C}| \leq k$, and an assignment $\sigma : P \to \mathcal{C}$, assigning each point of P to exactly one center in $\mathcal{C}$.*

When talking about any given solution $S = (\mathcal{C}, \sigma)$, we usually assume a fixed numbering of the centers, i.e. $\mathcal{C} = \{c_1, \dots, c_k\}$. Note that if we take the set $\sigma^{-1}(c_i)$ for all $i \in [k]$, these sets form a partition of P. We call such a set *cluster* and refer to the set of clusters induced by S as

$$\mathscr{C}_S = \{\sigma^{-1}(c_i) \mid i \in [k]\}.$$

For a given center $c_i \in \mathcal{C}_i$, we use $C_i = \sigma^{-1}(c_i)$ to refer to its associated cluster. For a cluster $C_i \in \mathscr{C}$ with center c_i, we denote its *radius* as

$$r_i = \max_{p \in C_i} d(p, c_i),$$

and call $r_1, \dots, r_k$ the *induced radii* of S. We assume without loss of generality that the radii are sorted non-increasingly, i.e. $r_1 \geq \dots \geq r_k$. In some cases, it might be more convenient to think about a solution in terms of this partition, so we might refer to $\mathscr{C}_S$ as *the solution* from time to time. See Figure I.1.1 for an example of how a set of centers with an assignment induces clusters and radii.

While a distance function d is defined on $P \times P$, we also use the notation $d(p, X)$ to denote the distance from a point p to a *set* of points $X \subset P$. By this, we mean the distance from p to its closest point in X, i.e.

$$d(p, X) = \min_{x \in X} d(p, x).$$

Sometimes it can be more convenient to think about solutions as a collection of *balls.* To this end, we first formalize what a ball is.

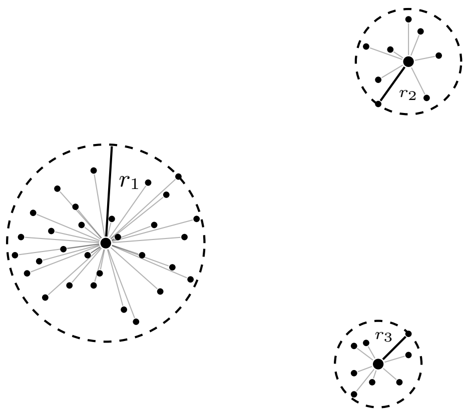


Figure I.1.1.: A solution $S = (\mathcal{C}, \sigma)$ for $k = 3$. The assignment induces three clusters, and the radius of a cluster is the distance from its center to the farthest assigned point.

Definition 2. *Let P be a set of points in a metric space. Let $c \in P$ be an arbitrary point and $r \in \mathbb{R}_{\geq 0}$. Then we call the set*

$$\mathrm{B}(c, r) = \{p \in P \mid d(c, p) \leq r\}$$

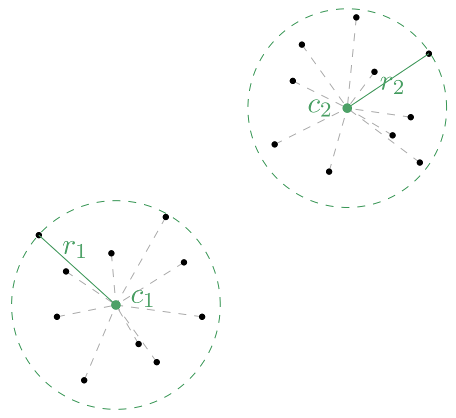
the r-ball around *c.*
We say that P is enclosed *or* covered *by a ball* $\mathrm{B}(c, r)$ *if $P \subseteq \mathrm{B}(c, r)$. The ball with the smallest radius that encloses P is called* Minimum Enclosing Ball (MEB) *of P and we denote it by* $\mathrm{MB}(P)$. *Furthermore, for $\gamma \geq 1$, we denote by* $\mathrm{MB}_\gamma(P)$ *the ball that results from increasing the radius of* $\mathrm{MB}(P)$ *by a factor of γ.*

With this definition in place, we can define the concept of a *covering.*

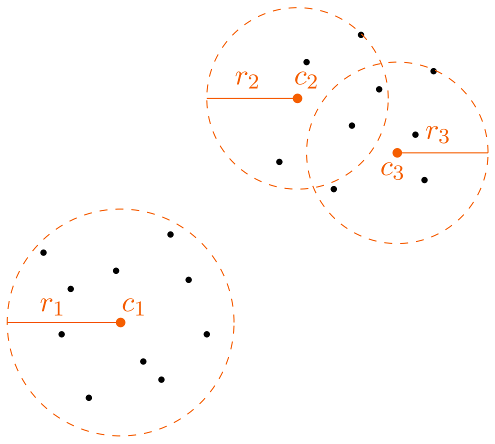
Definition 3 (Covering)**.** *Let $I = (P, d, k)$ be an instance of* Center-Based *k*-Clustering. *Let $\mathscr{B} = \{\mathrm{B}(c_1, r_1), \ldots, \mathrm{B}(c_k, r_k)\}$ be a set of k balls. We call $\mathscr{B}$ a* covering *of P, if the union of balls in $\mathscr{B}$ covers P, i.e. $\bigcup_{i=1}^{k} \mathrm{B}(c_i, r_i) = P$. We refer to the center set of $\mathscr{B}$ as $\mathcal{C}(\mathscr{B}) = \{c_1, \ldots, c_k\}$ and the radius set of $\mathscr{B}$ as $\mathcal{R}(\mathscr{B}) = \{r_1, \ldots, r_k\}$.*

Note how each feasible solution $S = (\mathcal{C}, \sigma)$ to Center-Based k-Clustering corresponds to a covering, namely the set of balls defined by the centers in $\mathcal{C}$ and the radii induced by σ. But a covering does not necessarily give a feasible solution [1] right away, as balls in the covering can overlap, making it unclear what the corresponding assignment should be. Moreover, a covering could also contain balls that have unnecessarily large radius, i.e. the radius might not correspond to the distance of the center of the ball to an assigned point. See Figure I.1.2 for an illustration of the differences.

[1]in the sense of Definition 1



(a) A solution consisting of two centers c_1 and c_2 along with assignment σ (indicated by the dashed lines). The induced radii r_1 and r_2 correspond to inter-point distances. The resulting balls $\mathrm{B}(c_1, r_1)$ and $\mathrm{B}(c_2, r_2)$ constitute a covering of P.



(b) A possible covering with three balls $\mathrm{B}(c_1, r_1), \mathrm{B}(c_2, r_2)$ and $\mathrm{B}(c_3, r_3)$. The union of the balls contains all points, but it does not imply an assignment right away. Also, r_1 is unnecessarily large.

Figure I.1.2.: Differences between solutions and coverings.

I.1.1. Metric Spaces

In Definition 1, we do not make any assumption towards the properties of the distance function d, other than its non-negativity. But it is common to assume that the distance function d is a *metric*. This encapsulates properties that one might intuitively expect from a distance measure and is in most cases necessary to even have a chance at designing efficient algorithms with provable guarantees.

Definition 4 ((Pseudo-)Metric Spaces)**.** *Let X be an arbitrary set. A function $d : X \times X \to \mathbb{R}_{\geq 0}$ is called a* metric *on X if it satisfies the following properties for all $x, y, z \in X$:*

(1) $d(x, y) = 0 \Leftrightarrow x = y$

(2) $d(x, y) = d(y, x)$

(3) $d(x, z) \leq d(x, y) + d(y, z)$

A metric space *is a tuple (X, d) where d is a metric on set X.*
A function d that satisfies properties (2) *and* (3) *but not* (1) *is called a* pseudometric*. A* pseudometric space *is a tuple (X, d) where d is a pseudometric on X.*

We usually refer to the elements of X as points. The first property is sometimes referred to as the *identity of indiscernibles*, i.e. if two points are indistinguishable by the metric, they should be considered as one and the same point. However, there might be settings in which it makes sense to allow distinct points to still have zero distance, giving rise to pseudometrics. The second property is called *symmetry* and states that the distance between two points is independent of any notion of direction. The third property is called *triangle inequality* and formalizes the idea that detours can never decrease the distance.

We can classify metric spaces according to two characteristics: whether they are finite or infinite, and whether they are discrete or continuous. We call a metric space (X, d) finite if X is finite, and infinite if X is infinite. The defining quality of a continuous metric space is that points can be arbitrarily close to each other: if (X, d) is continuous, then for any $x \in X$ and any $\delta > 0$, there exists $y \in X, y \neq x$, such that $d(x, y) < \delta$. On the other hand, in a discrete metric space, points are "isolated": there exists some minimum non-zero distance $\delta = \min\{d(x, y) \mid x, y \in X, x \neq y\} > 0$. Note that this implies that finite metric spaces cannot be continuous. However, infinite metric spaces can still be discrete.

Motivated by these distinctions, we also distinguish between discrete and continuous clustering settings. In *discrete clustering*, we only consider points from the input set P as possible centers. In *continuous clustering*, we assume that the input point set P is a subset of some infinite set X, and we can choose centers freely from X. Usually, this infinite set X is the d-dimensional Euclidean space $\mathbb{R}^d$, which is formally introduced in Section I.1.1.1.

Metrics can be represented in different ways. If $X = \{x_1, \ldots, x_n\}$ is finite, then (X, d) is necessarily discrete, and the simplest form of representation for d is an $n \times n$ matrix D, where entry D_{ij} contains $d(x_i, x_j)$. A related way of representing a metric is via a *graph metric*. To define this, we first need to formally introduce graphs and some adjacent concepts.

Definition 5 (Graphs, Subgraphs). *An* undirected graph *G is a tuple $G = (V(G), E(G))$ where $V(G)$ is a nonempty finite set, called the* vertices *of G, and $E(G) \subseteq \{X \subseteq V : |X| = 2\}$ a finite set called the* edges *of G. A* directed graph *is a graph $G = (V(G), E(G))$ where $E(G) \subseteq V \times V$, i.e. the edges are ordered tuples instead of sets.*
For a vertex v in a directed graph, the set of incoming edges *is defined as $\delta^-(v) = \{(x, y) \in E(G) \mid x \in V(G), y = v\}$, and the set of* outgoing edges *is defined as $\delta^+(v) = \{(x, y) \in E(G) \mid x = v, y \in V(G)\}$.*
A weighted *graph is a tuple (G, c) where $G = (V(G), E(G))$ is a graph and $c : E(G) \to \mathbb{R}_{\geq 0}$ a function that assigns a* weight *$c(e)$ to each edge $e \in E(G)$. A* subgraph *of G is a graph $H = (V(H), E(H))$ with $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$.*

Definition 6 (Shortest Paths). *Let $G = (V(G), E(G))$ be a graph. A (simple)* path *in G is a subgraph $P = (\{v_1, \ldots, v_{k+1}\}, \{e_1, \ldots, e_k\})$ such that*

(1) $v_i \neq v_j$ for $1 \leq i < j \leq k + 1$

(2) $e_i = \{v_i, v_{i+1}\} \in E(G)$ (or $e_i = (v_i, v_{i+1}) \in E(G)$ if G is directed) for $i = 1, \ldots, k$, and $e_i \neq e_j, 1 \leq i < j \leq k$.

We also refer to P as a v_1-v_{k+1}-path. If G is weighted, then we define the length *or* weight *of P as $c(P) = \sum_{i=1}^{k} c(e)$. If G is unweighted, then its length is defined as the number of edges in P.*
For two vertices $u, v \in V(G)$, a shortest u-v-path *is a path P in G that minimizes $c(P)$. We call the length of a shortest u-v-path the* distance *between u and v and use the shorthand* $\mathrm{dist}(u, v)$.

Note that we assume the edge weights in a weighted graph to be non-negative. In general, this does not have to be the case. But if there exists a cycle (i.e. a path with identical start and end vertex) of negative weight, shortest distances are not well-defined. One could traverse the negative cycle over and over, endlessly decreasing the distance. Therefore, it is often sensible to assume that no negative cycle with respect to the edge weights c exists. Since we only deal with non-negative distance functions (cf. Definition 1), we can also assume any distance function in a graph to be non-negative.

A weighted graph does not necessarily induce a metric on its own. To see this, take a look at Figure I.1.3a. If we simply try to take $d(u, v) = c(\{u, v\})$ for $u, v \in V$, we run into two kinds of problems. Not for every pair $u \neq v$ the corresponding edge $\{u, v\}$ has to be present in the graph. For these pairs, we are not able to define a distance right away. Additionally, the triangle inequality might be violated. In this specific example, $c(\{v_1, v_3\}) > c(\{v_1, v_2\}) + c(\{v_2, v_3\})$. But we can nevertheless get a (pseudo-)metric from *any* weighted graph by computing its metric closure.

Definition 7 (Metric Closure and Graph Metrics)**.** *Let (G, c) be a weighted undirected graph. For $u, v \in V(G)$, let $\mathrm{dist}(u, v)$ denote the shortest distance between u and v in G with respect to c. The* metric closure *of (G, c) is a weighted graph $(\overline{G}, \overline{c})$, where $V(\overline{G}) = V(G)$, $E(\overline{G}) = \{\{u, v\} \mid u, v \in V(G)\}$, and $\overline{c}(\{u, v\}) = \mathrm{dist}(u, v)$.*

In other words, we take a complete graph on $V(G)$ (i.e. one containing each possible edge) and set the weight of an edge to the minimum distance between its endpoints in the original graph (G, c). [2] Even in the case of an unweighted graph (which can be seen as a special case of a weighted graph where every edge weight is 1), taking the metric closure will always result in a metric distance function.

Observation 8. *Let (G, c) be an arbitrary weighted graph and $(\overline{G}, \overline{c})$ its metric closure. Then $\overline{c}$ defines a (pseudo-)metric distance function on $V(G)$.*

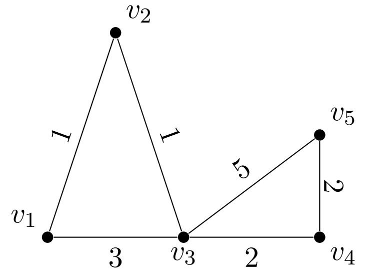
Proof. If $c(e) > 0$ for all $e \in E(G)$, then property (1) in Definition 4 is trivially fulfilled for $\overline{c}$ as well. If there are edges with weight 0, then we will also have edges with weight 0 in $\overline{G}$. In that case, (1) is not fulfilled, and we end up with a pseudometric.

Property (2) is also obviously fulfilled, since any shortest u-v-path is also a shortest v-u-path.

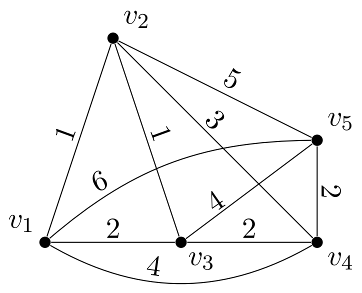
[2]Note that, according to the above definition, there would also be an edge $\{v, v\}$ with weight 0 for every $v \in V(G)$. For simplicity, these are omitted.

Let $u, v, w \in V(G)$ for which (3) is not fulfilled, i.e. $\overline{c}(u,w) > \overline{c}(u,v) + \overline{c}(v,w)$. But then the path resulting from taking the shortest u-v-path and then the shortest v-w-path is also a u-w-path of cost $\overline{c}(u,v) + \overline{c}(v,w)$. This is a contradiction. $\square$

Figure I.1.3b shows how the metric closure emerges from a given graph. Note that the resulting metric distance function can also be expressed as an $n \times n$-matrix D, where entry D_{ij} contains $\overline{c}(v_i, v_j)$, for any $v_i, v_j \in V(G)$ (see Figure I.1.3c).



(a) A weighted graph (G, c).



(b) The metric closure $(\overline{G}, \overline{c})$.

$$\begin{pmatrix} 0 & 1 & 2 & 4 & 6 \\ 1 & 0 & 1 & 3 & 5 \\ 2 & 1 & 0 & 2 & 4 \\ 4 & 3 & 2 & 0 & 2 \\ 6 & 5 & 4 & 2 & 0 \end{pmatrix}$$

(c) The distance matrix.

Figure I.1.3.: An example of a graph metric given via a weighted graph (G, c). Computing all shortest paths leads to the metric closure $(\overline{G}, \overline{c})$. The weights $\overline{c}$ define a metric distance function on $V(G)$.

I.1.1.1. Euclidean Space

Arguably the most important example for a continuous metric space is the Euclidean space with the Euclidean distance measure.

Definition 9 (Euclidean Space)**.** *The d-*dimensional Euclidean Space *is the metric space* $(\mathbb{R}^d, \text{dist})$*, where*

$$\mathbb{R}^d = \{(x_1, \ldots, x_n) \mid x_i \in \mathbb{R}, 1 \leq i \leq d\}$$

and

$$\text{dist} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}_{\geq 0}$$
$$\text{dist}(x, y) = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}.$$

The distance function dist *is called* Euclidean distance*.*

For $d = 2$ and $d = 3$, this corresponds to our intuitive understanding of “shortest distance” between two points in the plane and in 3-dimensional space.

Note that the standard inner product $\langle x, y \rangle = \sum_{i=1}^{d} x_i y_i$ on $\mathbb{R}^d$ induces the norm

$||x|| = \sqrt{\langle x, x\rangle}$. This means that we can write

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

For this reason, we usually make use of the notation $||x - y||$ when referring to the Euclidean distance between two points x and y.

I.1.2. Objective Functions

Judging the quality of a given solution is not always obvious. A common approach is the use of objective functions. In the following, fix a solution $S = (\mathcal{C}, \sigma)$ for the Center-Based k-Clustering Problem with centers $\mathcal{C} = \{c_1, \dots, c_k\}$ and clusters $\mathscr{C}_S = \{C_1, \dots, C_k\}$. Let $r_1, \dots, r_k$ be the radii of these clusters. Arguably one of the most extensively studied objectives is k-`Center`, where the goal is to minimize the maximum radius of the solution (cf. Figure I.1.4a). Note that this is identical to minimizing the maximum distance that any point has to its assigned center. More formally, the cost of S with respect to the k-`Center` objective is defined as

$$\texttt{kCenter}(S) = \max_{i\in[k]} r_i = \max_{p\in P} d(p, \sigma(p)).$$

This objective is popular because there are rather simple algorithms that yield good approximation ratios, even for some constrained versions (cf. Section I.3.1).

Two popular sum-based objectives are k-`Median` and k-`Means` (cf. Figure I.1.4b). In k-`Median`, the cost of S is given by the sum of all distances of points to their respective center, i.e.

$$\texttt{kMedian}(S) = \sum_{p\in P} d(p, \sigma(p)),$$

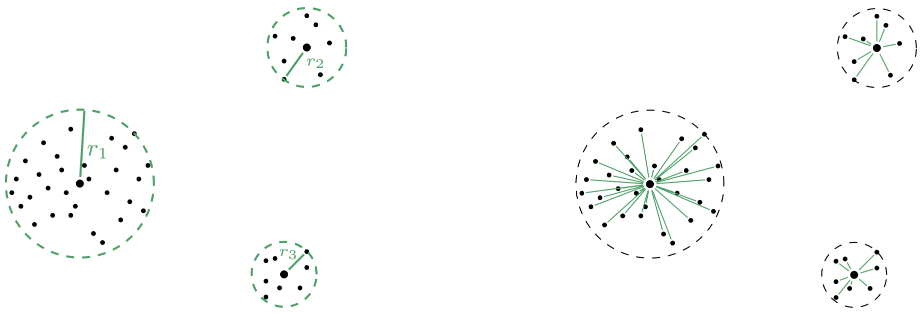
while in k-`Means` it is given by the sum of *squared* distances, i.e.

$$\texttt{kMeans}(S) = \sum_{p\in P} d(p, \sigma(p))^2.$$

Another objective function that has gained some traction in recent years (see e.g. [68], [55], [19], [35], [70], [32]) is k-`MinSumRadii`, which, at first, seems like a mixture between k-`Center` and k-`Median`. It judges a solution by the sum of the cluster radii (cf. Figure I.1.4a), i.e.

$$\texttt{kMSR}(S) = \sum_{i=1}^{k} r_i.$$

Although it is sum-based like k-`Median` and involves cluster radii like k-`Center`, it behaves noticeably different from both (cf. Section II.1.2). An illustration on how one solution can be judged differently by different objectives is depicted in Figure I.1.4.



(a) In k-`Center`, only the largest radius is of importance, i.e. $\texttt{kCenter}(S) = r_1$, whereas $\texttt{kMSR}(S) = r_1 + r_2 + r_3$.

(b) In k-`Median` and k-`Means`, each point contributes to the overall cost, which is $\texttt{kMedian}(S) = \sum_{p \in P} d(p, \sigma(p))$ and $\texttt{kMeans}(S) \sum_{p \in P} d(p, \sigma(p))^2$ respectively.

Figure I.1.4.: A solution for $k = 3$. The resulting partition can be judged by different objectives.

The work presented in this thesis is mainly concerned with k-`MinSumRadii` and k-`Means`, while k-`Center` is mainly relevant as a subproblem in obtaining an approximation algorithm for k-`MinSumRadii`.

It is easily possible to define the k-`Center` and k-`MinSumRadii` objectives for coverings. Given a covering $\mathscr{B} = \{\mathrm{B}(c_1, r_1), \ldots, \mathrm{B}(c_k, r_k)\}$, we have

$$\texttt{kCenter}(\mathscr{B}) = \max_{i \in [k]} r_i,$$

and analogously

$$\texttt{kMSR}(\mathscr{B}) = \sum_{i=1}^{k} r_i.$$

As illustrated in Figure I.1.2, a covering does not necessarily imply an assignment right away. However, in the absence of additional constraints (cf. Section I.3.1), we can always easily derive an assignment that does not increase the cost.

Observation 10. *Let $I = (P, d, k)$ be an instance of* Center-Based *k-*Clustering *and $\mathscr{B} = \{\mathrm{B}(c_1, r_1), \ldots, \mathrm{B}(c_k, r_k)\}$ a covering of P. Then we can turn $\mathscr{B}$ into a solution $S = (\mathcal{C}, \sigma)$ such that*

$$\textit{kCenter}(S) \leq \textit{kCenter}(\mathscr{B})$$

and

$$\textit{kMSR}(S) \leq \textit{kMSR}(\mathscr{B}).$$

Proof. Since $\mathscr{B}$ is a covering, for each point $p \in P$ there has to be at least one $\mathrm{B}(c_i, r_i) \in \mathscr{B}$ such that $p \in \mathrm{B}(c_i, r_i)$. We simply set $\sigma(p) = c_i$ (and choose arbitrarily if there are multiple). This way, we get a cluster $\sigma^{-1}(c_i)$ for every ball $\mathrm{B}(c_i, r_i) \in \mathscr{B}$. Consider the resulting solution $S = (\mathcal{C}, \sigma)$. Let $c_j \in \mathcal{C}$ be arbitrary and $C_j = \sigma^{-1}(c_j)$ be the cluster of

c_j. Let r'_j be the resulting radius of C_j. Each point that was assigned to c_j was already contained in the ball $\mathrm{B}(c_j, r_j)$ of $\mathscr{B}$. Therefore, we can conclude that $r'_j \leq r_j$. Thus, the maximum of the radii as well as their sum cannot increase. The claim follows. $\square$

I.2. Approximation Algorithms

Since we are judging the quality of solutions by objective functions, it is natural to formulate our Center-Based k-Clustering Problem as a *minimization* problem, i.e., with the goal to find a feasible solution that minimizes the given objective.

Except for a few special cases, clustering problems are usually NP-hard (see, e.g., [91, 8, 56, 65]). Therefore, there is little hope of finding efficient exact algorithms. A common approach is to resort to *approximation algorithms* instead.

Definition 11 (Approximation Algorithm)**.** *Let $\mathcal{P}$ be a minimization problem and $\mathcal{I}$ the set of all possible instances. Let $\mathtt{Opt}(I)$ denote the value of an optimal solution for instance $I \in \mathcal{I}$. For any algorithm A, let $A(I)$ denote the cost of the solution computed by A on input I.*
*An α-*approximation algorithm *for $\mathcal{P}$ is an algorithm A that runs in polynomial time in the input size and satisfies*

$$\sup_{I \in \mathcal{I}} \frac{A(I)}{\mathtt{Opt}(I)} \leq \alpha.$$

The number α is called the approximation factor *of A.*

In other words, an α-approximation algorithm returns – on any instance – a solution whose cost exceeds that of an optimal solution by a factor of at most α. Importantly, it does so with a polynomial runtime. So when using an approximation algorithm, we trade accuracy for efficiency.

As an introductory example to approximation algorithms, we will look at the optimization problem of minimizing the k-`Center` objective, which was shown to be NP-hard by Hochbaum [65].

Metric k-Center Problem

Input: A set P of n points, a metric distance function $d : P \times P \to \mathbb{R}_{\geq 0}$, and a number $k \in \mathbb{N}, k \leq n$

Output: A center set $\mathscr{C} = \{c_1, \ldots, c_k\}$ and an assignment $\sigma : P \to \mathscr{C}$ with induced radii $r_1, \ldots, r_k$, such that $\max_{i \in [k]} r_i$ is minimized

A well-known approximation algorithm for this problem is Gonzalez' Algorithm, also known as Farthest-First-Traversal Algorithm. It was introduced by Gonzalez in [58] and is known for its simplicity. It starts with an arbitrary point as the first center and then successively adds the point *farthest* away from the currently picked centers, until k centers are reached.

Algorithm 1: GONZALEZ (FARTHEST-FIRST-TRAVERSAL)

```
Input: Point set $P$, distance function $d$, integer $k$,
Output: Set of $k$ centers, assignment $\sigma$
1 Pick $c_1 \in P$ arbitrarily
2 for $i = 2, \dots, k$ do
3     $c_i \leftarrow \arg\max_{p \in P} \max_{c \in \{c_1, \dots, c_{i-1}\}} d(p, c)$
4 for $p \in P$ do
5     $\sigma(p) \leftarrow \arg\min_{c \in \{c_1, \dots, c_k\}} d(p, c)$
6 return $c_1, \dots, c_k$, $\sigma$
```

As Figure I.2.1 shows, the strategy of always choosing the point farthest away from all current centers naturally leads to suboptimal centers, even if we get the initial center choice right. Nevertheless, this simple method yields a good approximation ratio.

Lemma 12. *[58, 88] GONZALEZ' ALGORITHM is a 2-approximation algorithm for the METRIC* k-*CENTER PROBLEM and can be implemented to run in time $\mathcal{O}(nk)$.*

Since it can be shown that it is NP-hard to approximate METRIC k-CENTER within a factor of $(2 - \varepsilon)$ for any $\varepsilon > 0$ ([66], [65]), this is the best approximation ratio one can hope for (assuming $P \neq NP$). This farthest-first approach has also been utilized in other algorithms, e.g. by Dasgupta and Long [46] to obtain an approximation algorithm for hierarchical k-CENTER, and it also plays a role in our approximation for k-MIN-SUM-RADII presented in Chapter II.3.

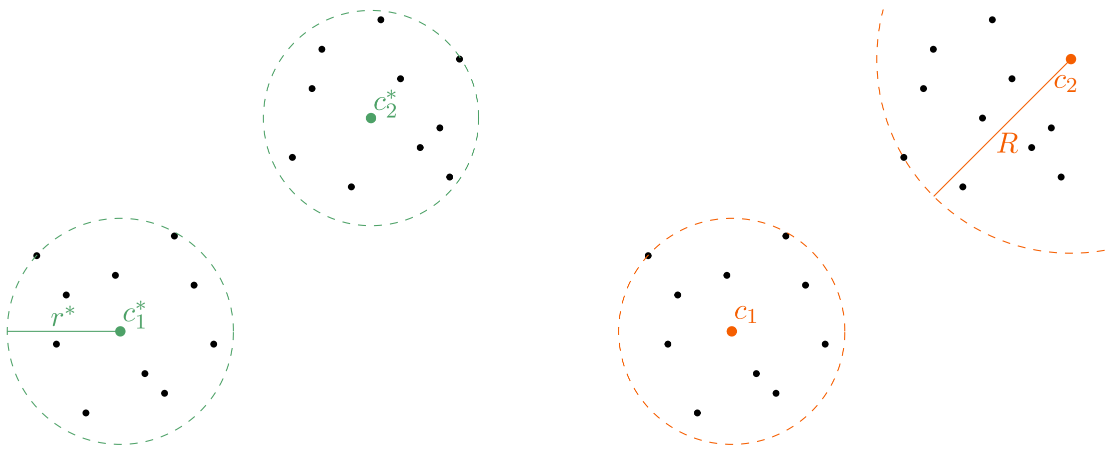
A noteworthy special case of an approximation algorithm is the so-called *Polynomial Time Approximation Scheme (PTAS).*

Definition 13 (PTAS)**.** *Let $\mathcal{P}$ be an optimization problem. An algorithm A is a* Polynomial Time Approximation Scheme (PTAS) *if it takes instance I of $\mathcal{P}$ and a number $\varepsilon > 0$ as input and, for each fixed ε, is a $(1 + \varepsilon)$-approximation algorithm for $\mathcal{P}$.*

The number ε can be viewed as the accuracy parameter of the PTAS, where smaller values lead to better solutions but higher runtimes. Most PTAS have a superpolynomial runtime dependency on $\frac{1}{\varepsilon}$, which is in compliance with the above definition, since A only has to have polynomial runtime for *fixed* ε. If the runtime is in fact also polynomial in $\frac{1}{\varepsilon}$, the approximation scheme is called *Fully Polynomial Time Approximation Scheme (FPTAS).*

Many approximation algorithms rely on randomization to obtain better approximation ratios than their deterministic counterparts. But if randomness is introduced, the notion of *approximation factor* needs to be adjusted. The most widely spread definition is the following.

Definition 14 (Expected Approximation Algorithm)**.** *Let $\mathcal{P}$ be a minimization problem and $\mathcal{I}$ the set of all possible instances. Let $\texttt{Opt}(I)$ denote the value of an optimal solution*



(a) Optimal solution with value r^*. The radius of the left cluster is slightly larger and therefore constitutes the solution value.

(b) A possible solution returned by Gonzalez' Algorithm. c_1 is chosen arbitrarily and c_2 is the point farthest away from it. The radius R of the right cluster now dominates the solution.

Figure I.2.1.: An example where Gonzalez' Algorithm does not compute an optimal solution. Even when the first chosen center happens to be optimal, choosing the second as the point farthest away gives a suboptimal center for the second cluster.

for instance $I \in \mathcal{I}$. For any algorithm A, let $E\left[A(I)\right]$ denote the expected value of the cost of the solution computed by A on input I. An expected α-approximation algorithm *for $\mathcal{P}$ is an algorithm A whose runtime is polynomial in the input size and which fulfills*

$$\sup_{I \in \mathcal{I}} \frac{E\left[A(I)\right]}{\mathtt{Opt}(I)} \leq \alpha.$$

I.2.1. FPT Approximation Algorithms

While the standard definition of approximation algorithms demands a polynomial runtime, there also exist approximate algorithms that have superpolynomial runtime. For example, the algorithms presented in Chapter II.2 and Chapter II.3 are so-called FPT approximation algorithms, where FPT stands for *fixed-parameter tractability*. The idea behind this concept is the following.

For some NP-hard optimization problems, we can identify a parameter k such that we can design algorithms that run in polynomial time if said parameter k is *fixed*. A famous example for this is the VERTEX COVER problem[1] with the size of the vertex cover as

[1] A vertex cover of a graph G is a set of vertices $X \subseteq V(G)$ such that each edge in $E(G)$ is incident to at least one vertex in X. The VERTEX COVER problem gets a graph G and a number k as input and asks whether there exists a vertex cover of G of size at most k.

parameter. A simple algorithm can decide whether there exists a vertex cover of size at most k in time $\mathcal{O}(2^k \cdot n)$. Thus, if k is small (e.g. at most $\log_2 n$), this is a polynomial time algorithm. We can formalize this concept as follows.

Definition 15 (Fixed-Parameter Tractability)**.** *Let $\mathcal{P}$ be a decision problem. $\mathcal{P}$ is called* fixed-parameter tractable *(FPT)* with parameter k *if there exists a parameter k such that there exists an algorithm A for $\mathcal{P}$ which can decide P and on every instance I runs in time*

$$f(k) \cdot \text{poly}(\textit{size}(I)),$$

where f is a computable function. We say that A is an FPT Algorithm.

If an algorithm is an FPT algorithm, we also say that it *runs in FPT time.* A more formal and thorough definition of the concept can for example be found in [43]. However, for the purpose of this thesis the above definition will suffice.

As the Vertex Cover example suggests, the original idea behind FPT algorithms was to find *exact* algorithms for hard problems. Of course, FPT algorithms can also be defined for optimization problems. For some particularly hard optimization problems, it has become a trend to design FPT *approximation* algorithms, i.e., algorithms that run in FPT time but may only return approximate solutions.

Definition 16 (FPT Approximation Algorithm)**.** *Let $\mathcal{P}$ be a minimization problem with parameter k and $\mathcal{I}$ the set of all possible instances. Let $\texttt{Opt}(I)$ denote the value of an optimal solution for instance $I \in \mathcal{I}$, and let $|I|$ denote the size of I. For any algorithm A, let $A(I)$ denote the cost of the solution computed by A on input I.*
An FPT α-approximation algorithm *for $\mathcal{P}$ is an algorithm A that runs in time $f(k) \cdot$* $\text{poly}(|I|)$ *and that satisfies*

$$\sup_{I \in \mathcal{I}} \frac{A(I)}{\texttt{Opt}(I)} \leq \alpha.$$

FPT approximations are usually applied in settings where common design techniques for polynomial time approximations do not work. Two examples are k-Median and k-Min-Sum-Radii with capacity constraints (cf. Section I.3.1). For example, Adamczyk et al. [4] gave an FPT $(7 + \varepsilon)$-appoximation for Capacitated k-Median, and Bandyapadhyay et al. an FPT $(15 + \varepsilon)$-approximation for Capacitated k-Min-Sum-Radii.

I.3. Further Basic Definitions

For the algorithms presented in Chapter II.3, we need to define some more concepts related to graphs. The first concerns connectivity and connected components.

Definition 17 (Connected Components)**.** *A graph G (directed or undirected) is called* connected *if there exists a u-v-path for all $u, v \in V(G)$. The maximal connected subgraphs of G are called its* connected components*, and we denote the set of connected components by $CC(G)$.*

An important graph class is that of bipartite graphs. They are defined by a vertex set that can be split in two, such that there are no edges within each subset.

Definition 18 (Bipartite Graphs)**.** *Let $G = (V(G), E(G))$ be an undirected graph. G is called* bipartite *if there exists a partition of $V(G)$ into two disjoint sets $V_1(G), V_2(G)$, such that each edge in $E(G)$ has one endpoint in $V_1(G)$ and one endpoint in $V_2(G)$.*

A classic example where bipartite graphs show up is finding non-conflicting pairings between two groups. This can be formalized by the concept of matchings.

Definition 19 (Matchings)**.** *Let $G = (V(G), E(G))$ be an undirected graph. A set $M \subseteq E(G)$ is called a* matching *if all edges in M are pairwise disjoint. If $|M| = |V(G)|/2$, then M is called* perfect*.*

Another type of graph that comes up frequently in applications is a *flow network*. Networks are especially useful, since many problems that relate to matching, assigning or routing can be reduced to finding a maximum flow in a flow network.

Definition 20 (Flow Networks)**.** *A* flow network *is a tuple (G, u, s, t) where*

- *$G = (V(G), E(G))$ is a directed graph*
- *$u : E(G) \to \mathbb{R}_{\geq 0}$ is a* capacity *function*
- *$s, t \in V(G)$ are two specified vertices called* source *and* sink*.*

An s-t-flow *in (G, u, s, t) is a function $f : E(G) \to \mathbb{R}_{\geq 0}$ that satisfies the following properties:*

- *$f(e) \leq u(e), \forall e \in E(G)$*
- *$\sum_{e \in \delta^-(v)} f(e) = \sum_{e \in \delta^+(v)} f(e)$, $\forall e \in V(G) \setminus \{s, t\}$.*

The value *of an s-t-flow f is defined as* $\mathrm{val}(f) = \sum_{e \in \delta^+(s)} f(s)$*, i.e. the outgoing flow of the sink. A* maximum flow *in (G, u, s, t) is a flow that has maximum value among all s-t-flows.*

I.3.1. Constrained Clustering

In some clustering applications, it is necessary to impose certain constraints on a feasible solution. In the following, we list some of the most commonly studied and the ones most relevant for the topics covered in this thesis.

Capacities A natural type of constraint is capacity. If the centers represent facilities of some sort, and the points potential clients, one might want to somewhat evenly distribute the clients among the facilities and thus ensure that none is overloaded. To this end, one defines a *capacity* $U \in \mathbb{R}_{\geq 0}$ and demands that $|\sigma^{-1}(c)| \leq U$ for all centers. A more general (and computationally even harder) scenario is that of *non-uniform capacities*. In this setting, any potential center c has its own dedicated capacity U_c, and we want a solution with $|\sigma^{-1}(c)| \leq U_c$ for all c. Generally, capacities tend to make clustering problems harder to approximate. While the unconstrained k-Center Problem allows for a 2-approximation (which is tight), the capacitated version can be shown to be NP-hard to approximate within a factor of 3 [44]. The natural LP formulation even has an unbounded integrality gap and only after some preprocessing to strengthen the LP can the gap be shown to be at most 9. At the same time, the gap of that improved formulation is at least 7. The best known approximation factor for uniform capacities is 6 [73], while for the non-uniform case it is 9 [11]. Likewise, for k-Median and k-Min-Sum-Radii the capacitated versions demand different approaches, e.g. resorting to FPT instead of polynomial time algorithms (cf. [4, 68, 19, 70]).

Lower Bounds Likewise, we can define clustering with lower bounds. In the uniform case, we have some global bound L and want to ensure that a solution $(\mathcal{C}, \sigma)$ satisfies $|\sigma^{-1}(c)| \geq L$. In the non-uniform case, we have center-specific values L_c and want to ensure that $|\sigma^{-1}(c)| \geq L_c$. Often, this is motivated by privacy concerns. If some clusters only contain very few points, information about cluster membership might be more telling. While this usually also leads to worse approximation guarantees[1] in comparison to unconstrained clustering, it usually allows for simpler algorithms and better approximation ratios than capacities [7, 32].

Fairness An emerging topic is that of algorithmic fairness. There exists a plethora of fairness notions, some of which aim at countering biases inherent to the data, while others aim at fair representation of certain groups in a solution. While we do not discuss all existent notions, there is a class of related fairness notions that has been used in multiple papers, which is of special importance in this thesis. We subsume this class under the title of *Group Fairness* and define different variants in the following. In all cases, we assume that our input points come from a predefined set of protected groups, encoded as *colors*. Each point belongs to exactly one color, and the goal is to (exactly or approximately) preserve the ratios of colors in the overall population in every cluster or among the centers. More formally, the input additionally contains a set of colors

[1]with k-`Center` being the exception

$\Gamma = \{1, \ldots, m\}$ and a *coloring* $\gamma : P \to \{1, \ldots, m\}$. For a point set $P' \subseteq P$ and a color j, we define $\Gamma_j(P') = \gamma^{-1}(j) \cap P'$, i.e. the set of all points in P' of color j.

Definition 21 (Fairness Notions)**.** *Let $I = (P, d, k)$ be an instance of the* Center Based *k-*Clustering Problem*. Let $\Gamma = \{1, \ldots, m\}$ be a set of colors and $\gamma : P \to \{1, \ldots, m\}$ be a* coloring *of P. Let $S = (\mathcal{C}, \sigma)$ be a solution with clusters $\mathscr{C}_S = \{C_1, \ldots, C_k\}$. We distinguish the following fairness notions:*

1. *Exact Fairness [23]: S satisfies* exact fairness *if*
$$\frac{|\Gamma_j(C_i)|}{|C_i|} = \frac{|\Gamma_j(P)|}{|P|}, \forall i \in [k], \forall j \in [m].$$

2. *(ℓ, u)-Fairness [22]: The input additionally contains lower and upper ratio bounds $\ell = (\ell_1, \ldots, \ell_m)$ and $u = (u_1, \ldots, u_m)$. S satisfies* (ℓ, u)-fairness *if*
$$\ell_j \leq \frac{|\Gamma_j(C_i)|}{|C_i|} \leq u_j, \quad \forall i \in [k], j \in [m].$$

3. *Ratio Balance [36]: If $m = 2$, i.e. there are only two colors, and we are additionally given a parameter $b \in [0, 1]$, S is* b-balanced *if*
$$\mathrm{bal}(C_i) := \min \left\{ \frac{|\Gamma_1(C_i)|}{|\Gamma_2(C_i)|}, \frac{|\Gamma_2(C_i)|}{|\Gamma_1(C_i)|} \right\} \geq b, \quad \forall i \in [k].$$

4. *Exact Balance [29]: S is said to satisfy* exact balance *if we have*
$$|\Gamma_1(C_i)| = |\Gamma_2(C_i)| = \ldots = |\Gamma_m(C_i)|, \quad \forall i \in [k].$$

5. *Center Fairness [74]: For every color, we are additionally given lower and upper ratio bounds $\ell = (\ell_1, \ldots, \ell_m)$ and $u = (u_1, \ldots, u_m)$. S satisfies* center fairness *if*
$$\ell_j \leq |\Gamma_j(\mathcal{C})| \leq u_j, \forall j \in [m].$$

Many of the fairness notions as well as lower bounds share a property. If any two (disjoint) clusters satisfy the constraint, then their union will also satisfy it.

Definition 22. *(Mergeable Constraint) Let C_i and C_j be two disjoint clusters of a solution to some k-*Clustering *Problem. A constraint is called* mergeable *if the following holds: if C_i and C_j both satisfy the constraint, then $C_i \cup C_j$ also satisfies the constraint.*

This is of special interest in the design of approximation algorithms as it allows for merging procedures like the one employed in the algorithm in Chapter II.3.

Observation 23. *The following constraints are mergeable:*

1. *(Uniform) Lower Bounds*

2. *Exact Fairness*
3. *(ℓ, u)-Fairness*
4. *Ratio Balance*
5. *Exact Balance.*

Proof. For all these constraints we have to show that the union of any two disjoint feasible clusters remains feasible.

1. If we have a uniform lower bound L, and we have two disjoint clusters C_i and C_j with $|C_i| \geq L$ and $|C_j| \geq L$, then clearly $|C_i \cup C_j| \geq 2L > L$. If we have non-uniform lower bounds, then let C_i and C_j be disjoint clusters with centers c_i and c_j and $|C_i| \geq L_{c_i}$ and $|C_j| \geq L_{c_j}$. Then clearly $|C_i \cup C_j| \geq \max\{L_{c_i}, L_{c_j}\}$.

2. Consider two disjoint clusters C_i and C_j with
$$\frac{|\Gamma_\ell(C_i)|}{|C_i|} = \frac{|\Gamma_\ell(P)|}{|P|} = \frac{|\Gamma_\ell(C_j)|}{|C_j|}$$
for all $\ell \in [m]$. Then for their union $C_i \cup C_j$ we have
$$\frac{|\Gamma_\ell(C_i \cup C_j)|}{|C_i \cup C_j|} = \frac{|\Gamma_\ell(C_i)| + |\Gamma_\ell(C_j)|}{|C_i| + |C_j|}.$$
Now observe that for $a, b, c, d \in \mathbb{R}^+$ with $\frac{a}{b} = \frac{c}{d}$, it holds that $\frac{a+c}{b+d} = \frac{a}{b}$, and therefore we have
$$\frac{|\Gamma_\ell(C_i)| + |\Gamma_\ell(C_j)|}{|C_i| + |C_j|} = \frac{|\Gamma_\ell(C_j)|}{|C_j|}.$$

3. Let $j \in [m]$ be an arbitrary color and C_1, C_2 be disjoint (ℓ, u)-fair clusters, i.e.
$$\ell_j \leq \frac{|\Gamma_j(C_1)|}{|C_1|} \leq u_j \text{ and } \ell_j \leq \frac{|\Gamma_j(C_2)|}{|C_2|} \leq u_j.$$
For their union we then have
$$\ell_j|C_1 \cup C_2| = \ell_j|C_1| + \ell_j|C_2| \leq |\Gamma_j(C_1)| + |\Gamma_j(C_2)| = |\Gamma_j(C_1 \cup C_2)|,$$
which implies
$$\ell_j \leq \frac{|\Gamma_j(C_1 \cup C_2)|}{|C_1 \cup C_2|}.$$
Analogously, we have
$$u_j|C_1 \cup C_2| = u_j|C_1| + u_j|C_2| \geq |\Gamma_j(C_1)| + |\Gamma_j(C_2)| = |\Gamma_j(C_1 \cup C_2)|,$$

implying

$$u_j \geq \frac{|\Gamma_j(C_1 \cup C_2)|}{|C_1 \cup C_2|}.$$

4. Let C be a cluster with $\mathrm{bal}(C) \geq b$. We first argue that we always have

$$b \leq \frac{|\Gamma_1(C)|}{|\Gamma_2(C)|} \leq \frac{1}{b}.$$

We immediately get the first inequality from the assumption $\mathrm{bal}(C) \geq b$. To see that the second quality holds, observe that, by the definition of $\mathrm{bal}(C)$, we have $\frac{|\Gamma_2(C)|}{|\Gamma_1(C)|} \geq b$ as well. This implies $\frac{|\Gamma_1(C)|}{|\Gamma_2(C)|} \leq \frac{1}{b}$.

Now let C_i, C_j be disjoint clusters with $\mathrm{bal}(C_i) \geq b$ and $\mathrm{bal}(C_j) \geq b$. From the above observation we can conclude that

$$b \leq \frac{|\Gamma_1(C_i)|}{|\Gamma_2(C_i)|} \leq \frac{1}{b} \quad \text{and} \quad b \leq \frac{|\Gamma_1(C_j)|}{|\Gamma_2(C_j)|} \leq \frac{1}{b}.$$

From this, it follows that $|\Gamma_1(C_i)| \geq b|\Gamma_2(C_i)|$ and $|\Gamma_1(C_j)| \geq b|\Gamma_2(C_j)|$, as well as $|\Gamma_1(C_i)| \leq \frac{1}{b}|\Gamma_2(C_i)|$ and $|\Gamma_1(C_j)| \leq \frac{1}{b}|\Gamma_2(C_j)|$. Summing up the first two inequalities, we get $|\Gamma_1(C_i)| + |\Gamma_1(C_j)| \geq b\,(|\Gamma_2(C_i)| + |\Gamma_2(C_j)|)$, which implies

$$\frac{|\Gamma_1(C_i \cup C_j)|}{|\Gamma_2(C_i \cup C_j)|} = \frac{|\Gamma_1(C_i)| + |\Gamma_1(C_j)|}{|\Gamma_2(C_i)| + |\Gamma_2(C_j)|} \geq b.$$

Likewise, we can sum up the latter two inequalities above to get $|\Gamma_1(C_i)| + |\Gamma_1(C_j)| \leq \frac{1}{b}\,(|\Gamma_2(C_i)| + |\Gamma_2(C_j)|)$, which can be rewritten as

$$\frac{|\Gamma_1(C_i \cup C_j)|}{|\Gamma_2(C_i \cup C_j)|} = \frac{|\Gamma_1(C_i)| + |\Gamma_1(C_j)|}{|\Gamma_2(C_i)| + |\Gamma_2(C_j)|} \leq \frac{1}{b}.$$

Together, these two inequalities imply

$$b \leq \frac{|\Gamma_1(C_i \cup C_j)|}{|\Gamma_2(C_i \cup C_j)|} \leq \frac{1}{b} \quad \text{and} \quad b \leq \frac{|\Gamma_2(C_i \cup C_j)|}{|\Gamma_1(C_i \cup C_j)|} \leq \frac{1}{b},$$

which lets us conclude that

$$\mathrm{balance}(C_i \cup C_j) = \min\left\{\frac{|\Gamma_1(C_i \cup C_j)|}{|\Gamma_2(C_i \cup C_j)|}, \frac{|\Gamma_2(C_i \cup C_j)|}{|\Gamma_1(C_i \cup C_j)|}\right\} \geq b.$$

5. Let C_1, C_2 be disjoint clusters with $|\Gamma_i(C_1)| = |\Gamma_j(C_1)|$ and $|\Gamma_i(C_2)| = |\Gamma_j(C_2)|$ for

all $i, j \in [m], i \neq j$. Then for $C_1 \cup C_2$ and for any two distinct colors i, j we have

$$\begin{aligned} |\Gamma_i(C_1 \cup C_2)| &= |\Gamma_i(C_1)| + |\Gamma_i(C_2)| \\ &= |\Gamma_j(C_1)| + |\Gamma_j(C_2)| \\ &= |\Gamma_j(C_1 \cup C_2)|. \end{aligned}$$

□

Part II.

Minimum Sum-of-Radii Clustering

II.1. Basics of k-Min-Sum-Radii Clustering

In this part, we explore the k-`MinSumRadii` objective and present FPT approximation algorithms for different versions of the k-Min-Sum-Radii Problem. First, we look at some properties of the objective and how it relates to (and differs from) other clustering objectives. Afterwards, we give an overview of hardness and approximability results.

In Chapter II.2, we then present an FPT $(1+\varepsilon)$-approximation for Euclidean k-Min-Sum-Radii, based on the paper *Approximating Fair k-Min-Sum-Radii in Euclidean Space* [47] by Drexler, Hennes, Lahiri, Schmidt and Wargalla. The algorithm presented in the original paper was claimed to be a $(1+\varepsilon)$-approximation for k-Min-Sum-Radii with mergeable constraints, but it turned out to contain a fatal mistake that is not trivial to fix. Therefore, we show an altered version here that only deals with the unconstrained problem.

In Chapter II.3, we present an FPT constant-factor approximation for the k-Min-Sum-Radii Problem with mergeable constraints in general metrics, based on the paper *FPT Approximations for Fair k-Min-Sum-Radii* [33] by Carta, Drexler, Hennes, Rösner and Schmidt. First, we show the algorithm as presented in that paper, which yields a $(6+\varepsilon)$-approximation for general mergeable constraints. Additionally, we show how an insight that was presented in [18] can be applied to our algorithm as well, leading to a $(4+\varepsilon)$-approximation. Finally, we present $(3+\varepsilon)$-approximations for two special cases of mergeable constraints.

II.1.1. Definitions

As defined in Definition 1, a solution S consists of a center set $\mathcal{C} = \{c_1, \dots, c_k\}$ of size at most k and an assignment $\sigma : P \to \mathcal{C}$, which induce clusters $C_1, \dots, C_k$ and radii $r_1, \dots, r_k$. The formal definition of the k-Min-Sum-Radii optimization problem can now be given as follows.

Metric k-Min-Sum-Radii Problem

Input: A set P of n points, a metric distance function $\mathrm{dist} : P \times P \to \mathbb{R}_{\geq 0}$, and a number $k \in \mathbb{N}, k \leq n$

Output: A center set $\mathcal{C} = \{c_1, \dots, c_k\}$ and an assignment $\sigma : P \to \mathcal{C}$, such that the induced radii $r_1, \dots, r_k$ minimize $\sum_{i=1}^{k} r_i$

Note that we make use of distinct notations for the objective (k-`MinSumRadii`) and the actual optimization problem (k-Min-Sum-Radii). We will keep this convention for other objectives as well. Since the k-`MinSumRadii` objective sums up these radii, they are the main point of interest. This motivates the following definition.

Definition 24 (Radius Profile)**.** *Let S be a solution to the k-*Min-Sum-Radii Problem* *with centers $c_1, \dots, c_k$ and induced radii $r_1, \dots, r_k$. Without loss of generality, assume that $r_1 \geq r_2 \geq \dots \geq r_k$. Then the tuple $(r_1, \dots, r_k)$ is called the* radius profile *of S. Let $(r_1^*, \dots, r_k^*)$ be the radius profile of an optimal solution. A radius profile $(r_1, \dots, r_k)$ is called* near-optimal *if $r_i^* \leq r_i \leq (1+\varepsilon)r_i^*, \forall i \in [k]$.*

Note that, while each solution has a unique radius profile, a center set along with a radius profile does not necessarily define a feasible solution. If the centers and radii are chosen unfavorably, the corresponding balls might not cover the instance. But even if they do, it might not imply a unique solution (in the sense of Definition 1). If, for example, some of the corresponding balls overlap, it is not clear to which cluster the points in the intersection belong. This becomes especially relevant when imposing additional constraints like lower bounds, capacities or fairness on the clusters. Instead, a center set combined with a radius profile defines a *covering* (cf. Definition 3) in this case.

An important special case of Metric k-Min-Sum-Radii is the Euclidean version, where the underlying metric space is the d-dimensional Euclidean space $\mathbb{R}^d$. Furthermore, we consider the continuous clustering version, where centers can be chosen freely from $\mathbb{R}^d$.

Euclidean k-Min-Sum-Radii Problem

Input: A set $P \subset \mathbb{R}^d$ of n points, a number $k \in \mathbb{N}, k \leq n$

Output: A center set $\mathcal{C} = \{c_1, \dots, c_k\} \subset \mathbb{R}^d$ and an assignment $\sigma : P \to \mathcal{C}$, such that the induced radii $r_1, \dots, r_k$ minimize $\sum_{i=1}^k r_i$

This setting is explored in more detail in Chapter II.2, where we derive a $(1+\varepsilon)$-approximation that runs in FPT time.

In the following, we explore some properties of the objective and how it differs from more common objectives like k-`Center` and k-`Median`/k-`Means`.

II.1.2. Properties of the k-Min-Sum-Radii Objective

As mentioned above, k-`MinSumRadii` behaves quite differently from k-`Center` or k-`Median`/k-`Means`. For the latter three, we can make the following simple observation: partitioning the points into many small clusters is cheaper than partitioning it into fewer large clusters. For k-`MinSumRadii`, this is not true. If we take a look at Figure II.1.1, we get an idea why this is the case. It displays two different solutions for $k = 4$ on the same point set. On the left, we see a good solution for the k-`Center` objective. Since k-`Center` only cares about the *maximum* radius in a solution, four equally sized clusters are fine. But if we evaluate this solution with the k-`MinSumRadii` objective, we have to pay for *all* four radii. It would in fact be cheaper to simply cover the whole instance with just one large cluster. In k-`MinSumRadii`, we therefore often encounter solutions like the one displayed in Figure II.1.1b: three of the four clusters are *singletons*, i.e., they

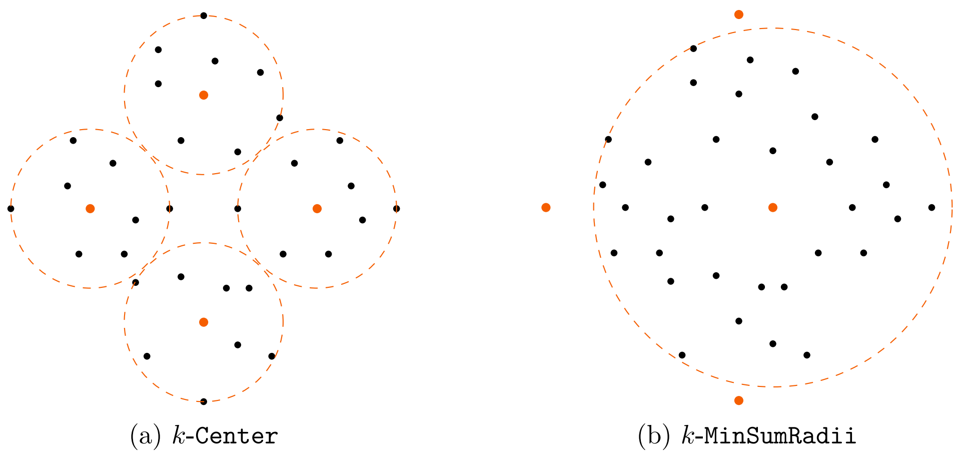


Figure II.1.1.: Two solutions for the same point set and $k = 4$. The k-`Center` solution has four equally sized clusters. However, the sum of the radii of these clusters is larger than the diameter of the entire point set. Therefore, it is a bad solution for k-`MinSumRadii`. A good k-`MinSumRadii` solution would choose three of the points at the fringes as singleton clusters (i.e., with radius 0) and cover the remaining points with one large cluster.

contain only the center itself and therefore have radius 0. The rest of the solution is then covered by one large ball, whose radius is significantly smaller than the sum of radii of the k-`Center` solution. Because of this, some techniques that are commonly used in other clustering objectives do not work here. One prominent example is clustering under fairness constraints (cf. Section I.3.1). For k-Center or k-Median, there exist approximation algorithms for some fairness settings. Most of these rely on a so-called *fairlet decomposition*, a concept which was first introduced by Chierichetti et al. [36]. A *fairlet* is a (cardinality-wise) minimal set of points that maintains fairness, i.e., represents all colors with the same proportion as in the overall population. A fairlet decomposition (as the name suggests) is a partition of the input points into such fairlets. Each of these fairlets then gets a representative, and the algorithms proceed by clustering these representatives. The fairlets are treated as inseparable entities, only ever getting assigned as a whole. Treating these representatives as centers, the fairlet decomposition can be viewed as a k'-clustering, where k' is the number of fairlets. A key argument why this leads to good approximations in k-Center and k-Median is the fact that we can bound the cost of an optimal fairlet decomposition by twice the cost of an optimum solution. For k-`MinSumRadii` however, it can be much more expensive, as Figure II.1.2 illustrates.

Another property that makes k-`MinSumRadii` stand out is that assigning points optimally to a given set of centers is hard. For the other objectives, it is always optimal to assign each point to its closest center – at least in unconstrained clustering. When dealing with capacities, lower bounds or fairness constraints, more elaborate assignment strategies are needed. But for k-`MinSumRadii`, assigning points to their closest centers can lead to very bad results, even in the unconstrained case. This is discussed in more

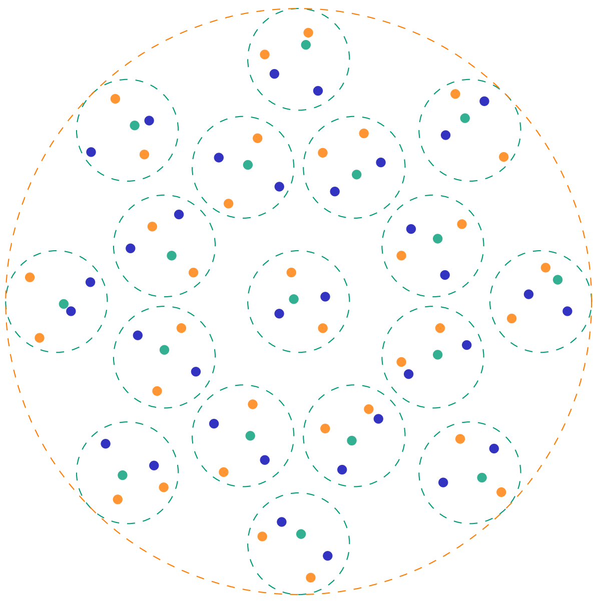


Figure II.1.2.: A fairlet decomposition for 3 colors with ratios 2:2:1. The sum of the radii of the fairlets is much larger than the radius of the large orange cluster.

detail in Section II.1.2.2.

One useful property that we use in the design of our algorithms is that we can assume that the solution we are trying to approximate does not contain arbitrarily small radii, without incurring too much of a loss. We first need the following definition.

Definition 25. *Let $\varepsilon > 0$. A covering $\{\mathrm{B}(c_1, r_1), \ldots, \mathrm{B}(c_k, r_k)\}$ of P is* ε-balanced *if $r_i \geq \frac{\varepsilon}{k} \max_j r_j$ for all $i \in \{1, \ldots, k\}$.*

We can now prove that, for any covering (and therefore for any feasible solution), there exists an ε-balanced covering whose cost exceeds the cost of the original solution by just an ε-fraction.

Observation 26. *For any covering $\mathscr{B} = \{\mathrm{B}(c_1, r_1), \ldots, \mathrm{B}(c_k, r_k)\}$ of P, there exists an ε-balanced covering $\mathscr{B}' = \{\mathrm{B}(c_1, r_1'), \ldots, \mathrm{B}(c_k, r_k')\}$, such that $\texttt{kMSR}(\mathscr{B}') \leq (1 + \varepsilon)\texttt{kMSR}(\mathscr{B})$.*

Proof. We obtain $\mathscr{B}'$ by setting

$$r_i' = \begin{cases} r_i, & \text{if } r_i \geq \frac{\varepsilon}{k} r_1 \\ \frac{\varepsilon}{k} r_1, & \text{otherwise.} \end{cases}$$

In other words, we "round up" all radii that are too small and do not change the rest of them. This increases the cost by at most $\frac{\varepsilon}{k} r_1$ per cluster. As there are at most k clusters, the total cost increase (with respect to the kMSR objective) is at most $\varepsilon r_1 \leq \varepsilon \cdot \texttt{kMSR}(\mathscr{B})$. □

This is useful since for a key technique used in our algorithms we need the assumption that the solution we approximate is ε-balanced (cf. Subsection II.1.3).

A similar property states that, for every optimal solution, there always exists a solution whose balls are well-separated and whose cost does not exceed the optimal cost by too much. This property was initially introduced and proven in [47] to obtain a $(1+\varepsilon)$-approximation for Euclidean k-Min-Sum-Radii with fairness constraints. However, the analysis of said algorithm turned out to be flawed which is why, in this thesis, we only present a $(1+\varepsilon)$-approximation for the unconstrained problem. But the property might still be of interest, so we state and prove it here.

Definition 27 (γ-separated Solution). *Let $\gamma \geq 1$. Two balls $\mathrm{B}(c_1, r_1)$ and $\mathrm{B}(c_2, r_2)$ are said to be* γ-separated *if* $\mathrm{B}(c_1, \gamma r_1) \cap \mathrm{B}(c_2, \gamma r_2) = \emptyset$. *A covering is* γ-separated *if all its balls are pairwise γ-separated.*

We first show that, in the continuous Euclidean setting, any solution can be turned into a γ-separated solution without incurring too much of a loss.

Lemma 28. *Let S be a solution for Euclidean k-Min-Sum-Radii on P and $\mathscr{C}_S = \{C_1, \ldots, C_k\}$ be the clusters of S. Then for all $\gamma \geq 1$, there exists a γ-separated covering $\mathscr{B} = \{B_1, \ldots, B_{k'}\}$ of P with $k' \leq k$ and $\mathtt{kMSR}(\mathscr{B}) \leq \gamma^{k-1}\mathtt{kMSR}(S)$.*

Proof. Starting with $\{\mathrm{MB}(C_1), \ldots, \mathrm{MB}(C_k)\}$, we construct $\mathscr{B}$ from $\mathscr{C}$ by successively merging balls that are too close to each other. More precisely, if there exist C_i, C_j such that $\mathrm{MB}_\gamma(C_i) \cap \mathrm{MB}_\gamma(C_j) \neq \emptyset$, we remove $\mathrm{MB}(C_i)$ and $\mathrm{MB}(C_j)$ from our covering and add $\mathrm{MB}(C_i \cup C_j)$. Of course, this new ball can now incur conflicts with other balls, so we might need to repeat this procedure iteratively. The key insight is that each time two balls are merged, the number of balls in the covering decreases by one. So this procedure can be repeated at most $k-1$ times.

Now we bound the cost that one such merge incurs. Let $\mathscr{B} = \{B_1, \ldots, B_{k'}\}$ denote the covering that has been constructed up to this point, where $B_i = \mathrm{B}(c_i, r_i), \forall i \in [k']$. Construct a graph G on top of $\mathscr{B}$, where two balls B_i and B_j are connected if and only if $\|c_i - c_j\| \leq \gamma(r_i + r_j)$. In other words, two balls are connected by an edge if and only if they are not γ-separated. We try to construct a γ-separated covering by merging all balls that belong to the same connected component. This just means that we replace all balls of the connected component with the minimal-enclosing-ball of the connected component. Take any connected component Z of G and consider two arbitrary points $x, y \in \bigcup_{B_\lambda \in Z} B_\lambda$, say $x \in B_i$ and $y \in B_j$. We can upper-bound the distance between them as follows: For any path $B_i = B_{i_0}, \ldots B_{i_\ell} = B_j$ in G that connects B_i and B_j we have

$$\begin{aligned}\|x-y\| &\leq \|x - c_i\| + \sum_{\lambda=0}^{\ell-1} \|c_{i_\lambda} - c_{i_{\lambda+1}}\| + \|y - c_j\| \\ &\leq r_i + r_j + \sum_{\lambda=0}^{\ell-1} \gamma(r_{i_\lambda} + r_{i_{\lambda+1}}) \leq \gamma \sum_{B_\lambda \in Z} 2r_\lambda\end{aligned}$$

In other words, the radius of the resulting ball is larger than the sum of the previous radii by a factor of at most γ. As argued above, this step can be repeated at most $k-1$ times, each time increasing the cost by a factor of at most γ. It follows that the cost of the resulting covering $\mathscr{B}$ can be at most $\gamma^{k-1}\texttt{kMSR}(S)$. □

We can even show that both γ-separatedness and ε-balancedness can be achieved simultaneously without increasing the cost too much.

Lemma 29. *Let S be a solution for* Euclidean k-Min-Sum-Radii *on P and $\mathscr{C}_S = \{C_1, \ldots, C_k\}$ be the clusters of S. Then for all $\varepsilon > 0$ and $\gamma \geq 1$, there exists an ε-balanced and γ-separated covering $\mathscr{B} = \{B_1, \ldots, B_{k'}\}$ of P with $k' \leq k$ and $\texttt{kMSR}(\mathscr{B}) \leq (1+\varepsilon)^k\gamma^{k-1}\texttt{kMSR}(S)$. Additionally, if S satisfies a given mergeable constraint, then so does the corresponding clustering $\{B_1 \cap P, \ldots, B_{k'} \cap P\}$.*

Proof. Starting with $\{\mathrm{MB}(C_1), \ldots, \mathrm{MB}(C_k)\}$, we construct $\mathscr{B}$ from $\mathscr{C}$ in phases consisting of two steps: (1) merge balls that are currently too close to each other and thus not γ-separated, (2) ensure that the current covering is ε-balanced by increasing the radii of balls that are too small. Both steps are alternately applied in phases until the resulting clustering is both ε-balanced and γ-separated. Now, even though step (1) might yield a covering that is neither ε-balanced nor γ-separated and step (2) might yield a clustering that is not γ-separated, since the number of balls reduces with every phase, except maybe the first, there can only be k phases altogether. By Lemma 28, step (1) increases the cost by a factor of at most γ. By Observation 26, step (2) will increase the cost by a factor of at most $(1+\varepsilon)$. The resulting covering will therefore cost at most $(1+\varepsilon)^k\gamma^{k-1}\texttt{kMSR}(S)$. □

Connections to k-Center An important property of the k-`Center` objective is that the optimal solution value is always one of the $\mathcal{O}(n^2)$ many pariwise point distances. This is famously used by Hochbaum and Shmoys in their 2-approximation [65]. Similarly, we can argue that *all* radii in any optimal k-Min-Sum-Radii solution correspond to pairwise point distances as well.

Observation 30. *Let $I = (P, d, k)$ be an instance for (discrete)* k-Min-Sum-Radii *and let S^* be an optimal solution with radius profile $(r_1^*, \ldots, r_k^*)$. Then $r_i^* \in D := \{d(p,q) \mid p, q \in P, p \neq q\} \cup \{0\}$ for all $i \in [k]$.*

Proof. Assume that there exists an optimal solution S^* where one of the induced radii, say r_i^*, does not correspond to a distance in the set D. Consider the corresponding ball $B(c_i^*, r_i^*)$. Let $q = \arg\max_{p \in B(c_i^*, r_i^*)} d(c_i^*, q)$ be the point farthest away from c_i^* in that ball and $d_q = d(c_i^*, q)$ its distance from c_i^*. Since r_i^* is not a pairwise point distance, we must have $r_i^* > d_q$. At the same time, $B(c_i^*, d_q)$ contains the exact same points as $B(c_i^*, r_i^*)$ (by choice of q). So replacing r_i^* by d_q yields a feasible solution with strictly smaller cost, contradicting the optimality of S^*. □

As illustrated in Figure II.1.1, good solutions can vary significantly depending on the objective. Nevertheless, we can prove the following relationship between an approximate value for k-Center and the *largest* radius in an optimal solution to k-Min-Sum-Radii.

Lemma 31. *Let I be an instance for Center Based k-Clustering. Let S_{kc} be an α-approximate* k-Center *solution with value $r_\alpha := \texttt{kCenter}(S_{kc})$, and let S^*_{msr} be an optimum* k-Min-Sum-Radii *solution with largest radius r_1^*. Then it holds that*

$$r_1^* \in \left[\frac{r_\alpha}{\alpha}, k \cdot r_\alpha\right].$$

Proof. Let S^*_{kc} be an *optimal* k-Center solution with value $\texttt{kCenter}(S^*_{\text{kc}}) = r_c$. Notice that any feasible k-Center solution is also a feasible k-Min-Sum-Radii solution and vice versa. As r_c is the radius of a largest cluster in the optimal k-center solution, we have $r_1^* \geq r_c$, as otherwise $\texttt{kCenter}(S^*_{\text{msr}}) < \texttt{kCenter}(S^*_{\text{kc}})$, contradicting the optimality of S^*_{kc}. Since r_α is an α-approximation for r_c (i.e., $r_\alpha \leq \alpha \cdot r_c$), we get $r_1^* \geq r_c \geq r_\alpha/\alpha$.

On the other hand, we must have $\texttt{kMSR}(S_{\text{kc}}) \leq k \cdot r_\alpha$, as we have k clusters and all their radii are at most r_α. For the largest radius r_1^* in S^*_{msr} we can therefore argue that $r_1^* \leq k \cdot r_\alpha$. For the sake of contradiction, assume $r_1^* > k \cdot r_\alpha$. Then $\texttt{kMSR}(S^*_{\text{msr}}) \geq r_1^* > k \cdot r_\alpha \geq \texttt{kMSR}(S_{\text{kc}})$, contradicting the optimality of S^*_{msr}.

In conclusion, we have $\frac{r_\alpha}{\alpha} \leq r_1^* \leq k \cdot r_\alpha$, proving the claim.

□

In Section II.1.3.1, we show how this property can be exploited to find candidate sets for radius profiles in the FPT algorithms presented in Chapter II.2 and Chapter II.3.

II.1.2.1. Hardness and Algorithms

The k-Min-Sum-Radii Problem admits a polynomially sized LP formulation that plays a key role in the design of some approximation algorithms. The standard formulation utilizes Observation 30, which states that every optimal solution will only have radii that correspond to pairwise distances of input points. For the LP, we can thus pre-compute a set of *sensible* balls as follows. For each point $p \in P$, compute $D_p = \{d(p, x) \mid x \in P\}$. This allows us to compute the set

$$\mathcal{B} = \{(c, r) \mid c \in P, r \in D_c\},$$

i.e. make a pair for each possible center with each of its sensible radii. This gives rise to the following ILP.

$$\begin{aligned}
\min \quad & \sum_{(c,r)\in\mathcal{B}} r x_{(c,r)} && && \text{(II.1.1)}\\
\text{subject to} \quad & \sum_{(c,r)\in\mathcal{B}:d(c,p)\le r} x_{(c,r)} \ge 1, && \forall p \in P && \text{(II.1.2)}\\
& \sum_{(c,r)\in\mathcal{B}} x_{(c,r)} \le k && && \text{(II.1.3)}\\
& x_{(c,r)} \in \{0,1\}, && \forall (c,r) \in \mathcal{B} && \text{(II.1.4)}
\end{aligned}$$

Setting variable $x_{(c,r)}$ to 1 corresponds to placing a ball of radius r at center c, and therefore incurs a cost of r in the objective (II.1.1). Constraints (II.1.2) ensure that, for each point, the sum of x-values of balls that contain it, is at least 1. In other words, they ensure that each point is covered. Constraint (II.1.3) ensures that the total number of centers picked is at most k. Behsaz [20] showed that, even for metrics induced by unweighted graphs, the integrality ratio is unbounded. However, the LP is still used in designing primal-dual algorithms. The earliest paper making use of this is by Charikar and Panigrahy [34]. It is based on the famous primal-dual scheme for k-Median and Facility Location by Jain and Vazirani [69]. As in that paper, Charikar and Panigrahy first consider a *facility location version* of k-Min-Sum-Radii. [1] In [34], the authors move from facility location to k-Median by first giving an approximation algorithm for the former and then showing how to choose the facility opening cost so that the returned solution has at most k centers. Charikar and Panigrahy adapt this for k-Min-Sum-Radii and consequently give a 3.504-approximation algorithm. Ahmadian and Swamy [7] use the approach to obtain a 3.83-approximation for k-Min-Sum-Radii with lower bounds and a 12.265-approximation for lower bounds and outliers. In 2022, Friggstad and Jamshidian [55] tightened the analysis in [34], improving the approximation ratio to 3.389. Finally, in 2024 Buchem et al. [32] gave a $(3+\varepsilon)$-approximation for the unconstrained version and a $(3.5+\varepsilon)$-approximation for both lower bounds and outliers. They achieve this by introducing some fundamentally new ideas to the well-established primal-dual scheme.

In 2010, Gibson et al. [56] presented an optimal algorithm for k-Min-Sum-Radii in general metrics that runs in time $n^{\mathcal{O}(\log n \log \Delta)}$, where Δ is the spread[2] (or aspect ratio) of the instance. They also show NP-hardness, even for metrics induced by planar graphs. In [57], the same authors also give an optimal algorithm in the Euclidean setting with $d = 2$. They bound the runtime by $\mathcal{O}(n^{881})$ and only outline how the algorithm could be adapted for higher dimensions. This would however add an exponential dependency on d in the already high runtime.

Lev-Tov and Peleg [79] and Alt et al. [10] study slightly different versions of the problem, which is similar to the k-supplier problem. Here, the possible centers are

[1]In this version, there is no number k restricting the number of centers, but instead "opening" a center (i.e. assigning points to it) incurs some fixed cost.

[2]This is the ratio of largest to smallest pairwise point distance in the given instance.

not the same as the input points but a distinct set, and the task is to choose the k centers from that set. Lev-Tov and Peleg give an exact polynomial time algorithm for $d = 1$ and a PTAS for $d = 2$, while also showing that, given arbitrary centers, the method of assigning each point to its nearest center only yields a 4-approximation in one dimension. Alt et al. also consider the case where the centers are a discrete predefined set and give a 2-approximation in one dimension, and they additionally show NP-hardness if the cost function is not the sum of the radii but rather the sum of *powers* of radii, i.e. $\sum_{i=1}^{k} r_i^\alpha$ for some $\alpha > 1$. Finally, there has been a lot of recent work on k-Min-Sum-Radii with capacities and various fairness constraints. Due to the fact that these types of constraints tend to make clustering problems a lot harder to approximate, they are tackled by FPT approximations. In 2020, Inamdar and Varadarajan [68] gave a 28-approximation with a runtime of $2^{\mathcal{O}(k^2)} \cdot n^{\mathcal{O}(1)}$ for uniform capacities. This was improved by Bandyapadhyay et al. [19] in several ways. Firstly, they present a $(4+\varepsilon)$-approximation with runtime $2^{\mathcal{O}(k \log(k/\varepsilon))} \cdot n^3$. Secondly, they present a $(2+\varepsilon)$- and a $(1+\varepsilon)$-approximation for the Euclidean setting, where the runtime of the former only has a polynomial dependency on the dimension and the latter an exponential dependency. Finally, they also give a $(1+\varepsilon)$-approximation in Euclidean space with polynomial runtime dependency on the dimension, if capacities are allowed to be violated by a factor of at most $(1+\varepsilon)$. They also present a $(15+\varepsilon)$-approximation with runtime $2^{\mathcal{O}(k^2 \log k)} \cdot n^3$ for *non-uniform* capacities in general metrics. In 2024, Jaiswal et al. [70] presented a randomized $(4+\sqrt{13}+\varepsilon)$-approximation to the non-uniform case, with an expected runtime of $2^{\mathcal{O}(k^3 + k\log(k/\varepsilon))} \cdot \mathrm{poly}(n)$. Finally, Chen et al. [35] also considered *center fairness* (cf. Definition 21). As in group fairness, the input points have colors, but we do not care about the proportion of colors within the clusters, but rather their proportion among the *centers*. This is a special case of a *matroid constraint*, which was already tackled by Inamdar and Varadarajan [68], who give a $(9+\varepsilon)$-approximation with FPT runtime. Chen et al. improve this to $(3+\varepsilon)$, for the fair center constraint as well as general matroid constraints. Additionally, they also give an FPT $(2+\varepsilon)$-approximation for unconstrained k-Min-Sum-Radii.

II.1.2.2. The Assignment Problem

As mentioned before, a property that separates k-Min-Sum-Radii from other objectives is that the *assignment problem* is already NP-hard. The assignment problem for a given k-clustering problem differs in that a set of k centers is already given as an input and the task is to find an optimal assignment for these centers.

Min-Sum-Radii Assignment Problem

Input: A set P of n points, a set $\mathcal{C}$ of k centers, and a metric distance function $d : (P \cup \mathcal{C}) \times (P \cup \mathcal{C}) \to \mathbb{R}_{\geq 0}$.
Output: An assignment $\sigma : P \to \mathcal{C}$, such that the induced radii $r_1, \ldots, r_k$ minimize $\sum_{i=1}^{k} r_i$

We abbreviate this problem as MSR-A. To show that MSR-A is already hard, we can use an idea similar to the proof of Gibson et al. [56] who show that a similar problem, the k-cover problem, is NP-hard.

Theorem 32. *The Min-Sum-Radii Assignment Problem is NP hard.*

Proof. Let $\Phi = (X, C)$ be an instance of 3-SAT where $X = \{x_1, ..., x_n\}$ are the variables and $\mathcal{C} = \{C_1, ..., C_m\}$ are the clauses, consisting of exactly three literals each. Consider the *incidence graph* $G' = (V', E')$ that has a vertex ℓ for each literal, a vertex c_j for each clause and has an edge between ℓ and c_j iff literal ℓ is contained in clause C_j. Let $G = (V, E)$ arise from G' by adding a vertex y_i and edges $\{x_i, y_i\}$ and $\{\overline{x_i}, y_i\}$ for all $i \in \{1, \ldots, n\}$. Furthermore, define a weight function $w : E \to \mathbb{R}_{\geq 0}$ by setting $w(e) = 2^{i-1}$ for all edges incident to x_i or $\overline{x_i}$. See Figure II.1.3 for an illustration. Let L be the set of vertices corresponding to literals. Our instance $\mathcal{I}_\Phi$ for the MSR-A problem is then given by setting $P = V$ and $\mathcal{C} = L$, while the distance metric d is obtained by taking the metric closure of (G, w). This transformation can be computed in polynomial time. We show that the 3-SAT instance Φ is satisfiable if and only if $\mathcal{I}_\Phi$ has a solution of cost at most $2^n - 1$.

Assume we have some truth assignment satisfying Φ. For all $i \in \{1, \ldots n\}$, either x_i or $\overline{x_i}$ must be true. We use the center corresponding to the true literal with radius 2^{i-1} and set the radius of the center corresponding to the false literal to 0. Since we have exactly one true literal for each i, all vertices y_i are covered by some center. Since the assignment is satisfying, we also know that all clauses must contain at least one true literal, so each clause vertex is also covered by some center. The cost of this solution is $\sum_{i=1}^{n} 2^{i-1} = 2^n - 1$.

Now assume that we have a solution to the MSR-A instance with cost at most $2^n - 1$. Note that we may assume that, for every center $c \in L$, its radius is either 0 or equal to some distance to a y_i or clause vertex. Let $L' \subseteq L$ denote the set of facilities with positive radius. We obtain a truth assignment for Φ by setting all literals corresponding to vertices in L' to true and all other literals to false. Now for each variable x_i, exactly one of the literals x_i and $\overline{x_i}$ must be true. For the sake of a contradiction, let us first assume that there exists some i for which neither x_i nor $\overline{x_i}$ is true. Consider the largest i for which this happens. This means that both ℓ_i and $\overline{\ell_i}$ have radius 0 and therefore y_i has to be covered by some other center with index i'. By construction of the graph, the radius of that center then has to be at least $2^{i'} + 2 \cdot 2^{i-1}$. But this would lead to an overall cost of at least $2 \cdot 2^{i-1} + \sum_{j=i}^{n-1} 2^j = 2^n$ which contradicts the upper bound on the cost of the solution. On the other hand, assume that there exists some i for which both x_i and $\overline{x_i}$ are true. Again take the largest such i. Then the cost of the solution is again lower-bounded by $2 \cdot 2^{i-1} + \sum_{j=i}^{n-1} 2^j = 2^n$.

Thus, it follows that exactly one literal is true for each variable, which makes the assignment feasible. Furthermore, the solution for the MSR-A instance is feasible and therefore covers all clause vertices. This translates to every clause containing at least one true literal, making the assignment satisfying as well. □

The above hardness result suggests that assigning each point to its closest center – like one would do for the other objectives – does not work for k-`MinSumRadii`. As it

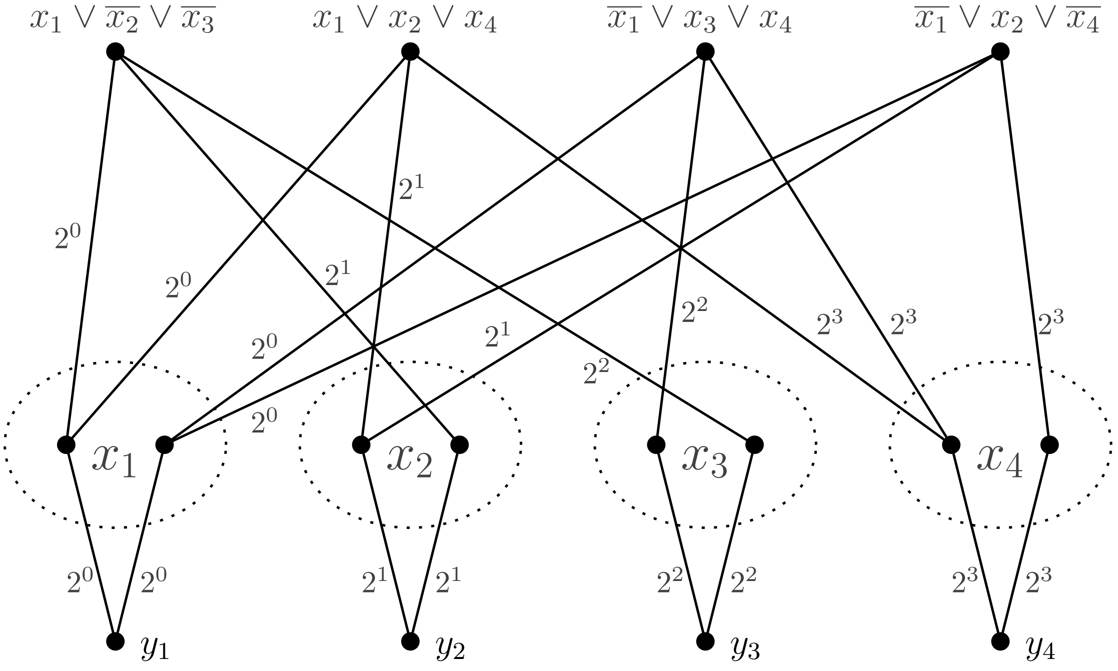


Figure II.1.3.: Example transformation of a 3-SAT instance with 4 variables and 4 clauses to a MSR-A instance.

turns out, this strategy can indeed lead to very bad solutions. Lev-Tov and Peleg [79] show that, already in the case where all the points and centers lie one a line, the closest assignment only gives a 4-approximation. For a more general metric setting we can show an even worse bound, as illustrated in II.1.4.

Lemma 33. *Assigning each point to its closest center yields a k-approximation to the* Min-Sum-Radii Assignment Problem *at best.*

Proof. Consider the instance $I = (P, \mathcal{C}, d)$ depicted in Figure II.1.4. In this instance, we have $2(k-1)$ points, arranged in two concentric rings around a center c_1. The $k-1$ points on the inner ring are at distance r from c_1, the $k-1$ points on the outer ring at distance $r+\epsilon$. Finally, we have $k-1$ centers on a third concentric ring at distance $2r$ from c_1. For every point in the inner ring, the closest center is c_1 with a distance of r. For the points in the outer ring, the closest center is one of the $k-1$ outer centers, with a distance of $r-\varepsilon$. So using the closest center assignment, we get a solution with one cluster of radius r and $k-1$ clusters of radius $r-\varepsilon$, leading to an overall cost of $kr-(k-1)\varepsilon$. The optimum assignment however assigns all points to c_1 leading to a cluster with radius $r+\varepsilon$, while the outer centers get no points and therefore all have cost 0. Since ε can be chosen arbitrarily small, we get an approximation ratio of at least

$$\lim_{\varepsilon \to 0} \frac{kr-(k-1)\,\varepsilon}{r+\varepsilon} = k,$$

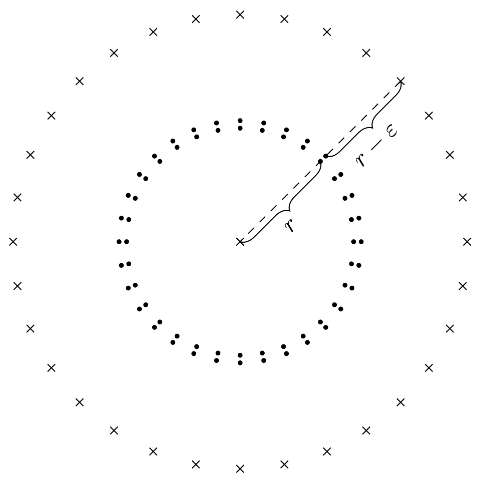


Figure II.1.4.: An example where the solution obtained by closest center assignment is up to k times as expensive as an optimal assignment. Each cross corresponds to a center. The optimum would assign all points to the center in the middle, leaving the outer centers with radius 0. Assigning each point to its closest center gives one cluster with radius r in the middle, and $k-1$ clusters with radius $r-\varepsilon$ each.

proving the claim. □

Note that this implies that a very common approach of algorithm design for clustering fails. Many algorithms proceed by finding some center set and then optimally assigning the points to these centers. In the end, they keep the center set with the lowest cost. But for k-Min-Sum-Radii, it is not even possible to (efficiently) judge the quality of a given center set, as we cannot efficiently find the optimal assignment for that specific set.

II.1.3. Oracles and Guessing Techniques in FPT Algorithms

Sometimes, one encounters the term *oracle* in algorithmics. Often, this refers to some assumed black box which can solve some subproblem. For example, in Linear Programming it can be possible to solve linear programs of exponential size in polynomial time by using a *separation oracle*. In this context, the oracle is some polynomial time algorithm that can, given some solution to the linear program, decide whether all constraints are satisfied and, if not, return a violated constraint.

But in the context of FPT algorithms, a different notion of oracles is of interest. As an example, we look at a type of problem that comes up in the $(1+\varepsilon)$-approximation algorithm for k-Center by Bădoiu et al. [31] as well as our algorithms in Chapter II.2 and Chapter II.3. Fix an optimal solution with clusters $C_1^*,\dots,C_k^*$ and an ordering $p_1,\dots,p_n$ of the input points P. We now go through the points and “guess” for each one to which optimal cluster it belongs, by consulting some oracle. We want to keep doing this until some pre-defined stopping criterion is met, say after ℓ iterations. Note that we can encode any such sequence of guesses as a tuple $a = (i_1, i_2, \dots, i_\ell) \in \{1, \dots, k\}^\ell$, where the j-th entry is the index of the cluster to which we assign p_j. Of course, information about cluster membership is not available to us. We ignore that fact for the time being, and instead focus on a sequence of guesses where we happen to make the correct decision in each iteration. Then we try to prove that, in this case, we reach our desired stopping criterion sufficiently quickly. If we can show, for example, that by guessing correctly in each iteration we reach the criterion in $f(k, \varepsilon)$[3] iterations, we can conclude that we only ever need to consider tuples of length $f(k, \varepsilon)$: if we do not reach the stopping criterion after that many steps, we *know* that we have not "guessed correctly". This opens up the possibility of trying out all sequences of length $f(k, \varepsilon)$. One of them will be the "correct" one for which we have proven that the stopping criterion will be reached. How many such sequences are there? For p_1, we have k choices. For each of these k choices, we then have another k choices where to assign p_2, making for k^2 choices for the first two points combined. Doing this for $f(k, \varepsilon)$ points leads to $k^{f(k,\varepsilon)}$ tuples. See Figure II.1.5 for an illustration. While this of course is exponential in k, it is completely independent of the input size n. So if the additional computational work needed for each tuple is polynomial in terms of n, we end up with an FPT-algorithm. In the case of [31], it is shown that after at most $64k/\varepsilon^2$ guesses, one can obtain a $(1 + \varepsilon)$-approximation for k-Center, if all guesses were correct (cf. Theorem 44). This means that k^{64k/ε^2} sequences need to be checked.

So guessing in FPT algorithms can be viewed as "clever enumeration". Instead of enumerating *all* possible solutions (whose number is usually exponential in n for combinatorial optimization problems), we identify a set of candidate solutions whose number only depends exponentially on the parameter k (and possibly a precision parameter ε). To see another example of guessing, we take a look at the two FPT approximation algorithms for k-Min-Sum-Radii presented in this chapter. For both, we assume that we can guess a near-optimal radius profile (cf. Definition 24). Of course, we do not have any prior knowledge on the optimum profile. Therefore, we identify a set of candidates radius profiles such that *a*) one of these candidates is sure to be near-optimal, and *b*) the overall number of candidates is not too big. Then we iterate over all candidates, which ensures that we will definitively consider the near-optimal candidate (even if we do not know which one it is). In the analysis of the algorithm, it then suffices to only consider the iteration where we consider this candidate, i.e. where we "guessed correctly". If we can show that, in this case, the algorithm always computes a feasible solution, and if we can bound the cost of that solution, we can conclude that the algorithm as a whole has (at least) that approximation guarantee. Since we aim at obtaining an FPT algorithm, we are again content with a candidate set whose size is $f(k, \varepsilon)$ for some computable function f. We describe this procedure in more detail in the following.

[3]where f is an arbitrary computable function

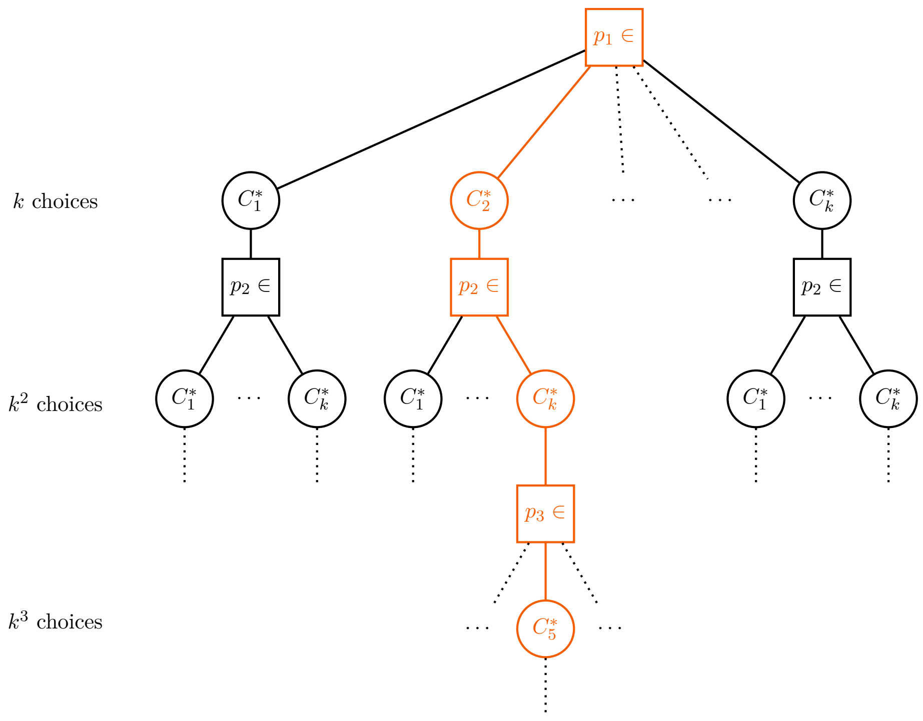


Figure II.1.5.: A visualization of all possible guessing sequences. In the first iteration, the algorithm guesses that p_1 belongs to C_2^*, in the second iteration it guesses that p_2 belongs to C_k^* and so on. The orange path therefore corresponds to a tuple $(2, k, 5, \ldots)$. For every point, we have k possible decisions, so at depth i, we have k^i nodes.

II.1.3.1. Guessing the Radius Profile

In the algorithms at hand, we actually proceed in two steps to obtain a near-optimal radius profile. At first, we generate a suitable candidate set for a near-optimal largest radius $\tilde{r}_1$, and then show how, for each of these candidates, we can generate a suitable candidate set for the remaining radii.

Obtaining a Candidate Set for $\tilde{r}_1$ We present two ways of obtaining a candidate set for the largest radius, each of which has upsides and downsides. The first variant has the advantage that the size of the candidate set only depends on k and ε, and it works in the continuous Euclidean setting as well as the discrete setting. Initially, we identify an interval in which the largest radius r_1^* of the optimal solution must lie. To this end, we utilize Lemma 31, shown in Section II.1.2.

Lemma 31. *Let I be an instance for* Center Based k-Clustering*. Let S_{kc} be an α-approximate* k-Center *solution with value $r_\alpha := \texttt{kCenter}(S_{kc})$, and let S^*_{msr} be an* *optimum* k-Min-Sum-Radii *solution with largest radius r_1^*. Then it holds that*

$$r_1^* \in \left[\frac{r_\alpha}{\alpha}, k \cdot r_\alpha\right].$$

In other words, if we have access to a constant factor approximation for the optimum k-Center solution, we can compute an interval in which the largest radius in the optimum k-Min-Sum-Radii solution must lie. Now we can discretize this interval to obtain a candidate set for r_1^*. Recall that we want to ensure that there always exists a candidate $\tilde{r}_1$ with $r_1^* \leq \tilde{r}_1 \leq (1+\varepsilon)r_1^*$ while simultaneously ensuring that the candidate set is not too big. We can use the following lemma.

Lemma 34. *Let $\varepsilon > 0$, $I = [a,b] \subset \mathbb{R}_{\geq 0}$ and $m = \lceil \log_{(1+\varepsilon)}(b/a) \rceil$. Let $r^* \in [a,b]$ be a number we want to approximate. Then the set*

$$R = \{(1+\varepsilon)^i a \mid i \in \{0,\dots,m\}\}$$

contains a number r with $r^* \leq r \leq (1+\varepsilon)r^*$.

Proof. For $j \in \{1,\dots m\}$, let $I_j = \left[(1+\varepsilon)^{j-1}a, (1+\varepsilon)^j a\right]$. Note that R consists of the endpoints of these intervals and that $\bigcup_{j=1}^{m} I_j \supseteq [a,b]$. Thus, for every $r^* \in [a,b]$, there exists some $i \in \{1,\dots,m\}$ such that $r^* \in I_i$. Then for $r = (1+\varepsilon)^i a$ (i.e. the right endpoint of I_i) we clearly have $r^* \leq r \leq (1+\varepsilon)r^*$, which proves the claim. □

We can think about this as covering the interval I with smaller intervals that get exponentially larger. This exponential growth in length ensures that for each of the smaller intervals the right endpoint is a $(1+\varepsilon)$-approximation for every number within the interval, while the number of such intervals is only logarithmic in the length of I (albeit with a base of $(1+\varepsilon)$). We can immediately apply this to the interval from Lemma 31.

Lemma 35. *Let $(r_1^*,\dots,r_k^*)$ be the radius profile of an optimal solution for k-Min-Sum-Radii. Let r_α denote the value of an α-approximate solution for* k-Center *on the same instance. Then the set*

$$R = \left\{(1+\varepsilon)^i \frac{r_\alpha}{\alpha} \mid i \in \left\{0,\dots,\lceil \log_{(1+\varepsilon)} \alpha k \rceil\right\}\right\}$$

contains a number $\tilde{r}_1$ with $r_1^* \leq \tilde{r}_1 \leq (1+\varepsilon)r_1^*$.

Proof. By Lemma 31, the interval $I = \left[\frac{r_\alpha}{\alpha}, k \cdot r_\alpha\right]$ contains r_1^*. We now cover this interval with smaller intervals.
For $j \in \left\{1,\dots,\lceil \log_{(1+\varepsilon)} \alpha k \rceil\right\}$, let $I_j = \left[(1+\varepsilon)^{j-1}\frac{r_\alpha}{\alpha}, (1+\varepsilon)^j \frac{r_\alpha}{\alpha}\right]$. So the first interval is given by $I_1 = \left[\frac{r_\alpha}{\alpha}, (1+\varepsilon)\frac{r_\alpha}{\alpha}\right]$, the second by $I_2 = \left[(1+\varepsilon)\frac{r_\alpha}{\alpha}, (1+\varepsilon)^2\frac{r_\alpha}{\alpha}\right]$ and so on.
Consider the radius r_1^* that we want to approximate. Notice that $\bigcup_j I_j \supseteq I$, so there has to be some j such that $r_1^* \in I_j$. The set R consists of the endpoints of these intervals,

so we take the corresponding endpoint $\tilde{r}_1 = (1+\varepsilon)^j \frac{r_\alpha}{\alpha}$. By the way the intervals are constructed, we know that $r_1^* \leq \tilde{r}_1 \leq (1+\varepsilon)r_1^*$, proving the claim. □

What this means is that we have found a way to compute a candidate set that is sure to contain a good approximation for the largest radius, and whose size only depends on k and ε. So even if we iterate over all these possibilities, it only adds factors depending on k and ε to the runtime.

We now introduce the second method of obtaining the candidates, based on Observation 30, by which we can obtain a candidate set of size $\mathcal{O}(n^2)$ for the largest radius by simply computing all pairwise point distances. Note that guessing *all* k radii this way would not lead to an FPT algorithm, as the size of D does depend on n and not on k. Therefore, we would have n^2 choices for the first radius, n^2 for second and so on, leaving us with n^{2k} candidates for the radius profile. In the FPT regime, the (possibly superpolynomial) terms in the runtime involving k need to be separated from the terms involving n. But what we can do instead is try out all candidates for r_1^*, which only introduces a single factor of n^2, and then proceed with the remaining radii in the same way as we would if we used the first method. So ultimately, it is a trade-off. If $\lceil \log_{(1+\varepsilon)} \alpha k \rceil < n^2$, the first method is preferable, otherwise the second method.

It should be noted that the second variant, while simpler than the first, only works in the discrete setting, i.e. if the centers choices are confined to input points. Only then is Observation 30 true. In the Euclidean setting for example, radii do not have to correspond to pairwise distances. Instead, if our points are from $\mathbb{R}^d$, each optimal radius will correspond to a minimum enclosing ball of a set of $d+1$ points (see e.g. [71]). So in this case, the number of candidates for *one* radius will be $\mathcal{O}(d)$, which, depending on d, might be undesirable.

Let it be noted that there is a third way of guessing the largest radius, which utilizes Observation 30 as well as Lemma 34. Again, it only works in the discrete setting and has the downside that it introduces another parameter that impacts the runtime. To use Lemma 34, we need an upper bound for the largest radius as well as a lower bound for the smallest radius. Let $d_{\min} = \min\{d(p,q) \mid p,q \in P, p \neq q\}$ and $d_{\max} = \max\{d(p,q) \mid p,q \in P, p \neq q\}$. By Observation 30, all radii must lie in the interval $[d_{\min}, d_{\max}]$. By Lemma 34, we can then generate a candidate set that will contain a $(1+\varepsilon)$-approximate guess for r_1^* of size $\Delta_P := d_{\max}/d_{\min}$. So, if this Δ_P guaranteed to be small, this approach could also be feasible. But in general, the spread can get arbitrarily large, even for small datasets.

Obtaining a Candidate Set for the Remaining Radii Once we have fixed a value for $\tilde{r}_1$, we would like to proceed similarly and identify an interval in which the remaining radii must lie. Recall that, even in the discrete setting, we cannot use the approach provided by Observation 30, since this would not lead to an FPT algorithm. Therefore, we again aim at identifying an interval in which the radii must lie and then "covering" that interval with smaller intervals. As an upper bound, we can simply use our guess $\tilde{r}_1$, but we are lacking a lower bound. Since we do not want to solve the problem exactly but with an approximation factor of $(1+\varepsilon)$ in the Euclidean and $(6+\varepsilon)$ in the metric case, a simple trick can serve as a workaround: we only consider ε-balanced solutions, i.e. solutions where the smallest radii are not too small (cf. Definition 25). Instead of approximating the optimum solution, we are now approximating the best among all ε-balanced solutions. By Observation 26, any optimum solution can be turned into an ε-balanced one with at most $(1 + \varepsilon)$ times the cost. Thus, the cost of the optimum ε-balanced solution can also be at most $(1 + \varepsilon)$ times the optimum. Note that this implies that a $(1+\varepsilon)$-approximate ε-balanced solution is only a $(1+\varepsilon)^2$-approximation of the overall optimum solution. We account for this later on by calling the algorithm with an appropriate $\varepsilon' < \varepsilon$.

Note that we are now trying to approximate a *covering* of the point set P. Since we possibly "inflate" smaller radii, we might have radii that can not be induced by a center set $\mathcal{C}$ and an assignment σ.

From now on we will assume that we are trying to approximate an ε-balanced covering $\mathscr{B}^* = \{\mathrm{B}(c_1^*, r_1^*), \ldots, \mathrm{B}(c_k^*, r_k^*)\}$ where $r_1^* \geq \ldots \geq r_k^*$. By Observation 26, the best ε-balanced covering will at most cost $(1+\varepsilon)$ times as much as the actual optimal solution. So if we are able to show that we can approximate the best ε-balanced covering within a factor of $(1+\varepsilon)$, we can conclude that we get a $(1+\varepsilon)^2$-approximation for the optimal solution. There are, of course, some caveats. Lemma 31 only applies to *optimal* k-MSR solutions, so we have to adjust it slightly to make it work for approximate solutions.

Lemma 36. *Let I be an instance for* Center-Based k-Clustering. *Let S_{kc} be an α-approximate* k-Center *solution with value r_α, and let S_{msr} be a β-approximate* k-Min-Sum-Radii *solution with largest radius r_1^*. Then it holds that*

$$r_1^* \in \left[\frac{r_\alpha}{\alpha}, \beta \cdot k \cdot r_\alpha\right]$$

Proof. The arguments closely follow those in the proof of Lemma 31. For the lower interval limit, the argument does not need to be changed. So we can still conclude that $r_1^* \geq \frac{r_\alpha}{\alpha}$.
For the upper limit, let S^*_{msr} be an optimal solution for k-Min-Sum-Radii. We again notice that $\texttt{kMSR}(S_{\mathrm{kc}}) \leq k \cdot r_\alpha$, which implies $\texttt{kMSR}(S^*_{\mathrm{msr}}) \leq k \cdot r_\alpha$. With this, we can conclude

$$r_1^* \leq \texttt{kMSR}(S_{\mathrm{msr}}) \leq \beta \cdot \texttt{kMSR}(S_{\mathrm{msr}^*}) \leq \beta \cdot k \cdot r_\alpha$$

□

Now we can also adjust Lemma 35 to obtain an interval in which the largest radius of a β-approximate solution must lie.

Corollary 37. *Let $\mathscr{B}^* = \{\mathrm{B}(c_1^*, r_1^*), \ldots, \mathrm{B}(c_k^*, r_k^*)\}$ be a covering with largest radius r_1^*, whose corresponding clustering $\{B_1 \cap P, \ldots, B_k \cap P\}$ is a β-approximate solution for k-MSR. Let r_α denote the value of an α-approximate k-center solution. Then the set*

$$R = \left\{(1+\varepsilon)^i \frac{r_\alpha}{\alpha} \;\mid\; i \in \left\{0, \ldots, \lceil \log_{(1+\varepsilon)} \alpha\beta k \rceil \right\}\right\}$$

contains a number $\tilde{r}_1$ with $r_1^* \leq \tilde{r}_1 \leq (1+\varepsilon)r_1^*$.

In our particular case, the factor β is $(1+\varepsilon)$, as we want to approximate the best ε-balanced solution; from Observation 26, we know that this is a $(1+\varepsilon)$-approximation for the optimal solution. So we can compute a candidate set of size $O\left(\log_{(1+\varepsilon)}(1+\varepsilon)k\right)$[4] that will contain a radius $\tilde{r}_1 \leq (1+\varepsilon)r_1^* \leq (1+\varepsilon)^2 r_{\text{opt}}$.

Turning back to the remaining radii, recall that we want to identify an interval $I = [a,b]$, such that $r_i^* \in [a,b]$ for all $i \in \{2,\dots,k\}$. The ε-balancedness of $\mathscr{B}^*$ and the fact that we have a guess $\tilde{r}_1$ for the largest radius allow us to set $I = \left[\frac{\varepsilon}{k}\tilde{r}_1, \tilde{r}_1\right]$. Now we can discretize this new interval in the same manner as before.

Lemma 38. *Let $\varepsilon > 0$ and let $\mathscr{B}^*$ be an ε-balanced covering with radii $r_1^* \geq \ldots \geq r_k^*$. Let $\tilde{r}_1$ be a guess for r_1^* that fulfills $r_1^* \leq \tilde{r}_1 \leq (1+\varepsilon)r_i^*$. Then the set*

$$R = \left\{(1+\varepsilon)^j \frac{\varepsilon\,\tilde{r}_1}{k} \mid i \in \{0,\dots,\lceil \log_{1+\varepsilon} \frac{k}{\varepsilon}\}\rceil\right\}$$

contains for each r_i^* a number $\tilde{r}_i$ with $r_i^* \leq \tilde{r}_i \leq (1+\varepsilon)r_i^*$.

Proof. Since the solution is ε-balanced, we have $r_i^* \in \left[\frac{\varepsilon\, r_1^*}{k}, r_1^*\right]$ for all $i \in \{2,\dots,k\}$. The claim then immediately follows from Lemma 34. □

Let us briefly sum up what we have learned so far and look at the possibilities this opens up for us. By applying any polynomial time approximation algorithm for k-Center (e.g. the well-known 2-approximation by Gonzalez [58]), we get an interval in which the value of the largest radius r_1^* of our optimal k-MSR solution must lie. We can discretize it to obtain a set of $\mathcal{O}\left(\log_{(1+\varepsilon)} k\right)$ candidate radii, one of which be a $(1+\varepsilon)$-approximation for r_1^*. For each of these candidates, we can then construct a set of $\mathcal{O}\left(\log_{(1+\varepsilon)} \frac{k}{\varepsilon}\right)$ numbers, which contains a $(1+\varepsilon)$-approximation for the remaining radii. Note that this is only guaranteed if our initial guess $\tilde{r}_1$ for r_1^* is "correct", in the sense that $\tilde{r}_1 \in [r_1^*, (1+\varepsilon)r_1^*]$. This, however, is not a problem, as we will enumerate over all possible candidates and will therefore at some point consider the correct one. How many possible configurations of radii do we need to check? For r_1^* we already know the number of possibilities. For each of the remaining $k-1$ radii, we have $\mathcal{O}\left(\log_{(1+\varepsilon)} \frac{k}{\varepsilon}\right)$ possible values. So the total number of possibilities is

$$\mathcal{O}\left(\log_{(1+\varepsilon)} k \cdot \left(\log_{(1+\varepsilon)} \frac{k}{\varepsilon}\right)^{k-1}\right) = \mathcal{O}\left(\left(\log_{(1+\varepsilon)} \frac{k}{\varepsilon}\right)^{k}\right),$$

which is fine within the FPT regime.

Corollary 39. *Let $(r_1^*,\dots,r_k^*)$ be the radius profile of an optimal solution to k-Min-Sum-Radii. Then we can compute a set of size $\mathcal{O}\left(\left(\log_{(1+\varepsilon)} \frac{k}{\varepsilon}\right)^k\right)$ that contains a near-optimal radius profile in FPT time.*

[4]if α is constant

II.2. A $(1+\varepsilon)$-Approximation for Euclidean k-Min-Sum-Radii

We now focus on EUCLIDEAN k-MIN-SUM-RADII, introduced in Section II.1.1. This means $P \subseteq \mathbb{R}^d$, i.e. our points live in d-dimensional Euclidean space, and we consider continuous clustering as well, where centers can be chosen freely from $\mathbb{R}^d$. This setting has its advantages but also, as already mentioned in the previous section, some disadvantages.

Our algorithm yields a $(1+\varepsilon)$-approximation for k-MSR. It works similar to the $(1+\varepsilon)$-approximation algorithm for k-CENTER by Bădoiu et al. [31], so we will first take a closer look at this algorithm as a warm-up. The actual k-CENTER algorithm is only outlined in said paper, so we are going to restate it in a more formal way here. Seeing how and why it works for k-CENTER also gives more insight into the changes that we have to make for our k-MIN-SUM-RADII algorithm.

II.2.1. A $(1+\varepsilon)$-Approximation for Euclidean k-Center

The paper by Bădoiu et al. actually focuses on finding what they call *coresets* for the input point set. A coreset for a point set P is a subset $S \subseteq P$ with the property that the radius of $\mathrm{MB}(S)$ is at least $1/(1+\varepsilon)$ the radius of $\mathrm{MB}(P)$. In other words, “inflating” the minimum enclosing ball of P by a factor of $(1+\varepsilon)$ yields a ball that covers all of P. In this paper, the authors show that, for any point set P, there exists a coreset of size $\mathcal{O}(1/\varepsilon^2)$, i.e. constant size for fixed $\varepsilon > 0$. This can then be used to design a $(1+\varepsilon)$-approximation for EUCLIDEAN k-CENTER.

Euclidean k-Center Problem

Input: A set $P \subset \mathbb{R}^d$ of n points, a number $k \in \mathbb{N}, k \leq n$

Output: A center set $\mathcal{C} = \{c_1, \dots, c_k\} \subset \mathbb{R}^d$ and an assignment $\sigma : P \to \mathcal{C}$, such that the induced radii $r_1, \dots, r_k$ minimize $\max_{i \in [k]} r_i$

We start by describing the algorithm in a high-level manner before turning to the details. Let us fix an arbitrary optimal solution OPT with clusters $\mathcal{C}_1^*, \dots, \mathcal{C}_k^*$ and radii $r_1^*, \dots, r_k^*$.

The algorithm now consecutively builds small representative sets $S_1, \dots, S_k$ for the OPT-clusters. It does so by employing the “guessing” techniques described in Section II.1.3. Start with an arbitrary point p_0 and ask the oracle for its cluster in the optimal solution, say $\mathcal{C}_i^*$. Add p_0 to the corresponding set $\mathcal{S}_i$. Now choose the point p_1 *farthest* away

from p_0 and repeat the procedure. Continue choosing the point farthest away from all previously chosen points until we reach an iteration in which a point p_i is assigned to a set S_j which already contains a point q. Compute the minimum enclosing ball $\mathrm{B}(c_j, r_j) := \mathrm{MB}(\{p_i, q\})$ and let $\mathrm{MB}_\gamma(\{p_i, q\}) := \mathrm{B}(c_j, \gamma r_j)$ be the ball that results from enlarging its radius by a factor of $\gamma > 1$. We now consider all points within $\mathrm{MB}_\gamma(\{p_i, q\})$ as *covered* and do not consider them in future iterations.
We proceed with constructing the sets $S_1, \dots, S_j$ and their corresponding enclosing balls $\mathrm{MB}(S_1), \dots, \mathrm{MB}(S_k)$ until the union of the enlarged balls $\mathrm{MB}_\gamma(S_1), \dots, \mathrm{MB}_\gamma(S_k)$ covers all of P. These enlarged balls constitute our actual solution. If we get the cluster membership guesses right, it is clear that this yields a γ-approximation for the k-Center objective. In fact, we can even state a stronger observation.

Observation 40. *Let $\mathcal{B} = \{\mathrm{MB}_\gamma(S_1), \dots, \mathrm{MB}_\gamma(S_k)\}$ be the set of balls obtained by the described procedure when cluster membership guesses are correct, and let $r_1, \dots, r_k$ be their radii. Then $\mathcal{B}$ is a feasible solution for the k-Center problem, and we have $r_i \leq \gamma r_i^*$, $1 \leq i \leq k$.*

So not only do we have a γ-approximation for the *largest* radius in the optimal solution, but for *every* radius. This is a first hint at why this method also works for k-Min-Sum-Radii, albeit with some alterations. To make the notation compatible with the original paper, we set $\gamma = 1 + \varepsilon$ for some $\varepsilon > 0$ for the remainder of this subsection.

As described in Section II.1.3, we can encode any sequence of ℓ membership guesses as tuple $u = (u_1, \dots, u_\ell) \in \{1, \dots, k\}^\ell$. As a next step, we therefore want to argue that the stopping criterion (i.e. the γ-enlarged balls covering the whole instance) is reached sufficiently quickly. More precisely, we want to argue that for the tuple that encodes a sequence of correct guesses we have $\ell \in \mathcal{O}(f(k, \varepsilon))$ for some (computable) function f.

The analysis in [31] proceeds in two steps. First, they show that the initial MEB (after adding a second point to a cluster consisting of a single point) has a sufficiently large radius. Afterwards, they show that each time a new point is added to a non-singleton cluster, the corresponding MEB grows by a sufficiently large factor and thus it cannot take too long until the ball is roughly the same size as the corresponding optimal solution ball. While the analysis in the paper is only done explicitly for $k = 1$, we will go through it for general k in a more detailed manner.

One detail that is not explicitly discussed by Badŏiu et al. is the complexity of computing minimum enclosing balls. The best known algorithm by Welzl [93] runs in $\mathcal{O}(n)$ for *constant dimension*, but has an exponential dependency on the dimension. In fact, the runtimes of all known exact algorithms depend exponentially on the dimension of the points. To get rid of this dependency, we instead compute approximate minimum enclosing balls (cf. Line 7 in Algorithm 2). This can for example be done via the algorithm presented by Yıldırım [96]. It also takes an accuracy parameter $\varepsilon > 0$ as input, and returns for any point set $P \subset \mathbb{R}^d$ a $(1+\varepsilon)$-approximate MEB in time $\mathcal{O}(\frac{|P|d}{\varepsilon})$[1]. This will of course impact the final approximation guarantee and runtime. We argue in the

[1]Note that in our case $|P| \in \mathcal{O}(1/\varepsilon^2)$ for all sets P for which we use the algorithm, so this introduces more runtime dependency on ε.

proof of Theorem 44 how we can mitigate this by adjusting the ε that we pass to the algorithm.

In the following, consider an arbitrary S_i with corresponding Opt-cluster C_i^* and radius r_i^*. Let S_i^t denote the state of set S_i at the end of iteration t of the algorithm, and let $\mathrm{B}(c_i^t, r_i^t)$ be the $(1+\varepsilon)$-approximation of $\mathrm{MB}(S_i^t)$ that the algorithm has computed.

Lemma 41. *Let $t > 1$ be the round of the algorithm in which a set S_i stops being a singleton cluster, i.e. the smallest t for which we have $|S_i^t| \geq 2$. Let r_i^t be the radius of $\mathrm{MB}(S_i^t)$. We then have $r_i^t \geq \frac{1}{2} r_i^*$.*

Proof. Let $k' \leq k$ and let $c_1^{t-1}, \dots c_{k'}^{t-1}$ be the current centers. Let p be a point that gets added to singleton set $S_i^{t-1} = \{c_i^{t-1}\}$ in iteration t.
Let c_ℓ^{t-1} be the center *closest* to p. Note that not necessarily $\ell = i$, as p might acually be closer to a center that belongs to some other set. But since p was chosen because it maximizes the distances to *all* current centers, we must have $\|p - c_\ell^{t-1}\| \geq \max_{j\in[k]} r_j^*$. To see this, assume for the sake of contradiction that it is not the case, i.e. $\|p - c_\ell^{t-1}\| < \max_{j\in[k]} r_j^*$. Consider the solution obtained by taking all current centers $c_1^{t-1}, \dots, c_{k'}^{t-1}$ and assigning each point to its closest center among these. Since p was the point maximizing the distance to these centers, all points must have a distance less than $\max_{j\in[k]} r_j^*$ as well. But $\max_{j\in[k]} r_j^*$ is exactly the value of the optimal solution, which is a contradiction.
Since c_ℓ^{t-1} is the closest center to p, we get

$$\|p - c_i^{t-1}\| \geq \|p - c_\ell^{t-1}\| \geq \max_{j\in[k]} r_j^* \geq r_i^*.$$

After adding p to S_i^{t-1}, we update the center, which now moves to the barycenter, i.e. the middle point between p and c_i^{t-1}. For the new radius r_i^t we therefore get $r_i^t = \frac{1}{2}\|p - c_i^{t-1}\| \geq \frac{1}{2} r_i^*$. ☐

Note that in the case $|S_i^t| = 2$ we can compute $\mathrm{MB}(S_i^t)$ exactly, as we just have to compute the midpoint of the two points in the set. So we have shown that the initial radii of our balls are a constant fraction of the actual radii in the optimal solution. Next, we argue that each addition of a point increases the radius by a sufficient amount.

Lemma 42. *Let $\varepsilon \in (0,1)$ and let $t > 1$ be the iteration in which a new point p is added to a non-singleton cluster S_i. We have $r_i^t \geq \left(1 + \frac{\varepsilon^2}{16}\right) \cdot r_i^{t-1}$.*

Proof. Let $\mathrm{B}\left(c_i^{t-1}, r_i^{t-1}\right)$ be the $(1+\varepsilon)$-approximation for $\mathrm{MB}(S_i^{t-1})$ that the algorithm has computed. Since we do not consider any point inside the $(1+\varepsilon)$-enlargements of the balls that we computed in iteration $t-1$, we know that $\|c_i^{t-1} - p\| \geq (1+\varepsilon) r_i^{t-1}$. The addition of p to S_i means that we have to compute the approximate MEB $\mathrm{B}(c_i^t, r_i^t)$ for S_i^t. To get a lower bound on r_i^t, we consider the *actual* minimum enclosing ball $\mathrm{MB}(S_i^t)$, whose center we will call c_i' and whose radius we will call r_i'. We first argue that r_i' is sufficiently larger than r_i^{t-1}, from which it directly follows that r_i^t is sufficiently larger

as well. To show that, we make a case distinction based on the distance between c_i' and the old center c_i^{t-1}.
If $\|c_i' - c_i^{t-1}\| < \frac{\varepsilon}{2} r_i^{t-1}$, then we have

$$\begin{aligned} r_i' \geq \|p - c_i'\| &\geq \|p - c_i^{t-1}\| - \|c_i^{t-1} - c_i'\| \\ &\geq (1+\varepsilon) r_i^{t-1} - \frac{\varepsilon}{2} r_i^{t-1} \\ &= \left(1 + \frac{\varepsilon}{2}\right) r_i^{t-1}. \end{aligned}$$

If $\|c_i' - c_i^{t-1}\| \geq \frac{\varepsilon}{2} r_i^{t-1}$, draw a line L through c_i^{t-1} and c_i' and let H be the $(d-1)$-dimensional hyperplane orthogonal to L anchored at c_i^{t-1}. We denote the open halfspace induced by H that does not contain c_i' by H^-. Lemma 2.2 from [31] shows that there exists a point x in $H^- \cap S_i^{t-1}$ that is at a distance r_i^{t-1} from c_i^{t-1}. Since the triangle drawn by x, c_i^{t-1} and c_i' has an obtuse angle at c_i^{t-1}, we can apply the law of cosines to get

$$r_i' \geq \|x - c_i'\| \geq \|c_i' - p\| \geq \sqrt{\left(r_i^{t-1}\right)^2 + \frac{\varepsilon^2}{4}\left(r_i^{t-1}\right)^2} \geq \left(1 + \frac{\varepsilon^2}{16} r_i^{t-1}\right).$$

So we have argued that, in any case, $r_i' \geq (1+\varepsilon)^2/16 r_i^{t-1}$. Since $\mathrm{B}(c_i^t, r_i^t)$ covers S_i^t, we can conclude that $r_i^t \geq r_i'$, which concludes the proof.

□

These two lemmata can now be combined to conclude that the required number of additions to a single set S_i until $C_i^* \subset \mathrm{B}(c_i^t, r_i^t)$ is fairly small.

Corollary 43. *For all $i \in [k]$ it holds that after at most $\frac{16}{\varepsilon^2}$ additions to S_i, we have $C_i^* \subset \mathrm{B}(c_i^t, r_i^t)$.*

Proof. By Lemma 41, we start with an initial radius of at least $r_i^*/2$. By Lemma 42, we know that the next addition of a point increases the radius by at least $\frac{r_i^*}{2}\frac{\varepsilon^2}{16} = \frac{r_i^* \varepsilon^2}{32}$. We can also use that as lower bound for the increase in future iterations, as the radii only get bigger. We can thus conclude that after at most $16/\varepsilon^2$ iterations we will have reached a radius of r_i^*, at which point C_i^* will be covered by the computed ball $\mathrm{B}(c_i^t, r_i^t)$. □

So we obtained a bound for the number of iterations that *one* of the sets needs to become a good approximation for the corresponding Opt-cluster. Since we focus on the tuple u which encodes the correct guess for any point, we know that we will be done after at most $16k/\varepsilon^2$ iterations (since there are k sets). Thus, we can focus on tuples $u \in \{1, \dots, k\}^{\frac{16k}{\varepsilon^2}}$, as any sequence of choices that will not terminate after that many steps cannot be the correct sequence. There are k^{16k/ε^2} different tuples of that length. While being exponential in k and $1/\varepsilon$, it does not depend on the input length whatsoever.
Algorithm 2 describes the execution of one such sequence. The algorithm gets a string

Algorithm 2: Selection for k-Center

```
Input: A set $P \subseteq \mathbb{R}^d$, a number $k \in \mathbb{N}$, a string $u \in \{1, \dots, k\}^{\frac{16k}{\varepsilon^2}}$, a value
       $0 < \varepsilon < 1$
Output: Balls $B_1, \dots, B_k \subseteq \mathbb{R}^d$, such that $P \subseteq \bigcup_{j=1}^{k} B_j$
1  $S_i \leftarrow \emptyset$ for $i = 1, \dots, k$
2  $X \leftarrow \emptyset$                                   // points that have been covered so far
3  for $t = 1, \dots, \frac{16k}{\varepsilon^2}$ do
4      Let $p_t = \arg\max_{p \in P \setminus X} d(p, X)$
5      $S_{u_t} \leftarrow S_{u_t} \cup \{p_t\}$
6      if $|S_{u_t}| \geq 2$ then
7          $\mathrm{B}(c, r) \leftarrow (1+\varepsilon)$-approximation of $\mathrm{MB}(S_{u_t})$
8          $B_{u_t} \leftarrow \mathrm{B}(c, (1+\varepsilon')r)$
9          $X \leftarrow X \cup (B_{u_t} \cap P)$
10 forall $S_t = \{s_t\}$ do
11     $B_i \leftarrow \mathrm{B}(s_t, 0)$ ;
12 return $B_1, \dots, B_k$
```

$u \in \{1, \dots, k\}^{16k/\varepsilon^2}$, where the t-th entry u_t tells us which set S_{u_t} to assign the point p_t to that was picked in iteration t. By the above argument, the algorithm Selection needs to be invoked k^{16k/ε^2} times.
In Algorithm 3, we simply iterate over all such strings and call Selection for each of them. Afterwards, we check if the resulting clustering is feasible and cheaper than the current best found clustering.

Algorithm 3: Clustering for k-Center

```
Input: A set $P \subseteq \mathbb{R}^d$, a number $k \in \mathbb{N}$, a value $0 < \varepsilon < 1$
Output: A $(1+\varepsilon)$-approximative $k$-clustering $\mathscr{C}$ of $P$
1  $\varepsilon' \leftarrow \frac{\varepsilon}{3}$
2  $\mathscr{C} \leftarrow \{P, \emptyset, \dots, \emptyset\}$                       // feasible clustering to start with
3  forall $u \in \{1, \dots, k\}^{\frac{16k}{\varepsilon'^2}}$ do
4      $B_1, \dots, B_k \leftarrow$ Selection$(P, k, u, \varepsilon')$
5      $\mathscr{C}' \leftarrow C_1, \dots, C_k$, where $C_i = B_i \cap P$
6      if $\mathscr{C}'$ is a valid clustering and kCenter$(\mathscr{C}')$ < kCenter$(\mathscr{C})$ then
7          $\mathscr{C} \leftarrow \mathscr{C}'$
8  return $\mathscr{C}$
```

Combining all of the above, we get the following result, which differs slightly from the original paper.

Theorem 44 ([31])**.** *Algorithm 3 computes for every $1 > \varepsilon > 0$ a $(1+\varepsilon)$-approximation for* Euclidean k-Center *in time $\mathcal{O}\left(dn \cdot 2^{\mathcal{O}\left(\frac{k}{\varepsilon^2} \log \frac{k}{\varepsilon^5}\right)}\right)$.*

Proof. In the main for-loop of Algorithm 3, we call Selection for each of the k^{16k/ε'^2} possible strings, where $\varepsilon' = \varepsilon/3$. So let us first analyze the time needed for one call to Selection. The for-loop in Selection goes through $\frac{16k}{\varepsilon'^2}$ iterations. The computation of the approximate MEB can be done in $\mathcal{O}\left(\frac{|S_{u_t}|d}{\varepsilon'}\right) = \mathcal{O}\left(\frac{d}{\varepsilon'^3}\right)$ by using Yıldırım's algorithm [96], and everything else within one iteration can be done in $\mathcal{O}(dn)$. So the overall runtime of Selection is

$$\mathcal{O}\left(\frac{16k}{\varepsilon'^2} \cdot \left(\frac{d}{\varepsilon'^3} + dn\right)\right) = \mathcal{O}\left(\frac{kd}{\varepsilon'^5} + \frac{kdn}{\varepsilon'^2}\right) = \mathcal{O}\left(\frac{k}{\varepsilon'^5} \cdot dn\right).$$

Afterwards, we check the resulting clustering for feasibility and cost, which can be done in $\mathcal{O}(dn)$. So the runtime of one iteration of the loop is dominated by the call to Selection. Thus, the total runtime of Clustering is

$$\begin{aligned}\mathcal{O}\left(k^{\frac{16k}{\varepsilon'^2}} \cdot \frac{k}{\varepsilon'^5} \cdot dn\right) &= \mathcal{O}\left(dn \cdot 2^{\mathcal{O}\left(\log\left(\frac{k}{\varepsilon'^5} k^{16k/\varepsilon'^2}\right)\right)}\right) \\ &= \mathcal{O}\left(dn \cdot 2^{\mathcal{O}\left(\frac{k}{\varepsilon'^2} \log \frac{k}{\varepsilon'^5}\right)}\right).\end{aligned}$$

Now we want to express the runtime dependent on ε instead of ε'. Since $\varepsilon' = \varepsilon/3$, we have

$$\frac{1}{(\varepsilon')^p} = \left(\frac{3}{\varepsilon}\right)^p \in \mathcal{O}\left(\left(\frac{1}{\varepsilon}\right)^p\right).$$

Thus, we can conclude that the runtime is

$$\mathcal{O}\left(dn \cdot 2^{\mathcal{O}\left(\frac{k}{\varepsilon^2} \log \frac{k}{\varepsilon^5}\right)}\right).$$

By Corollary 43, we know that among the calls to Selection, there must be one where we assign points "correctly". In that case, each C_i^* is covered by some ball $\mathrm{B}(c_i, r_i)$, which is the $(1+\varepsilon')$-enlargement of a $(1+\varepsilon')$-appoximation of $\mathrm{MB}(C_i^*)$. This implies

$$\begin{aligned} r_i \leq (1+\varepsilon')^2 r_i^* &= \left(1 + \frac{\varepsilon}{3}\right)^2 r_i^* \\ &= \left(1 + \frac{2}{3}\varepsilon + \frac{\varepsilon^2}{9}\right) r_i^* \\ &< \left(1 + \frac{7}{9}\varepsilon\right) r_i^* < (1+\varepsilon) r_i^*, \end{aligned}$$

for all $i \in [k]$, where the second to last inequality holds because $\varepsilon \in (0,1)$. This, of course, also applies to the cluster with the largest radius, which makes our solution a

$(1+\varepsilon)$-approximate solution for k-Center. □

II.2.2. Adjusting the Algorithm for k-MSR

Now we take a look at how this algorithm can be adapted for k-Min-Sum-Radii. As already mentioned in Observation 40, one key insight is that not only the largest ball in the solution gets approximated, but that in fact every Opt-cluster gets approximated within a factor of $(1+\varepsilon)$. At a first glance it might seem like a good idea to just use the algorithm for k-Center without changing anything, hoping that we still get $(1+\varepsilon)$-approximations for each cluster in the optimal k-Min-Sum-Radii solution, which would of course also give a $(1+\varepsilon)$-approximation for the sum of the radii. Unfortunately, this does not work straight away. To get some intuition for why it does not work, let us quickly recap how the analysis for the k-Center algorithm works. It proceeds in three steps:

1. Show that the initial MEB (after a set stops being a singleton cluster) has a radius that is a constant fraction of the actual radius of the optimal cluster.
2. Show how the addition of new points to a set increases the radius of the MEB by a sufficient multiplicative factor.
3. Show that the number of steps needed to reach the stopping criterion only depends on k and ε.

The problem lies with step 1. For k-Center, we can argue via the properties of the objective function that the point that is farthest away from all current balls is sufficiently far away from the point that is already in the singleton cluster. In k-Min-Sum-Radii, we cannot use the same argument, as the objective behaves differently and therefore optimal solutions can look very different. The lower bound on the initial radius we can obtain in this way is too weak.
We can, however, work around this problem. Consider some singleton cluster $S_i = \{p\}$ during the execution of the algorithm. We want to assure that the second point that is assigned to S_i is sufficiently far away from the first point p, i.e. we want to rule out points that are too close. So a first idea might be to put a ball around p and to exclude all points within it from being picked in future iterations. But how to choose the radius of that ball? By Corollary 39, we know that we can generate a candidate set which is guaranteed to contain a near-optimal radius profile. Recall that this means that, for each radius r_i^* in the optimal radius profile, the near-optimal profile has a radius $\tilde{r}_i$ with $r_i^* \leq \tilde{r}_i \leq (1+\varepsilon)r_i^*$. We can then put a small ball around p whose radius is a small fraction of $\tilde{r}_i$. More precisely, we choose $(\varepsilon/(1+\varepsilon)\tilde{r}_i)$ as the initial radius. This enables us to get similar guarantees to those in Lemma 42 and Corollary 43. Importantly, for guessing the radius profile, we have assumed the optimum solution to be ε-balanced. We therefore are trying to approximate the optimum ε-balanced solution, which introduces an increase in the approximation factor by $(1+\varepsilon)$. In the following, when we use the term *optimum solution*, we mean an optimum ε-balanced solution.

The resulting algorithm (cf. Algorithm 5) is a modification of Algorithm 3. It gets a near-optimal radius profile as an additional input and then similarly iterates over all oracle strings. One difference is the bigger length (and therefore larger number) of these strings, which is a consequence of having an initial radius which is also dependent on ε (cf. Lemma 46). In Line 4, it invokes Algorithm 4, which is a modification of Algorithm 2. The two main differences are Lines 5 and 6. In Line 5, it places the small initial balls around the singleton clusters. In Line 6, the next point is simply chosen according to an arbitrary numbering of P, instead of the point that is farthest away from all covered points. For the k-Center algorithm, we needed to choose the next point in that way to establish the initial radius. Here, we do this by placing the small balls based on the radius guesses around each singleton.

Algorithm 4: Selection for k-MSR

```
Input: An ordered set $P \subseteq \mathbb{R}^d$, a number $k \in \mathbb{N}$, a string $u \in \{1,\dots,k\}^*$, a set
       $\{\widetilde{r}_1,\dots,\widetilde{r}_k\}$ of $k$ radii, a value $0 < \varepsilon < 1$
Output: Balls $B_1,\dots,B_k \subseteq \mathbb{R}^d$, such that $P \subseteq \bigcup_j B_j$
1  $S_i \leftarrow \emptyset$ for $i = 1,\dots,k$
2  $X \leftarrow \emptyset$                               // points that have been covered so far
3  for $i = 1,\dots,|u|$ do
4      $I = \{j \mid S_j = \{s_j\}$ is a singleton$\}$
5      $R \leftarrow \bigcup_{j\in I} \mathrm{B}(s_j, \frac{\varepsilon}{1+\varepsilon}\widetilde{r}_j)$          // put small balls around singletons
6      Let $p_i$ be the point from $P \setminus (X \cup R)$ that is first in the order induced by $P$
7      $S_{u_i} \leftarrow S_{u_i} \cup \{p_i\}$
8      if $|S_{u_i}| \geq 2$ then
9          $\mathrm{B}(c,r) \leftarrow (1+\varepsilon)$-approximation of $\mathrm{MB}(S_{u_i})$
10         $B_{u_i} \leftarrow \mathrm{B}(c,(1+\varepsilon)r)$
11         $X \leftarrow X \cup (B_{u_i} \cap P)$
12 for $S_i = \{s_i\}$ do
13     $B_i \leftarrow \mathrm{B}(s_i, 0)$ ;
14 return $B_1,\dots,B_k$
```

We keep the same notation as in the previous section. That is, we let S_i^t denote the state of set S_i in iteration t of Algorithm 4, and let c_i^t denote the center and r_i^t the radius of the $(1+\varepsilon)$-approximation of the minimum enclosing ball of S_i^t that the algorithm has computed. We now observe that Lemma 42 still holds, i.e., the balls that we compute still grow by at least the same factor each time a point is added to a set. No argument in the proof depends on the objective in any way. Since we still choose the points from outside the enlarged balls, and we still compute approximate MEBs, all of the arguments still hold.

Observation 45. *Let $t > 1$ be the iteration of Algorithm 4 in which a new point p is added to a non-singleton cluster S_i. We have $r_i^t \geq \left(1 + \frac{\varepsilon^2}{16}\right) \cdot r_i^{t-1}$.*

With this, we can again conclude that after not too many additions, any given optimal solution cluster will be covered.

Lemma 46. *For any index $i \in [k]$ it holds that after at most $\frac{32(1+\varepsilon)}{\varepsilon^3}$ additions to S_i we have $C_i^* \subset \mathrm{B}(c_i^t, r_i^t)$.*

Proof. We can argue similar to the proof of Corollary 43, the main difference being the fact that the initial radius is now also dependent on ε. Of course, it may happen that C_i^* is covered at an earlier point in time and that fewer points have to be added to S_i. Assume that we get at least to an iteration, where S_i contains two points. Recall that $\tilde{r}_i \in [r_i^*, (1+\varepsilon)r_i^*]$, for all $i \in [k]$. Since we ignored all points that were at a distance of at most $\frac{\varepsilon}{(1+\varepsilon)}\tilde{r}_i \geq \frac{\varepsilon}{1+\varepsilon}r_i^*$ from the first selected point, $\mathrm{MB}(S_i)$ has to have an initial radius of at least $\frac{\varepsilon}{2(1+\varepsilon)}\tilde{r}_i \geq \frac{\varepsilon\, r_i^*}{2(1+\varepsilon)}$. By Observation 45, any subsequent additions of new points to S_i further increase the radius by a multiplicative factor of at least $(1+\frac{\varepsilon^2}{16})$. Combining both of these observations gives us an upper bound on the number of iterations. First, note that the initial radius of $\frac{\varepsilon\, r_i^*}{2(1+\varepsilon)}$ grows by at least $\frac{\varepsilon^2}{16} \cdot \frac{\varepsilon\, r_i^*}{2(1+\varepsilon)} = \frac{\varepsilon^3\, r_i^*}{32(1+\varepsilon)}$ when the next point is added. Since the radii only grow larger, each subsequent update also increases the radius by at least this amount. At the same time, r_i^* is clearly an upper bound for the radius of $\mathrm{MB}(S_i)$. We can therefore bound the total number of updates by $\frac{32(1+\varepsilon)}{\varepsilon^3}$. □

With this, we can conclude that Algorithm 5 computes a $(1+\varepsilon)$-approximation for Euclidean k-Min-Sum-Radii in FPT time.

Lemma 47. *For every $0 < \varepsilon < 1$, Algorithm 5 computes a $(1+\varepsilon)$-approximation for Euclidean k-Min-Sum-Radii in time $\mathcal{O}\left(dn \cdot 2^{\mathcal{O}\left(\frac{k}{\varepsilon^3}\log\frac{k}{\varepsilon^6}\right)}\right)$.*

Proof. The analysis closely follows that of Algorithm 3 in Theorem 44. In the main for-loop of Algorithm 5, we call Selection for each of the $k^{32k(1+\varepsilon'\,/\,\varepsilon'^3}$ possible strings. The for-loop in Selection goes through $\frac{32k(1+\varepsilon')}{\varepsilon'^3}$[2] iterations and everything within one iteration can be done in $\mathcal{O}\left(\frac{d}{\varepsilon'^3} + dn\right)$. Afterwards, we set the radius of all singleton clusters to 0, which takes $\mathcal{O}(k) \subseteq \mathcal{O}(n)$. So the overall runtime for one call to Selection is

$$\mathcal{O}\left(\frac{32k}{\varepsilon'^3} \cdot \left(\frac{d}{\varepsilon'^3} + dn\right)\right) = \mathcal{O}\left(\frac{k}{\varepsilon'^6} \cdot dn\right).$$

Checking the resulting covering for cost and feasibility can also be done in $O(dn)$. So one iteration of the loop in Algorithm 5 is dominated by the call to Selection. The

[2]It should be noted that, when using $\mathcal{O}$-notation in this case, we are actually conducting two asymptotic analyses at the same time: we want to argue how the runtime depends on *increasing* n, k and d, and *decreasing* ε. Therefore, we can omit the factor of $(1+\varepsilon')$ in the numerator when using $\mathcal{O}$-notation, as this tends to 1.

loop goes through $k^{\frac{32k(1+\varepsilon')}{\varepsilon'^3}}$ iterations, leading to an overall runtime of

$$\begin{aligned}\mathcal{O}\left(k^{\frac{32k}{\varepsilon'^3}} \cdot \frac{k}{\varepsilon'^6} \cdot dn\right) &= \mathcal{O}\left(dn \cdot 2^{\mathcal{O}\left(\log\left(\frac{k}{\varepsilon'^6} k^{32k/\varepsilon'^3}\right)\right)}\right) \\ &= \mathcal{O}\left(dn \cdot 2^{\mathcal{O}\left(\frac{k}{\varepsilon'^3} \log \frac{k}{\varepsilon'^6}\right)}\right).\end{aligned}$$

Since $\varepsilon' = \varepsilon/4$, we have $\frac{1}{(\varepsilon')^p} = \left(\frac{4}{\varepsilon}\right)^p \in \mathcal{O}\left(\left(\frac{1}{\varepsilon}\right)^p\right)$ for all $p \in \mathbb{R}_{\geq 0}$. Thus, we can conclude that the runtime is

$$\mathcal{O}\left(dn \cdot 2^{\mathcal{O}\left(\frac{k}{\varepsilon^3} \log \frac{k}{\varepsilon^6}\right)}\right).$$

By Corollary 46, we know that among the calls to Selection, there must be one where we assign points "correctly". Recall that we are approximating an optimum ε-balanced solution. For $i \in [k]$, let r_i be the corresponding radius of our computed solution, r_i^{bal} the radius of the optimum ε-balanced solution, and r_i^* the radius of the actual optimum solution. We have

$$\begin{aligned} r_i &\leq (1+\varepsilon')^2 r_i^{\text{bal}} \leq (1+\varepsilon')^3 r_i^* = (1+\frac{\varepsilon}{4})^3 r_i^* \\ &= \left(1 + \frac{3}{4}\varepsilon + \frac{3}{16}\varepsilon^2 + \frac{1}{64}\varepsilon^3\right) r_i^* \\ &\leq \left(1 + \frac{61}{64}\varepsilon\right) r_i^* < (1+\varepsilon) r_i^*, \end{aligned}$$

where the second to last inequality holds because $\varepsilon \in (0,1)$. Since this holds for all $i \in [k]$, it follows that

$$\sum_{i=1}^k r_i < (1+\varepsilon) \sum_i^k r_i^*,$$

proving the approximation ratio. □

Theorem 48. *There exists an algorithm that computes for every $1 > \varepsilon > 0$ a $(1+\varepsilon)$-approximation for Euclidean k-Min-Sum-Radii in FPT time.*

Proof. By Lemma 47, Algorithm 5 computes a $(1+\varepsilon)$-approximate solution in

$$\mathcal{O}\left(dn \cdot 2^{\mathcal{O}\left(\frac{k}{\varepsilon^3} \log \frac{k}{\varepsilon^6}\right)}\right),$$

which is FPT. As it expects a near-optimal radius profile as input, we generate a candidate set as described in Section II.1.3.1, which, by Corollary 39, has size

$$\mathcal{O}\left(\left(\log_{(1+\varepsilon)} \frac{k}{\varepsilon}\right)^k\right),$$

Algorithm 5: Clustering for k-MSR

```
Input: A set $P \subseteq \mathbb{R}^d$, a number $k \in \mathbb{N}$, a near optimal radius profile
       $(\tilde{r}_1, \dots, \tilde{r}_k)$, a number $\varepsilon > 0$
Output: A $(1+\varepsilon)$-approximate $k$-clustering $\mathscr{C}$ of $P$
1 $\varepsilon' \leftarrow \frac{\varepsilon}{4}$
2 $\mathscr{C} \leftarrow \{P, \emptyset, \dots, \emptyset\}$            // A feasible clustering to start with
3 forall $u \in \{1, \dots, k\}^{\frac{32k(1+\varepsilon')}{\varepsilon'^3}}$ do
4     $B_1, \dots, B_k \leftarrow$ Selection$(P, k, u, (\tilde{r}_1, \dots, \tilde{r}_k), \varepsilon')$
5     $\mathscr{C}' \leftarrow \{C_1, \dots, C_k\}$, where $C_i = B_i \cap P$
6     if $\mathscr{C}'$ is a valid clustering and kMSR$(\mathscr{C}') <$ kMSR$(\mathscr{C})$ then
7         $\mathscr{C} \leftarrow \mathscr{C}'$
8 return $\mathscr{C}$
```

which is also FPT. Then we call Algorithm 5 for each candidate, leading to an overall runtime of

$$\mathcal{O}\left(\left(\log_{(1+\varepsilon)} \frac{k}{\varepsilon}\right)^k \cdot dn \cdot 2^{\mathcal{O}\left(\frac{k}{\varepsilon^3} \log \frac{k}{\varepsilon^6}\right)}\right) = \mathcal{O}(\text{poly}(n, d) \cdot f(k, \varepsilon)).$$

□

II.3. A Constant-Factor FPT-Approximation for k-Min-Sum-Radii with Mergeable Constraints

We now turn to the more general metric setting, i.e., our input points live in some metric space that is not necessarily Euclidean. This means that we also deal with discrete clustering, i.e., the centers have to be a subset of the input points. Additionally, we will introduce mergeable constraints to our problem (cf. Definition 22), giving us the following formal problem description.

Metric k-MSR Problem with Mergeable Constraints

Input: A set P of n points, a metric distance function $d : P \times P \to \mathbb{R}_{\geq 0}$, a number $k \in \mathbb{N}, k \leq n$

Output: A center set $\mathcal{C} = \{c_1, \dots, c_k\}$ and an assignment $\sigma : P \to \mathcal{C} \subseteq P$ of size k, such that every induced cluster satisfies the mergeable constraint and the induced radii $r_1, \dots, r_k$ minimize $\sum_{i=1}^{k} r_i$

As mentioned in Section II.1.2.1, the unconstrained problem is already NP-hard. The current best approximation ratios are $(2 + \varepsilon)$ for FPT algorithms [35], and $(3 + \varepsilon)$ for polynomial time algorithms [32]. For uniform lower bounds (which is a mergeable constraint) the ratios are $(3 + \varepsilon)$ for FPT [33] and $(3.5 + \varepsilon)$ for polynomial time algorithms [32]. For group fairness, at the time of writing (January 2026), the best known approximation ratio is $(4 + \varepsilon)$ for FPT algorithms [18], while no polynomial time approximation algorithms are known.

II.3.1. Outline of the Algorithm

One can think of the algorithm as working in three steps:

(1) Guess a near-optimal radius profile.

(2) Successively build a covering of P with certain desirable properties and whose sum of radii is at most $(3 + \varepsilon)$ times the cost of an optimum.

(3) Merge overlapping balls of that covering to obtain a feasible solution.

Step (1) will be done as described in Section II.1.3. Since we are in the discrete setting, we can either utilize Observation 30 or Lemma 35[1] for finding a candidate for the largest radius. In any case, we get a reasonably sized candidate set in FPT time. Note that this, again, means that we are actually approximating the optimum ε-balanced solution, which is only a $(1+\varepsilon)$-approximation for the optimum.

In step (2), we want to successively choose centers and place radii around them such that each resulting ball is guaranteed to entirely cover at least one optimal solution cluster. This also involves guessing and requires the introduction of a variant on the k-Center problem, called k-Center Completion Problem. We define this problem and show how to solve it approximately in polynomial time in Section II.3.2. In Section II.3.3, we show how we can utilize this to obtain a covering (cf. Definition 3) of the input points which, in general, does not constitute a feasible solution yet. However, the cost of that covering is guaranteed to be a $(3+\varepsilon)$-approximation to the optimal solution cost.

Step (3) is presented in two variants. The first, presented in Section II.3.4, is taken from the paper *FPT Approximations for Fair k-Min-Sum-Radii* [33] and leads to a $(6-\frac{3}{k}+\varepsilon)$-approximation, while the second, presented in Section II.3.5 was introduced by Bandyapadhyay and Chen [18] and leads to a $(4+\varepsilon)$-approximation.

II.3.2. The k-Center Completion Problem

We now present a variation on the k-Center Problem where we are already given a partial solution and want to complete it. To this end, we first define the following altered distance function. Consider an instance $\mathscr{I} = (P, d, k)$ for k-Center and suppose we already have a partial solution, given by a set of centers $\{c_1, \dots, c_\ell\}$ and radii $\{r_1, \dots, r_\ell\}$, where $1 \leq \ell < k$. Then we define

$$d'(x,y) = \begin{cases} \max\{d(x,y) - r_i, 0\} & \text{if } x \in P \setminus \{c_1, \dots, c_\ell\},\ y = c_i, i \in \{1, \dots, \ell\} \\ \max\{d(x,y) - r_i - r_j, 0\} & \text{if } x = c_i,\ y = c_j \text{ with } i, j \in \{1, \dots, \ell\} \\ d(x,y) & \text{otherwise.} \end{cases}$$

Let us unpack this definition. For a non-center point $x \in P \setminus \{c_1, \dots, c_\ell\}$, the distance to a center c_i gets reduced by the corresponding radius r_i, cutting off at 0 (so d' can never be negative). This effectively means that all points within $\mathrm{B}(c_i, r_i)$ (which are already covered by this partial solution) have distance 0 to c_i under d'. The intuition behind this is that we already have the possibility to cover this point "for free" using the partial solution. For a non-center point x which is not contained in $\mathrm{B}(c_i, r_i)$ for any $i \in [\ell]$, the distance under d' to any c_i is now only the distance to the "edge" of the ball, encapsulating the fact that we could cover x by increasing r_i by only that amount. Similarly, for two centers c_i and c_j, we reduce their cost by *both* r_i and r_j (stopping at

[1] if we have a constant factor approximation algorithm for the k-Center version of our constrained clustering problem

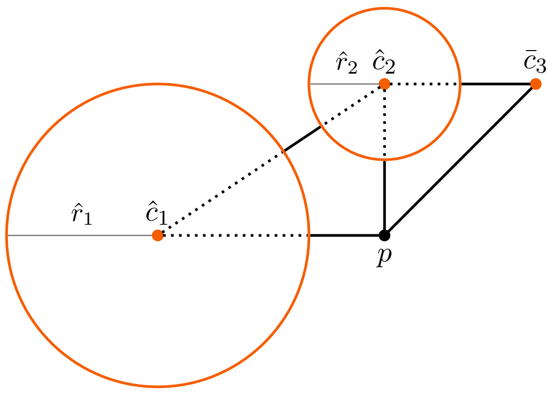


Figure II.3.1.: An instance of a 3-center completion problem. The centers $\hat{c}_1$ and $\hat{c}_2$ with corresponding radii $\hat{r}_1 = 1$ and $\hat{r}_2 = 0.5$ are already given. The underlying distances are given by $d(p, \hat{c}_1) = 1.5$, $d(p, \hat{c}_2) = 1$, $d(p, \bar{c}_3) = \sqrt{2}$. In d', all distances to one of the centers $\hat{c}_1, \hat{c}_2$ are shortened by the respective radius $\hat{r}_1, \hat{r}_2$. Dotted parts indicate the segments that do not contribute to the distance d'. For example, $d'(p, \hat{c}_1) = 0.5$, $d'(p, \hat{c}_2) = 0.5$. However, distances not involving $\hat{c}_1$ or $\hat{c}_2$ as one of the end points stay the same, i.e., $d'(p, \bar{c}_3) = d(p, \bar{c}_3)$. Originally, the point p is closer to $\bar{c}_3$ than to $\hat{c}_1$. But under d', p is closer to $\hat{c}_1$. This example also shows that the distance d' does not fulfill the triangle inequality: While $d'(p, \bar{c}_3) = \sqrt{2}$, the detour via $\hat{c}_2$ is shorter: $d'(p, \hat{c}_2) + d(\hat{c}_2, \bar{c}_3) = 1$.

0), while for all points that are neither a center or covered by a ball yet their distance stays unchanged.

Figure II.3.1 illustrates the change of the distance function and highlights the fact that d' does not satisfy the triangle inequality anymore. We can now formalize this problem as follows.

k-Center Completion Problem

Input: A set P of n points, a metric distance function $d : P \times P \to \mathbb{R}_{\geq 0}$, a number $k \in \mathbb{N}$ with $k \leq n$, a set of $\ell < n$ centers $\mathcal{C}_{\text{in}} = \{c_1, \dots, c_\ell\}$ and radii $\{r_1, \dots, r_\ell\}$
Output: A center set $\mathcal{C} = \{c_1, \dots, c_k\}$ with $\mathcal{C} \supseteq \mathcal{C}_{\text{in}}$ and an assignment $\sigma : P \to \mathcal{C}$, such that $\max_{p \in P} d'(p, \sigma(p))$ is minimized

Even though the triangle inequality is violated, we can show that a slightly adapted version of Gonzalez' Algorithm (cf. Algorithm 1) still yields a 2-approximation for the k-Center Completion Problem. Algorithm 6 simply computes d' according to the given centers and radii and then proceeds in a farthest-first manner like Gonzalez, with respect to that new distance function. In the end, every point gets assigned to its closest center.

Lemma 49. *Algorithm 6 computes a 2-approximation for the* k-Center Completion Problem *in time $\mathcal{O}(kn)$.*

Algorithm 6: FARTHEST-FIRST-TRAVERSAL-COMPLETION

```
Input: Point set $P$, distance function $d$, integer $k$, centers $c_1, \dots, c_i$, radii
       $r_1, \dots, r_i$
Output: Set of $k$ centers, assignment $\sigma$
// Update distance function
$d' \leftarrow d$
for $j = 1, \dots, i$ do
    for $p \in P$ do
        $d'(c_j, p) \leftarrow \max\{d(c_j, p) - r_j, 0\}$
// Complete centers by farthest-first-traversal
for $j = i + 1, \dots, k$ do
    $c_{i+1} \leftarrow \arg\max_{p \in P} \min_{c \in \{c_1, \dots, c_i\}} d'(p, c)$
// Assign points to their closest centers
for $p \in P$ do
    $\sigma(p) \leftarrow \arg\min_{c \in \{c_1, \dots, c_k\}} d'(p, c)$
return $c_1, \dots, c_k$, $\sigma$
```

Proof. We follow the proof for the approximation guarantee of Gonzalez' algorithm and verify that it still works in the case of the k-CENTER COMPLETION problem. Let D be the maximum distance of any point to its closest point in $\{c_1, \dots, c_k\}$ with respect to d', i.e., D is the cost of the solution computed by FARTHEST-FIRST-TRAVERSAL-COMPLETION with $c_1, \dots, c_\ell$ already fixed. Let c_{k+1} be a point with $\min_{i \in [k]} d'(c_{k+1}, c_i) = D$. Observe that all c_i with $i \geq \ell+1$ satisfy that $d'(c_i, c_j) \geq D$ for all $j \in \{1, \dots, \ell\}$ because otherwise c_{k+1} would have been chosen as a center since its minimum distance is D. Inductively, we also get for all c_i, c_j with $i, j \geq \ell + 1$ that $d'(c_i, c_j) \geq D$ is true because otherwise c_{k+1} would have been chosen. Now we get to the point where the proof differs slightly from the original proof because we need to make a case distinction. We have $k+1$ points $c_1, \dots, c_{k+1}$. In an optimum solution for the somewhat different k-center problem, we can assume that every point is assigned to its closest center, and in particular, all centers are assigned to themselves. There is always an optimum solution that satisfies this. Let $c^*_{\ell+1}, \dots, c^*_k$ and $\sigma^* \colon P \to \{c_1, \dots, c_\ell, c^*_{\ell+1}, \dots, c^*_k\}$ be such an optimal solution, i.e., $\sigma^*(c_i) = c_i$ for all $i \in [\ell]$.

Case 1: there exists $i \in \{\ell+1, \dots, k+1\}$ with $\sigma^*(c_i) = c_j \in \{c_1, \dots, c_\ell\}$, i.e., one of the points we picked as a center or the additional point c_{k+1} is assigned to one of the predefined centers in the optimum solution. In this case, $\mathrm{OPT} \geq d'(c_i, c_j) \geq D$, since we argued that all our centers have distance of at least D to the predefined centers.

Case 2 is that none of $c_{\ell+1}, \dots, c_{k+1}$ is assigned to any predefined center. Thus, they are all assigned to the $k - \ell$ centers $c^*_{\ell+1}, \dots, c^*_k$. By the pigeonhole principle, this means that $\sigma^*(c_i) = \sigma^*(c_j) = c^*_m$ for some $i, j \in \{\ell + 1, \dots, k + 1\}, i \neq j$, and $m \in \{\ell+1,\dots,k\}$. Since $i,j > [\ell]$, $d'(c_i,c_j) \geq D$ as argued above. Also since $i,j,m > [\ell]$, by definition, $d'(c_i,c_j) = d(c_i,c_j)$, $d'(c_i,c_m^*) = d(c_i,c_m^*)$ and $d'(c_j,c_m^*) = d(c_j,c_m^*)$.

We conclude by the triangle inequality that

$$\begin{aligned} D \leq d'(c_i,c_j) = d(c_i,c_j) &\leq d(c_i,c_m^*) + d(c_m^*,c_j) \\ &\leq 2 \max_{g=i,j}\{d(c_g,\sigma^*(c_g))\} \\ &\leq 2 \max_{g\geq \ell+1}\{d(c_g,\sigma^*(c_g))\}. \end{aligned}$$

As $d(c_g,\sigma^*(c_g)) = d'(c_g,\sigma^*(c_g))$ for all $g \geq \ell+1$ by definition, this implies $\mathrm{OPT} \geq \frac{1}{2}D$.

Regarding the runtime, we observe that updating the distance function takes $\mathcal{O}(kn)$: the number of given centers i is at most k and we only need to recompute distances between these centers and the rest of the input points, making for $i \cdot n \leq k \cdot n$ computations. The remainder of Algorithm 6 is just Gonzalez' Algorithm with $k' := k - i$. By Lemma 12, this can be done in $\mathcal{O}((k-i)n)$. The claim follows. □

II.3.3. Obtaining a Preliminary Covering

With the methods for radius guessing and k-Center Completion in place, we can describe the algorithm to obtain a covering from which we will then derive the actual solution.

In the following, we fix an optimum solution for k-Min-Sum-Radii with mergeable constraints that we would like to approximate. We assume this solution to be ε-balanced (cf. Definition 25) which – according to Observation 26 – only increases the cost by a factor of at most ε. This allows us to compute the candidates for a near-optimal radius profile as described in Section II.1.3. We first fix some notation:

- $C_1^*,\dots,C_k^*$ denote the clusters of this optimal ε-balanced solution, with centers $c_1^*,\dots,c_k^*$ and radii $r_1^*,\dots,r_k^*$. We assume without loss of generality that $r_1^* \geq r_2^* \geq \ldots \geq r_k^*$.
- $(\tilde{r}_1,\dots,\tilde{r}_k)$ denotes a near-optimal radius profile (with respect to the above optimum ε-balanced solution).
- $\hat{c}_1,\dots,\hat{c}_i$ and $\hat{r}_1,\dots,\hat{r}_i$ denote the centers and radii computed by the algorithm up until iteration i.

The general idea of the algorithm is as follows: Assume that, in the beginning of iteration i, we already fixed candidate centers $\hat{c}_1,\dots,\hat{c}_{i-1}$ and candidate radii $\hat{r}_1,\dots,\hat{r}_{i-1}$. We compute a 2-approximation for the induced k-center completion instance. The resulting output consists of centers $\hat{c}_1,\dots,\hat{c}_{i-1},\bar{c}_i,\dots,\bar{c}_k$, radii $\hat{r}_1,\dots,\hat{r}_{i-1},\bar{r}_i,\dots,\bar{r}_k$ and an assignment σ. We guess $\sigma(c_i^*)$, i.e. where the i-th center of the *optimal* solution is assigned to in the k-center completion solution. Recall that we already have a good approximation for $\tilde{r}_i$ for the corresponding optimal solution radius r_i^*. If we guess that $\sigma(c_i^*)$ is among the newly chosen centers, i.e. if $\sigma(c_i^*) = \bar{c}_j \in \{\bar{c}_i, \dots, \bar{c}_k\}$, we open a new ball with radius $\hat{r}_i = 3\tilde{r}_i$ at this center $\hat{c}_i \coloneqq \bar{c}_j$. Otherwise, if we guess that $\sigma(c_i^*) = \hat{c}_j \in \{\hat{c}_1, \dots, \hat{c}_{i-1}\}$, there already exists a ball around this center, and we only need to enlarge this ball by $3\tilde{r}_i$. In order to have k centers in the end, we set $\hat{c}_i$ to some arbitrary point and $\hat{r}_i = 0$.

The guessing of the center assignments can be handled as described in Section II.1.3. In every iteration i, we have k possible choices for $\sigma(c_i^*)$. So each sequence of k "guesses" can be encoded by a tuple $a = (a_1, \dots, a_k) \in \{1, \dots, k\}^k$, where $a_i = \ell$ means that in the i-th iteration, we choose the ℓ-th center of the k-center completion solution as $\sigma(c_i^*)$ (i.e. $\hat{c}_\ell$ if $\ell < i$, or $\bar{c}_\ell$ if $\ell \geq i$). Thus, we can emulate the guessing by generating all such tuples upfront, computing the candidate balls for each of these, and choosing the best feasible one in the end. For a formal description of the algorithm, see Algorithm 7.

Algorithm 7: CENTERS-AND-RADII

```
Input: Points $P$, distances $d$, $k \in \mathbb{N}$, radius profile $(\tilde{r}_1, \dots, \tilde{r}_k)$, tuple
       $(a_1, \dots, a_k)$
Output: Set of $k$ centers, set of $k$ radii
1  $I_0 \leftarrow (P, d, k, \emptyset, \emptyset)$
2  for $i = 1, \dots, k$ do
3      $(\mathcal{C}_{kcc}, \sigma) \leftarrow$ FARTHEST-FIRST-TRAVERSAL-COMPLETION$(I_{i-1})$ , where
       $\mathcal{C}_{kcc} = \{\hat{c}_1, \dots, \hat{c}_{i-1}, \bar{c}_i, \dots, \bar{c}_k\}$ and $\sigma \colon P \to \mathcal{C}_{kcc}$
4      if $a_i < i$ then
5          // guess that $\sigma(c_i^*) \in \{\hat{c}_1, \dots, \hat{c}_{i-1}\}$
6          Set $\hat{r}_{a_i} \leftarrow \hat{r}_{a_i} + 3\tilde{r}_i$, choose $\hat{c}_i$ arbitrarily, set $\hat{r}_i \leftarrow 0$
7      else if $a_i \geq i$ then
8          // guess that $\sigma(c_i^*) \in \{\bar{c}_i, \dots, \bar{c}_k\}$
9          Set $\hat{c}_i \leftarrow \bar{c}_{a_i}, \hat{r}_i \leftarrow 3\tilde{r}_i$
10     $I_i \leftarrow (P, d, k, \{\hat{c}_1, \dots, \hat{c}_i\}, \{\hat{r}_1, \dots, \hat{r}_i\})$
11 return $\{\hat{c}_1, \dots, \hat{c}_k\}$, $\{\hat{r}_1, \dots, \hat{r}_k\}$
```

With this notation and Algorithm 7 in place, we can now formally define what it means to guess correctly.

Definition 50 (Guessing correctly)**.** *Let $(\mathcal{C}_{kcc}, \sigma)$ be a solution for the k-center completion problem with input $\hat{c}_1, \dots, \hat{c}_{i-1}$ and $\hat{r}_1, \dots, \hat{r}_{i-1}$, where $\mathcal{C}_{kcc} = \{\hat{c}_1, \dots, \hat{c}_k\}$. We say that Algorithm 7* guesses correctly *if the input tuple $a = (a_1, \dots, a_k)$ satisfies the following: for all $i \in [k]$, a_i is the smallest index such that $\hat{c}_{a_i} = \sigma(c_i^*)$, where c_i^* is the center of optimal cluster C_i^*.*

In other words, the algorithm guesses correctly if every entry a_i of a is actually the correct index of the center $\hat{c}_{a_i}$ to which the i-th center c_i^* is assigned in $(\mathcal{C}_{kcc}, \sigma)$. The crucial point of Algorithm 7 is that it fully covers one so-far uncovered optimal cluster in every iteration, if the initial radius profile is near-optimal and Algorithm 7 guesses correctly. For the analysis, we need the following Lemma that bounds the cost of an optimal k-center completion solution in any iteration of Algorithm 7 by the radius of the largest remaining optimal cluster that is not fully covered yet. Combining with Lemma 49 gives an upper bound on the distance between an optimal center c^* and the center $\sigma(c^*)$ it is assigned to.

Lemma 51. *Assume that, up to the end of iteration i, Algorithm 7 chose centers $\hat{c}_1, \ldots, \hat{c}_i$ and radii $\hat{r}_1, \ldots, \hat{r}_i$ such that, for all $p \in \bigcup_{j \le i} C_j^*$, there exists a center $\hat{c}_\ell \in \{\hat{c}_1, \ldots, \hat{c}_i\}$ with $d'(p, \hat{c}_\ell) = 0$. Then, the value of an optimal solution for* k-Center Completion *with input $\{\hat{c}_1, \ldots, \hat{c}_i\}, \{\hat{r}_1, \ldots, \hat{r}_i\}$ is at most r_{i+1}^*.*

Proof. Consider extending $\{\hat{c}_1, \ldots, \hat{c}_i\}$ by the optimal solution centers $\{c_{i+1}^*, \ldots, c_k^*\}$. By the precondition of the lemma, we can assign every point in $p \in \bigcup_{j \le i} C_j^*$ to a center in $\{\hat{c}_1, \ldots, \hat{c}_i\}$ at distance 0 with respect to d'. For all $h \ge i+1$, any $x \in C_h^*$ can be assigned to c_h^* at distance $\le r_h^*$. As the optimal radii are sorted in decreasing order, r_{i+1}^* is the largest remaining radius among the optimal clusters under d'. Hence, the resulting assignment σ' satisfies $d'(x, \sigma'(x)) \le r_{i+1}^*$ for all $x \in P$. Notice that $\{\hat{c}_1, \ldots, \hat{c}_i, c_{i+1}^*, \ldots, c_k^*\}$ and σ' form a feasible solution for k-Center, and that the maximum radius of this solution (under d') is r_{i+1}^* as argued above. Hence, the optimum value for k-Center Completion is upper bounded by r_{i+1}^*. □

Let $\hat{\mathcal{B}} = \{B(\hat{c}_1, \hat{r}_1), \ldots, B(\hat{c}_k, \hat{r}_k)\}$ be the balls resulting from the output of Algorithm 7. Next, we want to establish that (1) for every optimum solution cluster C_i^*, there exists a ball $B(\hat{c}_i, \hat{r}_i) \in \hat{\mathcal{B}}$ that completely contains it, and (2) for every $B(\hat{c}_i, \hat{r}_i) \in \hat{\mathcal{B}}$ with non-zero radius, there exists an optimal solution cluster C_j^* that is completely contained in it. The next Lemma formalizes this.

Lemma 52. *Assume that our guess of the initial radius profile is near-optimal and that Algorithm 7 guesses correctly. Let $\hat{\mathcal{B}}$ denote the set of balls $B(\hat{c}_1, \hat{r}_1), \ldots, B(\hat{c}_k, \hat{r}_k)$ found by Algorithm 7. Then the following two statements hold true*

1. *for all $j \le k$, there exists $\ell \le k$ such that $C_j^* \subseteq B(\hat{c}_\ell, \hat{r}_\ell)$*
2. *for all $\ell \le k$, if $r_\ell > 0$ then there exists $j \le k$ such that $C_j^* \subseteq B(\hat{c}_\ell, \hat{r}_\ell)$.*

Proof. We show that the statement holds at the end of every iteration of Algorithm 7. That is, we show that for all $i \le k$, the following holds

(1) for all $j \le i$, there exists $\ell \le i$ such that $C_j^* \subseteq B(\hat{c}_\ell, \hat{r}_\ell)$

(2) for all $\ell \le i$, if $r_\ell > 0$ then there exists $j \le i$ such that $C_j^* \subseteq B(\hat{c}_\ell, \hat{r}_\ell)$.

Then setting $i = k$ implies the result. We prove this via induction over $i \le k$. For $i = 0$, the statements are trivially fulfilled. Now let the statement be fulfilled at the end of iteration $i - 1 < k$ for some $i > 0$.

By Lemma 51, $\mathrm{OPT}_{kcc} \leq r_i^*$, where OPT_{kcc} is the value of an optimal solution for the k-center completion problem that takes the centers and radii generated until the end of iteration $i-1$ as input. By Lemma 49, FARTHEST-FIRST-TRAVERSAL-COMPLETION in Line 3 of Algorithm 7 computes a 2-approximation for the k-center completion problem. These two arguments together imply $d(c_i^*, \sigma(c_i^*)) \leq 2\mathrm{OPT}_{kcc} \leq 2r_i^*$.

If $c_i^* = \hat{c}_j$ for some $j \leq i-1$, then for all $p \in C_i^*$, it is $d(p, \hat{c}_j) = d(p, c_i^*) \leq r_i^* \leq r_j^*$, where the last inequality holds because the optimal radii are sorted decreasingly.

If Algorithm 7 guesses correctly, $c_{a_i} = \sigma(c_i^*) = c_i^* = \hat{c}_j$ and the radius $\hat{r}_j^{\text{new}}$ produced in Line 6 fulfills $\hat{r}_j^{\text{new}} := \hat{r}_j + 3\tilde{r}_i \geq r_i^*$. Therefore C_i^* is covered completely by $B(\hat{c}_j, \hat{r}_j^{\text{new}})$. This implies that (1) holds. For index i, the algorithm creates a new ball with radius $\hat{r}_i = 0$. Therefore, statement (2) is fulfilled by the induction hypothesis.

Now, we assume that $c_i^* \notin \{\hat{c}_1, \ldots, \hat{c}_{i-1}\}$. There are two cases for the guess a_i.

1. $a_i < i$: Then, we are in Line 6 of Algorithm 7 and enlarge an already existing ball centered at $\sigma(c_i^*) = \hat{c}_{a_i}$ by $3\tilde{r}_i$, i.e., the a_i-th ball is $B(\hat{c}_{a_i}, \hat{r}_{a_i} + 3\tilde{r}_i)$ at the end of the iteration. For every $p \in C_i^*$,

$$\begin{aligned} d(p, \hat{c}_{a_i}) &\leq d(p, c_i^*) + d(c_i^*, \hat{c}_{a_i}) \leq d(p, c_i^*) + d'(c_i^*, \hat{c}_{a_i}) + \hat{r}_{a_i} \\ &= d(p, c_i^*) + d'(c_i^*, \sigma(c_i^*)) + \hat{r}_{a_i}. \end{aligned}$$

   It is $d(p, c_i^*) \leq r_i^*$ as $p \in C_i^*$, and $d'(c_i^*, \sigma(c_i^*)) \leq 2r_i^*$ as σ is the assignment given by the 2-approximation. Further, $r_i^* \leq \tilde{r}_i$. Overall, $d(p, \hat{c}_{a_i}) \leq 3\tilde{r}_i + \hat{r}_{a_i}$, which implies that the ball $B(\hat{c}_{a_i}, \hat{r}_{a_i} + 3\tilde{r}_i)$ covers C_i^* completely.

2. $a_i \geq i$: In this case, the algorithm creates a new ball $B(\hat{c}_i, \hat{r}_i)$ with $\hat{c}_i := c_{a_i} = \sigma(c_i^*)$ and $\hat{r} := 3\tilde{r}_i$. For every $p \in C_i^*$,

$$d(p, \hat{c}_i) = d(p, \sigma(c_i^*)) \leq d(p, c_i^*) + d(c_i^*, \sigma(c_i^*)) = d(p, c_i^*) + d'(c_i^*, \sigma(c_i^*)),$$

   where the last equality holds because $c_i^* \notin \{\hat{c}_1, \ldots, \hat{c}_{i-1}\}$ and a_i is the smallest index such that $c_{a_i} = \sigma(c_i^*)$, which implies $c_{a_i} \notin \{\hat{c}_1, \ldots, \hat{c}_{i-1}\}$. Again, $d(p, c_i^*) \leq r_i^*$ and $d'(c_i^*, \sigma(c_i^*)) \leq 2r_i^*$. Overall, $d(p, \hat{c}_i) \leq 3r_i^* \leq 3\tilde{r}_i$, and hence, C_i^* is completely covered by $B(\hat{c}_i, \hat{r}_i)$.

□

This directly implies the existence of a mapping of optimal solution clusters to output balls with non-zero radius.

Corollary 53. *Let $\hat{\mathcal{B}}_{>0}$ denote the set of balls in $\hat{\mathcal{B}}$ with non-zero radius. If the radius profile is near-optimal and Algorithm 7 guesses correctly, then there exists a mapping $\varphi : \{C_1^*, \ldots, C_k^*\} \to \hat{\mathcal{B}}_{>0}$, such that $\forall i \in [k] : C_i^* \subseteq \varphi(C_i^*)$.*

Notice that the candidate balls might overlap, but the optimal clusters are pairwise disjoint by definition of a clustering. This is dealt with in the next section. The following lemma relates the total cost of the clustering consisting of the candidate balls to the cost of an optimal solution for k-Min-Sum-Radii with mergeable constraints. This will be useful for analyzing the cost of our final solution later on. Notice that this statement does not imply an approximation ratio for the vanilla k-Min-Sum-Radii.

Lemma 54. *Let $\hat{r}_1, \dots, \hat{r}_k$ be the radii produced by Algorithm 7. Then*

$$\sum_{j=1}^{k} \hat{r}_j \leq 3(1+\varepsilon) \sum_{j=1}^{k} r_j^*.$$

Proof. We show by induction that $\sum_{j=1}^{i} \hat{r}_j \leq 3 \cdot \sum_{j=1}^{i} \tilde{r}_j$ for all $i \leq k$. Then for $i = k$, the result follows since $\tilde{r}_j \leq (1+\varepsilon) r_j^*$ for all $j \leq k$.

For $i = 0$, the statement trivially holds. Now assume that the statement holds for $i-1$. Either the algorithm sets $\hat{r}_i := 3\tilde{r}_i$ during iteration i. Then, $\sum_{j=1}^{i} \hat{r}_j = \sum_{j=1}^{i-1} \hat{r}_j + \hat{r}_i \leq 3 \sum_{j=1}^{i-1} \tilde{r}_j + 3\tilde{r}_i$. For the remaining case, let $\hat{r}_j^{(i-1)}$ denote the value of $\hat{r}_j$ at the beginning of the ith iteration, and $\hat{r}_j^{(i)}$ its value at the end of the iteration, $j \leq i$. There exists $\ell < i$ such that the algorithm sets $\hat{r}_\ell^{(i)} := \hat{r}_\ell^{(i-1)} + 3\tilde{r}_i$ and $\hat{r}_i^{(i)} := 0$. Then, $\sum_{j=1}^{i} \hat{r}_j^{(i)} = \sum_{j=1}^{i-1} \hat{r}_j^{(i)} = \sum_{j=1}^{i-1} \hat{r}_j^{(i-1)} + 3\tilde{r}_i \leq 3 \cdot \sum_{j=1}^{i} \tilde{r}_j$. □

II.3.4. Finding the Assignment

As was established in the previous section, we have obtained a collection of balls $\hat{\mathcal{B}} = \{\mathrm{B}(\hat{c}_1, \hat{r}_1), \dots, \mathrm{B}(\hat{c}_k, \hat{r}_k)\}$ that has two properties:

(1) the sum of their radii is a $(3+\varepsilon)$-approximation for the optimal solution

(2) for each optimal solution cluster C^*, there exists a ball $\mathrm{B}(\hat{c}, \hat{r}) \in \hat{\mathcal{B}}$ such that $C^* \subseteq \mathrm{B}(\hat{c}, \hat{r})$.

Since the balls might overlap, this does not imply an assignment straight away. Arbitrarily assigning points within the balls is also not feasible, since this might violate the constraints. Therefore, we describe a procedure that *merges* overlapping balls into one cluster without increasing the cost too much.

We construct a graph from the center and radii candidates computed in the first part of the algorithm and observe that the clustering induced by the connected components of this graph fulfills the given mergeable constraint. We define the *access graph* $G = (V, E)$ as follows. The set of vertices corresponds to the given point set, i.e. $V = P$. We add an edge between any pair of vertices $x, y \in V$ if and only if $x = \hat{c}_i$ for a center $\hat{c}_i$ constructed in Algorithm 7 and $d(y, \hat{c}_i) \leq \hat{r}_i$ for the corresponding radius $\hat{r}_i$. The construction is exemplified for exact fairness constraints in Figure II.3.2. Let $\mathrm{CC}(G)$ denote the set of connected components (cf. Definition 17) of G. The first key insight is the fact that each connected component fulfills the constraint, i.e. if we treated each connected component as a cluster, the resulting solution would be feasible.

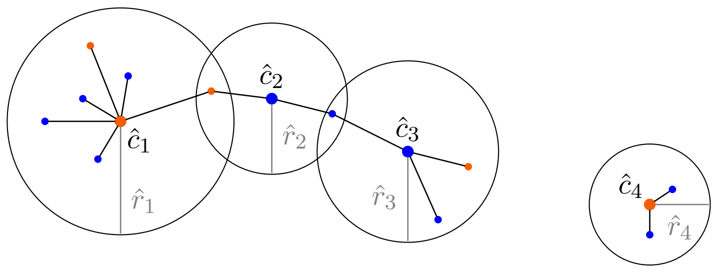


Figure II.3.2.: An instance of a k-min-sum-radii problem with exact fairness constraint with two colors and a blue:orange ratio of 2:1. The larger dots indicate centers and the gray lines indicate the radii returned by Alg. 7. The black circles show the induced balls $B(\hat{c}_i, \hat{r}_i)$. The black lines between points represent the edges of the induced access graph. Note that the balls themselves are not necessarily fair, but every connected component is.

Lemma 55. *Assume Algorithm 7 guessed correctly in each iteration and terminates with centers $\hat{\mathcal{C}} = \{\hat{c}_1, \dots, \hat{c}_k\}$ and radii $\hat{r}_1, \dots, \hat{r}_k$. Let $G = (V, E)$ be the corresponding access graph. Let $Z \in \mathrm{CC}(G)$ be a connected component of G. Then assigning all vertices from Z to an arbitrary point in Z yields a cluster that is feasible with respect to the given mergeable constraint.*

Proof. Let $V(Z)$ denote the set of vertices of Z. We will show that $V(Z)$ consists solely of ℓ optimal clusters that all lie entirely in $V(Z)$ for some $\ell \geq 1$. As optimal clusters fulfill the mergeable constraint, the union of these also fulfills the mergeable constraint.

Every point $p \in V(Z)$ lies in some optimal cluster. Hence, there exists at least one optimal cluster that intersects $V(Z)$. We want to conclude that such a cluster already is completely contained in $V(Z)$. Assume for a contradiction that there exists an optimal cluster C^* such that $C^* \cap V(Z) \neq \emptyset$ and $C^* \not\subseteq V(Z)$. By Lemma 52, there exists a ball $B(\hat{c}, \hat{r})$ such that $C^* \subseteq B(\hat{c}, \hat{r})$. Let $v \in C^* \setminus V(Z)$. Then $d(v, \hat{c}) \leq \hat{r}$ and therefore $\hat{c}$ must be part of the connected component Z, a contradiction. □

For every connected component $Z \in \mathrm{CC}(G)$, we pick one of the centers $\hat{c} \in \hat{\mathcal{C}} \cap V(Z)$ that lie inside the connected component and assign all points in $V(Z)$ to $\hat{c}$. This way, we get a solution that contains one cluster per connected component. To ensure that the radius of the resulting cluster is not too large in comparison to the sum of the radii of the original balls, we choose $\hat{c}$ with the largest corresponding $\hat{r}$ as the final center.

Lemma 56. *Let $\hat{\mathcal{C}} = \{\hat{c}_1, \dots, \hat{c}_k\}$, $\hat{r}_1, \dots, \hat{r}_k$ and $G = (V, E)$ as in Lemma 55. For every connected component $Z \in CC(G)$, let $\hat{c}^Z \in \hat{\mathcal{C}} \cap V(Z)$ be the center with maximal corresponding radius $\hat{r}^Z$ in Z. Then the solution $(\mathcal{C}, \sigma)$ with $\mathcal{C} = \{\hat{c}^Z \mid Z \in \mathrm{CC}(G)\}$*

and $\sigma\colon P \to \mathcal{C}$ with $\sigma(z) = \hat{c}^Z$ for all $z \in Z$ and for all connected components Z is a $(6 - \frac{3}{k} + \varepsilon)$-approximation for k-*Min-Sum-Radii under a mergeable constraint.*

Proof. Since each cluster in the solution corresponds to exactly one connected component of G, Lemma 55 implies that the solution fulfills the mergeable constraint. Since Algorithm 7 returns at most k centers and the merging procedure can only reduce the number of centers, $(\mathcal{C}, \sigma)$ also has at most k centers.

It remains to prove the approximation factor. Let $Z \in \mathrm{CC}(G)$ be a connected component in G. Let $v^Z \in \arg\max_{p \in Z} d(\hat{c}^Z, p^Z)$. There exists a path from $\hat{c}^Z$ to v^Z in Z. Due to the construct of G, there can only be edges where where one endpoint is in $\hat{\mathcal{C}}$ and the other in $V(G) \setminus \hat{\mathcal{C}}$, or where both are in $\hat{\mathcal{C}}$. For simplicity, we assume that a shortest path from $\hat{c}^Z$ to v^Z alternatingly visits points in $\hat{\mathcal{C}}$ and $V(G) \setminus \hat{\mathcal{C}}$. An edge between two centers on the path cannot lead to higher cost. Let $\hat{c}^Z = \hat{c}_0^Z, v^1, \hat{c}_1^Z, v^2, \hat{c}_2^Z, \dots v_Z^\ell, \hat{c}_{\ell_Z}^Z, v^Z$ be such a path, with $\ell_Z \le k$. Its length is then bounded by $\sum_{i \le \ell_Z} 2\hat{r}_i^Z - \hat{r}_{\max}^Z$, where $\hat{r}_{\max}^Z := \max_{i\colon \hat{c}_i \in Z} \hat{r}_i^Z$. The radius of a cluster with center $\hat{c}^Z$ is given by $d(v^Z, \hat{c}^Z)$. Hence, the sum of the radii of such clusters is bounded by

$$\sum_{Z \in \mathrm{CC}(G)} d(v^Z, \hat{c}^Z) \le \sum_{Z \in \mathrm{CC}(G)} \left(\sum_{i \le \ell_Z} 2\hat{r}_i^Z - \hat{r}_{\max}^Z \right) = \sum_{j=1}^{k} 2\hat{r}_j - \sum_{Z \in \mathrm{CC}(G)} \hat{r}_{\max}^Z$$

where the second equality holds because a graph's connected components are disjoint. There exists a connected component Z' such that $\hat{r}_{\max}^{Z'} = \max_{i \le k} \hat{r}_i =: \hat{r}_{\max}$. Therefore, $\sum_{Z \in \mathrm{CC}(G)} \hat{r}_{\max}^Z \ge \hat{r}_{\max}$. Further, $\hat{r}_{\max} \ge \frac{1}{k} \sum_{j=1}^{k} \hat{r}_j$. Hence,

$$\sum_{j=1}^{k} 2\hat{r}_j - \sum_{Z \in \mathrm{CC}(G)} \hat{r}_{\max}^Z \le (2 - \frac{1}{k}) \sum_{j=1}^{k} \hat{r}_j,$$

and by Lemma 54,

$$(2 - \frac{1}{k}) \sum_{j=1}^{k} \hat{r}_j \le 3(1 + \varepsilon)(2 - \frac{1}{k}) \sum_{j=1}^{k} r_j^* = (6 - \frac{3}{k})(1 + \varepsilon) \sum_{j=1}^{k} r_j^*.$$

□

Algorithm 9 finds such a solution. Now we are ready to prove our main Theorem.

Theorem 57. *For every $1 > \varepsilon > 0$, there exists a $(6 - \frac{3}{k} + \varepsilon)$-approximation algorithm for* k-*Min-Sum-Radii with mergeable constraints with runtime $O((k \log_{1+\varepsilon}(k/\varepsilon))^k \cdot \mathrm{poly}(n))$, if the corresponding constrained* k-*Center problem has a constant factor approximation algorithm.*

Proof. We invoke Algorithm 7 for all possible guesses of radius profiles and center assignments. For each of these, we compute the access graph G and invoke Algorithm 8 to obtain a solution. Note that we are approximating the optimum ε-balanced solution, since we need that assumption for our radius guessing technique. Let Opt_{bal} denote the

Algorithm 8: Assignment

```
Input: Graph $G = (V, E)$, centers $\hat{c}_1, \dots, \hat{c}_k$ and radii $\hat{r}_1, \dots, \hat{r}_k$
Output: Set of $\leq k$ centers $\mathcal{C}$, assignment $\sigma$
1  $\mathcal{C} \leftarrow \emptyset$
2  for each $Z \in CC(G)$ do
3      find a center $\hat{c} \in Z \cap \hat{\mathcal{C}}$ such that $\hat{r}$ is largest
4      $\mathcal{C} \leftarrow \mathcal{C} \cup \{\hat{c}\}$
5      for all $p \in Z$ do
6          $\sigma(p) \leftarrow \hat{c}$
7  return $\mathcal{C}$, $\sigma$
```

Algorithm 9: k-min-sum-radii with mergeable constraints

```
Input: Point set $P$, distance function $d$, $k \in \mathbb{N}$, $\varepsilon \in (0, 1)$
Output: Set of $\leq k$ centers $\mathcal{C}$, assignment $\sigma$
1  $\varepsilon' \leftarrow \frac{\varepsilon}{8}$
2  $U \leftarrow (6 + \varepsilon') \max_{x,y \in P} d(x, y)$    // upper bound on the sum of radii cost
3  $\mathscr{R} \leftarrow$ set of radius profile guesses
4  forall $(\tilde{r}_1, \dots, \tilde{r}_k) \in \mathscr{R}$ do
5      forall $a \in \{1, \dots, k\}^k$ do
6          $\{\hat{c}_1, \dots, \hat{c}_k\}, \{\hat{r}_1, \dots, \hat{r}_k\} \leftarrow$ Centers-and-radii$(P, d, k, (\tilde{r_1}, \dots, \tilde{r_k}), a)$
7          Compute the access graph $G$ based on $\{\hat{c}_1, \dots, \hat{c}_k\}$ and $\{\hat{r}_1, \dots, \hat{r}_k\}$
8          $(\mathcal{C}, \sigma) \leftarrow$ Assignment$(G,\ d,\ \{\hat{c}_1, \dots, \hat{c}_k\})$
9          if $(\mathcal{C}, \sigma)$ is feasible and kMSR$(\mathcal{C}, \sigma) < U$ then
10             $(\mathcal{C}^*, \sigma^*) \leftarrow (\mathcal{C}, \sigma)$
11             $U \leftarrow$ kMSR$(\mathcal{C}, \sigma)$
12 return $\mathcal{C}^*, \sigma^*$
```

value of this solution, and Opt the value of the actual optimum. By Lemma 56, the solution that Algorithm 7 produces is feasible and has cost at most $(6 - \frac{3}{k} + \varepsilon')\text{Opt}_{\text{bal}}$, assuming that it guessed correctly. Since it iterates over all possible guesses, we can be sure that in one of the iterations we do indeed guess correctly. In the end, we return the best solution found, whose cost can therefore be upper bounded by

$$\begin{aligned}(6 - \frac{3}{k} + \varepsilon')\text{Opt}_{\text{bal}} &\leq (6 - \frac{3}{k} + \varepsilon')(1 + \varepsilon') < (6 + \varepsilon')(1 + \varepsilon') - \frac{3}{k} \\ &= \left(6 + 7\varepsilon' + (\varepsilon')^2\right) - \frac{3}{k} = \left(6 + \frac{7\,\varepsilon}{8} + \frac{\varepsilon^2}{64}\right) - \frac{3}{k} \\ &< \left(6 + \frac{57\,\varepsilon}{64}\right) - \frac{3}{k} < 6 + \varepsilon - \frac{3}{k}.\end{aligned}$$

To be able to guess a radius profile, we first need to compute an approximate solution for constrained k-Center, which can be done in polynomial time. By Corollary 39, we can then construct the set $\mathscr{R}$ in Line 3 of Algorithm 9 in time $O(\log_{1+\varepsilon}(k/\varepsilon)^k)$. The outer for-loop in Line 4 then goes through $|\mathscr{R}|$ iterations, which is $O(\log_{1+\varepsilon}(k/\varepsilon)^k)$. The inner for-loop in Line 5 goes through k^k iterations. So in total, lines 6-11 are invoked $O\left((k\log_{1+\varepsilon}(k/\varepsilon))^k\right)$ times. The runtime of one call to Centers-and-radii is dominated by the runtime of the calls to Farthest-first-traversal-completion. This can be implemented to run in $O(kn)$, as shown in Lemma 49. So we can bound the runtime of Centers-and-radii by $O(k^2n)$. The construction of the access graph can be performed in $O(kn)$, as can one call to Assignment. The feasibility of a solution can be checked in $O(n)$. Thus, we obtain an overall running time of $O\left((k\log(k/\varepsilon))^k \cdot \mathrm{poly}(n)\right)$. $\square$

II.3.5. Improved Merging to Obtain a $(4+\varepsilon)$-Approximation

In the following, we show how to improve the approximation ratio of our algorithm to $(4+\varepsilon)$ by utilizing an improved merging procedure presented by Bandyapadhyay and Chen [18]. To do this, we first need to introduce some more notation. Let $\hat{\mathcal{B}} = \{\mathrm{B}(\hat{r}_1,\hat{c}_1),\ldots,\mathrm{B}(\hat{r}_k,\hat{c}_k)\}$ be the set of balls returned by Algorithm 7. We define the *cluster graph* $G_{\hat{\mathcal{B}}} = (V,E)$ of $\hat{\mathcal{B}}$, where $V = \{\hat{c}_i \mid \mathrm{B}(\hat{c}_i,\hat{r}_i) \in \hat{\mathcal{B}}\}$ and $E = \{\{v_i,v_j\} \mid \exists p \in P : p \in \mathrm{B}(\hat{c}_i,\hat{r}_i) \cap \mathrm{B}(\hat{c}_j,\hat{r}_j), i \neq j\}$. In other words, this is just a simplified version of the access graph defined in the previous section: we have one vertex for each ball and an edge between two vertices if and only if there exists a point which lies in *both* corresponding balls. Figure II.3.3 illustrates this construction. Let $Z = (V(Z),E(Z)) \in CC(G_{\hat{\mathcal{B}}})$ be a connected component of $G_{\hat{\mathcal{B}}}$. We say that a ball $\mathrm{B}(\hat{c},\hat{r})$ is *involved* in Z if $\hat{c} \in V(Z)$. Let $\hat{\mathcal{C}}_Z$ denote the set of centers involved in Z and $\hat{\mathcal{B}}_Z$ the set of balls involved in Z. Then we let P_Z denote the union of all points that lie within balls that are involved in Z, i.e. $P_Z = \{p \in P \mid \exists B \in \hat{\mathcal{B}}_Z : p \in B\}$. We can observe that the statement in Lemma 55 stays true for this slightly modified graph.

Observation 58. *If Algorithm 7 guessed correctly in each iteration, then for every connected component Z of $G_{\hat{\mathcal{B}}}$, the set P_Z fulfills the mergeable constraint.*

It is still our goal to merge all balls involved in Z, i.e., replacing them by one large ball covering all of P_Z. The key difference of the approach in [18] is that we now try *all points* (not just centers) in P_Z and choose the one for which the radius needed to cover all of P_Z is minimal. Before moving on to the main lemma, we need some more notation. For a given connected component $Z \in CC(G_{\hat{\mathcal{B}}})$, we fix one arbitrary spanning tree T_Z. For any two leaves of T_Z there exists a unique path between them in T_Z. We call such a path a *leaf-to-leaf path*. Let $\pi = (\{\hat{c}_{\pi_1},\ldots,\hat{c}_{\pi_\ell}\},\{\{\hat{c}_{\pi_1},\hat{c}_{\pi_2}\},\ldots,\{\hat{c}_{\pi_{\ell-1}},\hat{c}_{\pi_\ell}\}\})$ be a leaf-to-leaf path in T_Z (i.e. $\hat{c}_{\pi_1}$ and $\hat{c}_{\pi_\ell}$ are leaves) that maximizes the sum of the radii of the corresponding balls, i.e. $\sum_{i=1}^{\ell}\hat{r}_{\pi_i}$. For an illustration of these definitions, see Figure II.3.4.

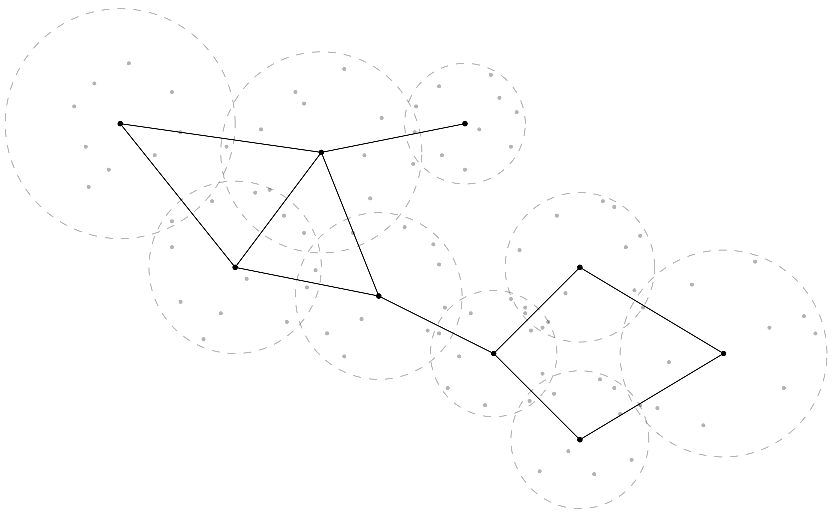


Figure II.3.3.: A set of overlapping balls and the corresponding connected component of the cluster graph. Every vertex corresponds to a ball and we have an edge iff the intersection of the corresponding balls contains at least one point.

Now consider the graph $T_Z - \pi$ that arises from T_Z by deleting all vertices and edges of π. $T_Z - \pi$ itself consists of trees, each of which is connected to a unique vertex $\hat{c}_{\pi_i}$ of π in T_Z. We say that such a tree is *attached* to $\hat{c}_{\pi_i}$. For an arbitrary vertex $\hat{c}_{\pi_i} \in \pi$, let $\Gamma(\hat{c}_{\pi_i}) = \{\hat{c}_{\pi_i}\} \cup \{\hat{c} \in V(T) \mid T \in CC(T_Z - \pi) \text{ and } T \text{ is attached to } \hat{c}_{\pi_i}\}$ be the union of vertices of such trees that are attached to $\hat{c}_{\pi_i}$, plus the vertex $\hat{c}_{\pi_i}$ itself. Finally, imagine the following setting. We can always draw T_Z in such a way that the path π runs from left to right. In the following proof, we want to move along π "from left to right" and keep track of all points and radii that are to the left and to the right of our current position. To this end, we define the following center sets for $i \in \{1, \dots, \ell\}$:

$$\mathcal{C}_L(i) = \bigcup_{j=1}^{i-1} \Gamma(\hat{c}_{\pi_j}) \qquad \mathcal{C}_M(i) = \Gamma(\hat{c}_{\pi_i}) \qquad \mathcal{C}_R(i) = \bigcup_{j=i+1}^{\ell} \Gamma(\hat{c}_{\pi_j})$$

We can think of $\mathcal{C}_L(i)$ as all centers "to the left" of $\hat{c}_{\pi_i}$ when traversing T_Z along π, $\mathcal{C}_R(i)$ as all centers "to the right" $\hat{c}_{\pi_i}$, and $\mathcal{C}_M(i)$ as the centers in trees that are attached to $\hat{c}_{\pi_i}$ (i.e. the centers "in the middle"). Note that for all $i \in [\ell]$ we have $\mathcal{C}_L(i) \cup \mathcal{C}_M(i) \cup \mathcal{C}_R(i) = V(Z)$. Analogously, we define the set of all *points* to the left and

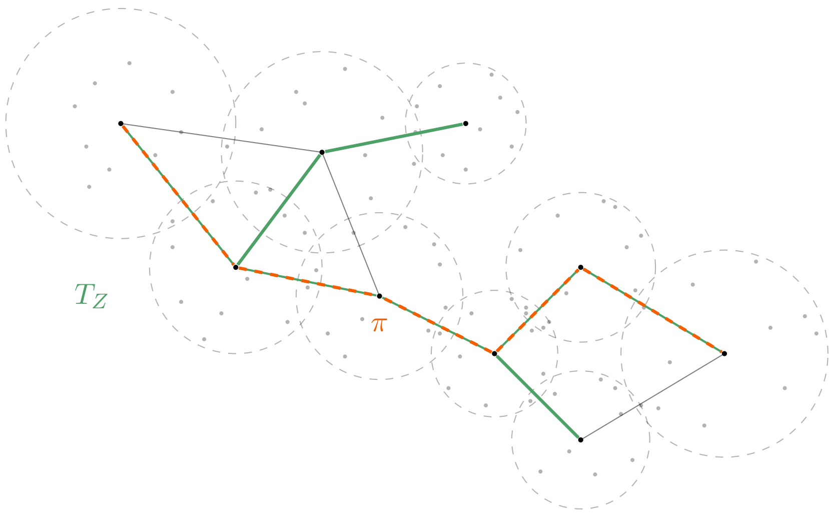


Figure II.3.4.: The edges of a spanning tree T are shown in green and the dashed orange edges indicate a leaf-to-leaf path π that maximizes the sum of radii of balls along the path.

right of $\hat{c}_{\pi_i}$ and in the middle as

$$P_L(i) = \bigcup_{j=1}^{i-1} \left(\bigcup_{\hat{c}_h \in \Gamma(\hat{c}_j)} \mathrm{B}(\hat{c}_h, \hat{r}_h) \right)$$
$$P_M(i) = \bigcup_{\hat{c}_j \in \Gamma(\hat{c}_{\pi_i})} \mathrm{B}(\hat{c}_j, \hat{r}_j)$$
$$P_R(i) = \bigcup_{j=i+1}^{\ell} \left(\bigcup_{\hat{c}_h \in \Gamma(\hat{c}_j)} \mathrm{B}(\hat{c}_h, \hat{r}_h) \right).$$

For a set of centers $\mathcal{C}$ we define the sum of corresponding radii as

$$\mathcal{R}(\mathcal{C}) = \sum_{c \in \mathcal{C}} r(c).$$

Now we can prove the following lemma showing that there always exists a center choice within Z for which the cost increases by a factor of at most $\frac{4}{3}$.

Lemma 59 ([18]). *Let $Z \in CC(G_{\hat{\mathcal{B}}})$ and let $\mathcal{R}_Z = \sum_{\mathrm{B}(c,r) \in \hat{\mathcal{B}}_Z} r$, i.e., the sum of radii of all balls involved in Z. Then there always exists a point $p \in P_Z$ such that $\max_{q \in P_Z} d(p,q) \leq \frac{4}{3}\mathcal{R}_Z$.*

Proof. We consider two cases. If there exists a ball $\mathrm{B}(c,r) \in \hat{\mathcal{B}}_Z$ with $r \geq \frac{2}{3}\mathcal{R}_Z$, we choose c as the center. Let $p \in P_Z$ be arbitrary. We can bound $d(c,p) \leq r + 2 \cdot (\mathcal{R}_Z - r) = 2\mathcal{R}_Z - r \leq \frac{4}{3}\mathcal{R}_Z$, where the last inequality comes from the assumption that $r \geq \frac{2}{3}\mathcal{R}_Z$.

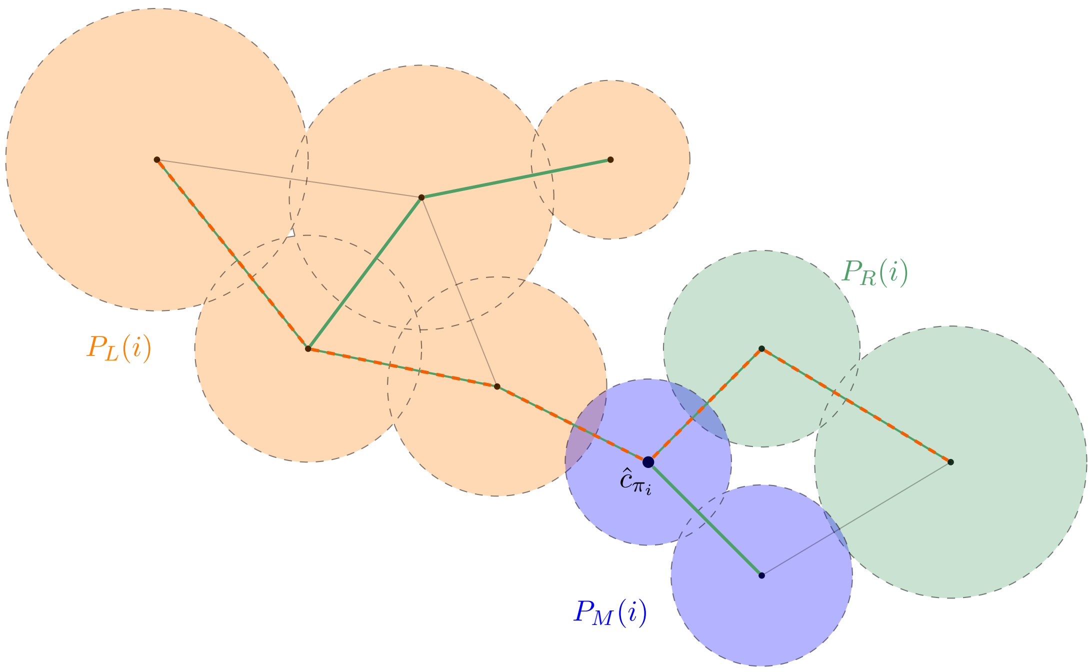


Figure II.3.5.: Illustration of $P_L(i), P_M(i)$ and $P_R(i)$. Note how not only the balls along π are included, but also all balls that are attached to the vertices in π.

If, on the other hand, for all balls $\mathrm{B}(c,r) \in \hat{\mathcal{B}}_Z$ we have $r < \frac{2}{3}\mathcal{R}_Z$, the argument becomes a bit more complicated. Let T_Z be a spanning tree of Z and π a leaf-to-leaf path in T_Z that maximizes the sum of radii. Let i be the smallest index such that $\mathcal{R}(\mathcal{C}_L(i) \cup \mathcal{C}_M(i)) \geq \frac{2}{3}\mathcal{R}_Z$. We make another case distinction based on the sum of radii “to the left“ of $\hat{c}_{\pi_i}$.

- **Case 1:** $\mathcal{R}(\mathcal{C}_L(i)) \geq \frac{1}{3}\mathcal{R}_Z$.
  In this case, we choose an arbitrary point $p \in B(\hat{c}_{\pi_{i-1}}, \hat{r}_{\pi_{i-1}}) \cap B(\hat{c}_{\pi_i}, \hat{r}_{\pi_i})$ as center. Since all balls have radius smaller than $\frac{2}{3}\mathcal{R}_Z$, we must have $i \geq 2$, and therefore $i-1$ is well-defined.
  Recall that, for any i, we have $P_L(i) \cup P_M(i) \cup P_R(i) = P_Z$. Therefore, we can bound the distance from points in $P_L(i) \cup P_M(i)$ and $P_R(i)$ to that new center separately. Consider an arbitrary $p_L \in P_L(i) \cup P_M(i)$. By choice of i, we must have $\mathcal{R}\left(\mathcal{C}_L(i)\right) < \frac{2}{3}\mathcal{R}_Z$. Therefore, we can bound

$$d(p, p_L) \leq 2\mathcal{R}\left(\mathcal{C}_L(i)\right) \leq \frac{4}{3}\mathcal{R}_Z.$$

  To bound the cost for points in $P_R(i)$, let $p_R \in P_R(i)$ be arbitrary. Since we are in the case where $\mathcal{R}(\mathcal{C}_L(i)) \geq \frac{1}{3}\mathcal{R}_Z$, we know that $\mathcal{R}(\mathcal{C}_R(i)) \leq \frac{2}{3}\mathcal{R}_Z$. Therefore, we can again conclude that

$$d(p, p_R) \le 2\mathcal{R}(\mathcal{C}_R(i)) < \frac{4}{3}\mathcal{R}_Z.$$

- **Case 2:** $\mathcal{R}(\mathcal{C}_L(i)) < \frac{1}{3}\mathcal{R}_Z$.
In this case, we choose $\hat{c}_{\pi_i}$ as the center. We bound the cost for $P_L(i)$, $P_M(i)$ and $P_R(i)$ separately.
So let $p_L \in P_L(i)$. Since we are in the case where $\mathcal{R}(\mathcal{C}_L(i)) < 1/3\mathcal{R}_Z$ and all balls have radius less than $2/3\mathcal{R}_Z$, we have

$$\begin{aligned} d(p_L, \hat{c}_{\pi_i}) &\le 2\mathcal{R}(\mathcal{C}_L(i)) + \hat{r}_{\pi_i} \\ &< \frac{2}{3}\mathcal{R}_Z + \frac{2}{3}\mathcal{R}_Z \\ &= \frac{4}{3}\mathcal{R}_Z. \end{aligned}$$

We can argue similarly for $P_R(i)$. Observe that $\mathcal{R}(\mathcal{C}_L(i) \cup \mathcal{C}_M(i)) \ge 2/3\mathcal{R}_Z$ also implies $\mathcal{R}(\mathcal{C}_R(i)) \le 1/3\mathcal{R}_Z$. Let $p_R \in P_R(i)$. We have

$$\begin{aligned} d(p_R, \hat{c}_{\pi_i}) &\le 2\mathcal{R}(\mathcal{C}_R(i)) + \hat{r}_{\pi_i} \\ &\le \frac{2}{3}\mathcal{R}_Z + \frac{2}{3}\mathcal{R}_Z \\ &= \frac{4}{3}\mathcal{R}_Z. \end{aligned}$$

Finally, consider an arbitrary $p_M \in P_M(i)$ and let $\hat{c} \in \mathcal{C}_M(i)$ be the center of a ball $B(\hat{c}, \hat{r})$ that contains p_M. Let π' be a path from some leaf in $\Gamma(\hat{c}_{\pi_i})$ to $\hat{c}_{\pi_i}$ that contains $\hat{c}$. Such a path has to exist as $\hat{c} \in \mathcal{C}_M(i) = \Gamma(\hat{c}_{\pi_i})$. Let $\pi' - \hat{c}_{\pi_i}$ be the path that results from π' by removing $\hat{c}_{\pi_i}$. Observe that we can bound

$$d(p_M, \hat{c}_{\pi_i}) \le \hat{r}_{\pi_i} + 2\mathcal{R}(V(\pi' - \hat{c}_{\pi_i})),$$

so if we get an upper bound on the right hand side of this inequality we are done. Since $\mathcal{R}(\mathcal{C}_L(i)) + \mathcal{R}(\mathcal{C}_M(i)) + \mathcal{R}(\mathcal{C}_R(i)) = \mathcal{R}_Z$ and $V(\pi')$ is a subset of $\mathcal{C}_M(i)$, we have

$$\begin{aligned} \mathcal{R}_Z &\ge \mathcal{R}(\mathcal{C}_L(i)) + \mathcal{R}(V(\pi')) + \mathcal{R}(\mathcal{C}_R(i)) \\ &= \mathcal{R}(\mathcal{C}_L(i)) + \mathcal{R}(V(\pi' - \hat{c}_{\pi_i})) + \hat{r}_{\pi_i} + \mathcal{R}(\mathcal{C}_R(i)). \end{aligned}$$

At the same time, we have

$$\begin{aligned} \mathcal{R}(V(\pi')) = \mathcal{R}(V(\pi' - \hat{c}_{\pi_i})) + \hat{r}_{\pi_i} &\le \mathcal{R}(V(\pi)) \\ &\le \mathcal{R}(\mathcal{C}_L(i)) + \hat{r}_{\pi_i} + \mathcal{R}(\mathcal{C}_R(i)), \end{aligned}$$

where the first inequality comes from the fact that π is defined as the leaf-to-leaf path with the largest sum of radii. From this, we can conclude

$$\mathcal{R}(V(\pi' - \hat{c}_{\pi_i})) \leq \mathcal{R}(\mathcal{C}_L(i)) + \mathcal{R}(\mathcal{C}_R(i)).$$

Combining all of this, we get

$$\mathcal{R}_Z \geq \hat{r}_{\pi_i} + 2\mathcal{R}(V(\pi' - \hat{c}_{\pi_i})),$$

giving us the desired upper bound and proving the claim.

□

We now get the following improved approximation guarantee.

Theorem 60. *For every $\varepsilon > 0$, there exists a $(4+\varepsilon)$-approximation algorithm for* k-*Min-Sum-Radii with mergeable constraints with runtime $O((k \log_{1+\varepsilon}(k/\varepsilon))^k \cdot \text{poly}(n))$, if the corresponding constrained* k-*Center problem has a constant factor approximation algorithm.*

Proof. We can argue just as in the proof of Theorem 57. The radius guessing and computation of balls is unchanged and therefore takes up exactly the same amount of time. The merging procedure can again be implemented in polynomial time in n, as in the worst case we have to try out all n points as possible centers, computing distances to at most $n-1$ other points, leaving us with a total runtime of $\mathcal{O}(n^2)$.

Since the set of balls returned by Algorithm 7 approximate the cost within a factor of $(3+\varepsilon)$, and by Lemma 59 the merging increases cost by a factor of at most $\frac{4}{3}$, we get an overall approximation ratio of $(3+\varepsilon) \cdot \frac{4}{3} = (4 + \frac{4}{3}\varepsilon)$. Thus, calling the algorithm with $\varepsilon' = \frac{3}{4}\varepsilon$ gives a $(4+\varepsilon)$-approximation. □

II.3.6. Improved Approximation Ratios for Special Cases

While we showed that the algorithm works for *all* mergeable constraints in a black box manner, we can get improved approximation guarantees for some special cases by slightly adjusting it. First, we show how we obtain a $(3+\varepsilon)$-approximation for a restricted fairness setting, before showing the same factor for uniform lower bounds on cluster sizes.

II.3.6.1. Fairness with Two Colors and Equal Proportions

We consider the fairness setting with two colors and equal proportions, i.e. when $\Gamma = \{1,2\}$ and $|\Gamma_1(P)|/|P| = |\Gamma_2(P)|/|P| = \frac{1}{2}$. In this case, we can find a fair assignment such that none of the radii $\hat{r}$ has to be enlarged. The idea is that we first compute a fairlet decomposition and then assign the resulting fairlets to a common center. We want to partition the set P into pairs consisting of two points p_1, p_2 where $p_1 \in \Gamma_1(P)$ and $p_2 \in \Gamma_2(P)$. This is equivalent to finding a perfect matching (cf. Definition 19) of size $\frac{n}{2}$ between $\Gamma_1(P)$ and $\Gamma_2(P)$.

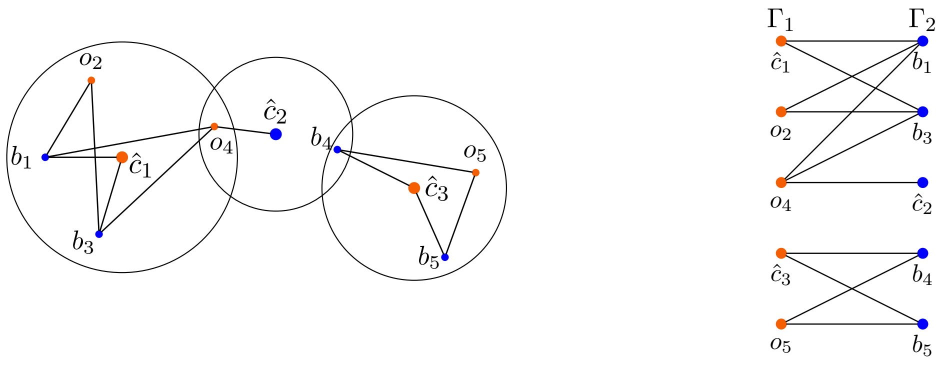


Figure II.3.6.: A fair k-Min-Sum-Radii instance with two colors and equal proportions. Left: Output of Alg. 7 as balls $B(\hat{c}_i, \hat{r}_i)$ for $i = 1, 2, 3$ and the graph edges between any fair pair of points that have access to the same center $\hat{c}$. Right: The corresponding graph in which we compute the matching.

Let $\hat{\mathcal{C}} = \{\hat{c}_1, \dots, \hat{c}_k\}$ be the centers and $\hat{R} = \{\hat{r}_1, \dots, \hat{r}_k\}$ be the radii returned by Algorithm 7. We construct a bipartite graph $G = (V, E)$, where

$$V(G) = \Gamma_1(P) \cup \Gamma_2(P)$$
$$E(G) = \{\{p, q\} \mid \exists i \in [k] : d(\hat{c}_i, p) \leq \hat{r}_i \ \wedge \ d(\hat{c}_i, q) \leq \hat{r}_i\}$$

In other words, we have a bipartite graph where one bipartition contains all points of color 1 and the other all points of color 2. There is an edge between two points of different colors if and only if there exists a ball that contains *both* of them. See Figure II.3.6 for an illustration. It remains to show that there always exists a perfect matching in that graph if Algorithm 7 guessed correctly.

Lemma 61. *Assume that Algorithm 7 guessed correctly. Then there always exists a perfect matching in the graph G constructed as above.*

Proof. Since any optimal solution must be fair, there exists a matching

$$M^* = \Big\{\{p_1, q_1\}, \dots, \{p_{n/2}, q_{n/2}\}\Big\}$$

such that, for $i = 1, \dots, n/2$, both p_i and q_i are in the same optimal solution cluster. By Lemma 52, for each optimum solution cluster C^*, there exist $\hat{c}$ and $\hat{r}$ in the output of Algorithm 7, such that $C^* \subseteq B(\hat{c}, \hat{r})$. This means that, for any $\{p_i, q_i\} \in M^*$, there will be an edge $\{p_i, q_i\}$ in G. These edges constitute a perfect matching in G. □

Each fairlet in this matching is now assigned as a whole. More precisely, we assign both points from the fairlet to one of the centers to which both points have access. This obviously does not increase the radii.

Observation 62. *Let $F = \{p_1, p_2\}$ be a fairlet constructed as described above, let $\hat{c}$ be the center to which it gets assigned and $\hat{r}$ the corresponding radius. Then*

$$\max_{i=1,2} d(p_i, \hat{c}) \leq \hat{r}.$$

Together with Lemma 54, this implies the following theorem.

Theorem 63. *For every $\varepsilon > 0$, there exists an FPT $(3+\varepsilon)$-approximation algorithm for* Fair k-Min-Sum-Radii *in the special case where $\Gamma = \{1, 2\}$ and $|\Gamma_1(P)| = |\Gamma_2(P)| = n/2$.*

Proof. We run Algorithm 7 for all $\mathcal{O}(\log_{1+\varepsilon}(k/\varepsilon)^k)$ radius profile candidates and all k^k tuples to obtain balls $B(\hat{c}, \hat{r})$. Since one call to Algorithm 7 takes polynomial time, all of these calls take $f(k) \cdot \text{poly}(n)$ time. We then try to compute a partitioning of P into fairlets as described above. If the algorithm guessed correctly, a perfect matching will be found. The bipartite graph has n vertices and $m \in \mathcal{O}(n^2)$ edges. Finding a perfect matching in a bipartite graph can be done in $\mathcal{O}(nm)$ (see, e.g., [75]). We assign each fairlet to a center to which both points of the pair have access, which can also be done in polynomial time.

By Lemma 54, $\sum_{i=1}^{k} \hat{r}_i \leq 3(1+\varepsilon) \sum_{i=1}^{k} r_i^*$. As noted in Observation 62, assigning fair pairs does not increase the cost, which concludes the proof. □

II.3.6.2. Uniform Lower Bounds

As introduced in Section I.3.1, in k-Min-Sum-Radii with uniform lower bounds, we have an additional input number $L \in \mathbb{N}$, and every cluster in the solution needs to contain at least L points. Let $\hat{\mathcal{C}} = \{\hat{c}_1, \ldots, \hat{c}_k\}$ be the centers and $\hat{R} = \{\hat{r}_1, \ldots, \hat{r}_k\}$ be the radii returned by Algorithm 7. In this case, we set up a flow network (G, u, s, t) as follows. The graph G is again bipartite, with the centers $\hat{\mathcal{C}}$ on the left and the points P on the right. There is a source s and an edge $(s, \hat{c})$ for every $\hat{c} \in \hat{\mathcal{C}}$. The capacity of these edges is set to the lower bound L. We also have a sink t and an edge (p, t) for every $p \in P$, where all of these edges have capacity 1. Finally, every $\hat{c}$ is connected to all $p \in P$ with $d(\hat{c}, p) \leq \hat{r}$, also with capacity 1. More formally, we have

$$V(G) = \hat{\mathcal{C}} \,\dot\cup\, P \cup \{s, t\}$$
$$E(G) = \{(s, \hat{c}) \mid \hat{c} \in \hat{\mathcal{C}}\} \cup \{(p, t) \mid p \in P\} \cup \{(\hat{c}, p) \mid \hat{c} \in \hat{\mathcal{C}}, p \in P : d(\hat{c}, p) \leq \hat{r}\}$$

and $u : E(G) \to \mathbb{R}_{\geq 0}$ with

$$u(e) = \begin{cases} L & \text{if } e \in \delta^+(s), \\ 1 & \text{otherwise.} \end{cases}$$

Lemma 64. *Let $N = (G, u, s, t)$ be the network constructed from $\hat{\mathcal{C}}$ and $\hat{R}$ as described above. There exists an integral maximum flow f with $\text{val}(f) = k \cdot L$, if Algorithm 7* *guessed correctly.*

Proof. We first show that the maximum flow value in N is at least $k \cdot L$ by proving the existence of a flow of that value. Let $C_1^*, \dots, C_k^*$ be the clusters of the optimal solution we want to approximate. Let $\hat{\mathcal{B}}_{>0}$ be the set of balls with non-zero radius induced by the output of Algorithm 7. By Corollary 53, there exists a mapping $\varphi : \{C_1^*, \dots, C_k^*\} \to \hat{\mathcal{B}}_{>0}$, such that $C_i^* \subseteq \varphi(C_i^*)$ for all $i \in [k]$. We construct f as follows. For an arbitrary $\hat{c} \in \hat{\mathcal{C}}$ with corresponding radius $\hat{r} \in \hat{R}$, let $C^* \in \varphi^{-1}(\mathrm{B}(\hat{c}, \hat{r}))$. Since $C^* \subseteq \mathrm{B}(\hat{c}, \hat{r})$, we must have an edge $(\hat{c}, p)$ for all $p \in C^*$. C^* has to satisfy the lower bound, so we must have $|C^*| \geq L$, i.e., there are at least L such edges. We choose set $f(e) = 1$ for exactly L of these edges. Additionally, we set $f(s, \hat{c}) = L$. This way, $\hat{c}$ has exactly L units of flow coming in and going out.

Note that, since each edge (p, t) has capacity 1, each p can have only one incoming edge with flow 1. But since φ is a well-defined mapping (i.e. $C^* \notin \varphi^{-1}(\mathrm{B}')$ for any other ball $\mathrm{B}' \in \hat{\mathcal{B}}_{>0})^2$, we can do the above procedure for each $\hat{c}$ independently, without having to worry that any p already has an incoming edge with flow 1. Finally, we set $f((p, t)) = 1$ for all p that have an incoming edge with flow. This way, we get a feasible flow f with $\mathrm{val}(f) = k \cdot L$.

To see that the value of a maximum flow in N is also *at most* $k \cdot L$, we observe that s has exactly k outgoing edges with capacity L. It follows that any maximum flow must have value $k \cdot L$, and since all capacities are integral, there always has to exist an integral maximum flow [75]. □

We now show that any integral maximum flow in this network has another useful property.

Lemma 65. *Let f be an integral maximum flow in N. Then $|\{e \in \delta^+(\hat{c}) : f(e) = 1\}| = L$ for all $\hat{c} \in \hat{\mathcal{C}}$.*

Proof. By Lemma 64, $\mathrm{val}(f) = k \cdot L$. This implies that $f((s, \hat{c})) = L$ for all $\hat{c} \in \hat{\mathcal{C}}$. Let $\hat{c} \in \hat{\mathcal{C}}$. f is a feasible flow, so we must have $\sum_{e \in \delta^+(e)} f(e) = L$. Since $u(e) = 1$ for all $e \in \delta^+(e)$ and f is integral, we can conclude that $|\{e \in \delta^+(\hat{c}) : f(e) = 1\}| = L$. □

We now have everything we need to compute a feasible solution from the output of Algorithm 7 without increasing the cost.

Theorem 66. *For every $\varepsilon > 0$, there exists an FPT $(3 + \varepsilon)$-approximation algorithm for* k-Min-Sum-Radii *with uniform lower bounds.*

Proof. We again run Algorithm 7 for all radius profile candidates and all tuples to obtain $\hat{\mathcal{C}}$ and $\hat{R}$. From that, we can construct the network $N = (G, u, s, t)$ as described above and compute an integral maximum flow f. If the algorithm guessed correctly, by Lemma 64, $\mathrm{val}(f) = k \cdot L$, and by Lemma 65 we also have $|\{e \in \delta^+(\hat{c}) : f(e) = 1\}| = L$ for all $\hat{c} \in \hat{\mathcal{C}}$. We now compute an assignment $\sigma : P \to \hat{\mathcal{C}}$ from f in the following way. If $f(\hat{c}, p) = 1$,

[2]It should be noted that, of course, C^* might be contained in more than one ball from $\hat{\mathcal{B}}_{>0}$, but under φ it is only assigned to exactly one.

set $\sigma(p) = \hat{c}$. Note that this ensures that $|\sigma^{-1}(\hat{c})| \geq L$ for all $\hat{c} \in \hat{\mathcal{C}}$. If all $p \in P$ are already assigned after this step, we are done. If not, let p be a point without any assigned center. By Lemma 52, there exists a center $\hat{c} \in \hat{\mathcal{C}}$ with corresponding radius $\hat{r}$, such that $p \in \mathrm{B}(\hat{c}, \hat{r})$. We assign p to an arbitrary such $\hat{c}$. Thus, we end up with an assignment that fulfills the lower bound and that only assigns points within the balls computed by Algorithm 7. By Lemma 54, the radii of these balls yield a $(3+\varepsilon)$-approximation.

The calls to Algorithm 7 again take $f(k) \cdot \mathrm{poly}(n)$ time. The graph of the flow network has $\mathcal{O}(n)$ vertices and at most $\mathcal{O}(n^2)$ edges. The computation of f can be done in polynomial time in the size of that graph (see, e.g., [75]). The subsequent computation of σ can be done in $\mathcal{O}(kn)$ time. This proves the claim. $\square$

Part III.

k-Means Clustering

III.1. Preliminaries

Arguably one of the most famous clustering objective functions is k-`Means`. As already mentioned in the introduction, it judges the quality of a clustering by the sum of squared distances of points to their centers. In many practical applications, like customer segmentation, image processing (cf. the example of image compression in Part I), document clustering or logistics, this objective has proven useful. In the following, we give formal problem definitions and mention some properties of this objective, before turning to previous work and milestone results. Then we present a series of algorithms that are either of historical importance or of special importance to our own work. Finally, we present our algorithm `FLS++`, along with extensive experimental results. We begin by formally stating the Metric k-Means problem.

Metric k-Means Problem

Input: A set P of n points, a metric distance function $\mathrm{dist} : P \times P \to \mathbb{R}_{\geq 0}$, and a number $k \in \mathbb{N}, k \leq n$

Output: A center set $\mathcal{C} = \{c_1, \dots, c_k\} \subseteq P$ and an assignment $\sigma : P \to \mathcal{C}$ minimizing $\sum_{p \in P} \mathrm{dist}(p, \sigma(p))^2$

Note that, in the absence of additional constraints, for any given set of centers it is always feasible and optimal to assign each point to its closest center. Therefore, we mostly talk about solutions for k-means only in terms of the center set $\mathcal{C}$, with σ assumed to be the closest center assignment. In this case, we change notation and write

$$\texttt{kMeans}(\mathcal{C}) = \sum_{p \in P} \mathrm{dist}(p, c(p))^2,$$

where $c(p) = \arg\min_{c \in \mathcal{C}} \mathrm{dist}(p, c)$. In the following, we focus on the Euclidean setting where $P \subset \mathbb{R}^d$ for some $d \in \mathbb{N}$, and $\mathrm{dist}(p, q) = ||p - q||$ is the Euclidean distance. Additionally, centers can then be chosen freely from $\mathbb{R}^d$. For point set $P \subset \mathbb{R}^d$, we define its *centroid* as $\mu(P) = \frac{1}{|P|} \sum_{p \in P} p$. With that, we formally define the Euclidean k-Means problem as follows.

Euclidean k-Means Problem

Input: A set $P \subset \mathbb{R}^d$ of n points, a number $k \in \mathbb{N}, k \leq n$

Output: A center set $\mathcal{C} = \{c_1, \dots, c_k\} \subset \mathbb{R}^d$ minimizing $\sum_{p \in P} ||p - c(p)||^2$

One nice property of Euclidean k-Means is the fact that, if a partition (i.e., the clusters themselves) is already given, finding an optimal center for each cluster is easy. To show this, we first prove the following statement.

Lemma 67 ([88])**.** *Let $P \subset \mathbb{R}^d$ be an arbitrary set of points and let $\mu(P)$ be the centroid of P. Then for any point $c \in \mathbb{R}^d$ it holds that*

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

Proof. Note that for any two points $x, y \in \mathbb{R}^d$ it holds that

$$||x+y||^2 = \langle x,x\rangle + 2\cdot\langle x,y\rangle + \langle y,y\rangle = ||x||^2 + 2\cdot\langle x,y\rangle + ||y||^2.$$

Then we can write

$$\begin{aligned}\sum_{p\in P} ||p-c||^2 &= \sum_{p\in P} ||p-\mu(P)+\mu(P)-c||^2\\ &= \sum_{p\in P}\Big(||p-\mu(P)||^2 + 2(p-\mu(P))^T\cdot(\mu(P)-c) + ||\mu(P)-c||^2\Big)\\ &= \sum_{p\in P} ||p-\mu(P)||^2 + 2(\mu(P)-c)^T\cdot\sum_{p\in P}(p-\mu(P)) + |P|\cdot||\mu(P)-c||^2.\end{aligned} \tag{1}$$

Now since $\mu(P) = 1/|P| \cdot \sum_{p\in P} p$, we have

$$\begin{aligned}\sum_{p\in P}(p-\mu(P)) &= \sum_{p\in P}\left(p - \frac{1}{|P|}\sum_{p\in P} p\right)\\ &= \sum_{p\in P} p - |P|\cdot\frac{1}{|P|}\sum_{p\in P} p\\ &= 0.\end{aligned}$$

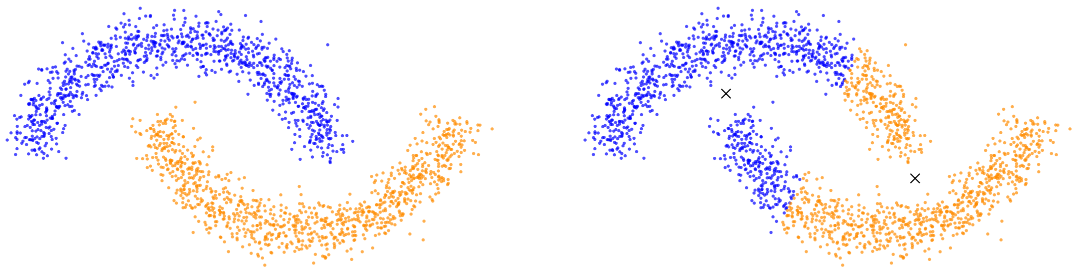
Inserting this into (1) yields

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

□

An immediate consequence of this Lemma is the following.

Corollary 68. *For any set $P \subset \mathbb{R}^d$ of points, the optimal 1-center solution is its centroid $\mu(P)$.*


(a) The desired partition.

(b) The partition induced by the optimum k-`Means` solution.

Figure III.1.1.: An instance where minimizing the k-`Means` objective will not lead to good results. The desired partitioning is obvious, but the optimum k-`Means` solution for $k = 2$ looks very different.

Proof. This follows directly from Lemma 67 by substituting $c = \mu(P)$. Then the second term $|P| \cdot ||\mu(P) - c||^2$ is 0, and therefore the whole expression is minimized. $\square$

This fact justifies the name of the objective, as the centroid of P can also be thought of as the *mean* of the points in P. Additionally, it plays a central role in the design of an important algorithm for k-Means (cf. Section III.1.2).

k-`Means` naturally favors "spherical" clusters that are well separated, whereas more complex shapes can be impossible to retrieve. Especially non-convex or elongated cluster shapes happen to be problematic for the objective. As an example, consider Figure III.1.1. To the human eye, the two clusters are easy to identify. But k-`Means` is not able to capture these non-convex arcs and produces an unintuitive solution[1]. This can be partially explained by the connection of k-`Means` to maximum-likelihood estimation (MLE) for Gaussian mixture models. Intuitively, MLE aims at finding the parameters of a statistical model that "best explain" the observed data under that model. Thus, MLE for Gaussian mixture models makes the implicit assumption that the data was "produced" by multiple multivariate Gaussian distributions, and aims at finding the parameters of these distributions that maximize the likelihood of the dataset. If we additionally assume that all components of the mixture model have equal variance and no directional preference (i.e., each cluster is assumed to be spread out spherically around its center), and that every point belongs to exactly one cluster, it can be shown that maximizing the likelihood is the same as minimizing the k-`Means` objective (with k being the number of components of the mixture model). For a formal derivation of this fact, see, e.g., [28]. It therefore seems only natural that k-`Means` works best for point sets that come close to this idealized model. We address the question when the application of k-`Means` might be sensible again in Section III.2.3.

[1]For instances like this, single linkage, spectral clustering or density-based methods like DBSCAN lead to much better results.

III.1.1. Related Work

In this section, we give a brief overview of important milestones regarding (approximation) hardness, approximation algorithms and heuristics for the k-Means problem.

III.1.1.1. Hardness

Like most clustering problems, k-Means is NP-hard. A first proof of this fact by Drineas et al. [49] turned out to be flawed. The problem with their reduction is pointed out by Aloise et al. [8], who subsequently give a valid reduction from the sparsest cut problem. More precisely, they show that k-means is NP-hard for $k = 2$ in general dimension. Independently, Dasgupta [45] obtained the same result via a reduction from not-all-equal 3Sat.

In [82], Mahajan et al. show that, even in the planar case (i.e. $d = 2$), the problem remains NP-hard if k is not constant. While they do this by a reduction from Planar 3Sat, Vattani [91] shows it by a reduction from Exact Cover by 3-Sets. In [17] by Awasthi et al., it is shown that in the Euclidean case it is APX-hard, i.e., that there exists an $\varepsilon > 0$ such that it is NP-hard to approximate k-Means within a factor of $(1 + \varepsilon)$. In [78], Lee et al. show that this ε is at least 0.0013, i.e. it is NP-hard to approximate the problem within a factor of 1.0013. Cohen-Addad et al. [39] later improved this to 1.06.

III.1.1.2. Approximability and Algorithms

The natural question arising from these hardness results is whether there are matching upper bounds. The result in [17] shows that one cannot hope for a PTAS. A common approach in the design of approximation algorithms is therefore to consider the number of centers k as fixed, i.e. not part of the input. This is, of course, a rather limiting assumption. It has, however, become common in the literature to still refer to these algorithms as “approximation algorithms” or “approximation schemes”. Sometimes, it is additionally assumed that, in the Euclidean case, the dimension d of the input point set is fixed. Inaba et al. [67] show that to find an optimal partition of the point set it suffices to check all Voronoi partitions. The number of such partitions can be upper bounded by $\mathcal{O}\left(n^{\mathcal{O}(kd)}\right)$, giving an exact algorithm for constant k and d. In [83], Matoušek obtained a PTAS under these assumptions, with a runtime of $O\left(\varepsilon^{-2dk^2} n \log^k n\right)$. Har-Peled and Mazumdar [62] improve this to $\mathcal{O}\left(n + k^{k+2} \varepsilon^{-(2d+1)k} \log^{k+1} n \log^k \frac{1}{\varepsilon}\right)$ using coresets [2]. Feldman et al. [52] present the concept of weak coresets to obtain a PTAS with runtime $O\left(nkd + d \cdot \mathrm{poly}(k/\varepsilon) + 2^{\tilde{O}(k/\varepsilon)}\right)$. Kumar et al. give a PTAS for the setting where only k is fixed, in which case the runtime is $\mathcal{O}(nd)$ [76].

If we do not restrict k and d, the best known approximation ratios are significantly worse. In [72], Kanungo et al. present a local search algorithm that achieves an approx-

[2]In k-Means clustering, a coreset is a weighted subset $P' \subset P$ such that, for *every* set $\mathcal{C}$ of k centers, the cost of $\mathcal{C}$ on P' is at least $(1 - \varepsilon)\mathtt{kMeans}(\mathcal{C})$ and at most $(1 + \varepsilon)\mathtt{kMeans}(\mathcal{C})$, for some $\varepsilon > 0$. The size of P' may depend on $1/\varepsilon$.

imation ratio of $(9+\varepsilon)$ for every $\varepsilon > 0$. Additionally, they show that an approximation ratio of 9 is essentially the best that can be achieved for local search approaches. In [6], Ahmadian et al. tackle the problem via a primal-dual approach. They show that for every $\varepsilon > 0$ there exists a $(6.357+\varepsilon)$-approximation algorithm for k-Means. More precisely, they show that for every ε there exists a polynomial time algorithm that returns a solution of cost at most $(6.357+\varepsilon)\cdot LP$, where LP denotes the cost of optimal solution to the LP-relaxation. This directly implies an integrality gap of at most (6.357) for the standard LP-formulation. Grandoni et al. [59] improve their approach to get a 6.12903-approximation, while Cohen-Addad et al. [38] use a different approach to obtain a ratio of 5.912. So at the time of writing (December 2025), the gap between the best upper and best lower bound is still significant.

It should be noted that it is also possible to obtain constant factor approximations for k-Means by using constant factor approximation algorithms for k-Median, e.g. Jain and Vazirani [69], Charikar et al. [34], or Mettu and Plaxton [84]. While the best known ratios for k-Median are lower, they do not directly transfer to the k-Means objective. The reason for this is that the squared-distance cost function does not satisfy the triangle inequality, but only a relaxed version. As a consequence, every time a distance is bounded by the triangle inequality in the analysis for k-Median, an additional factor is introduced in the analysis for k-Median. This drastically worsens the analysis and leads to very large approximation factors.

Heuristics Since many of the algorithms mentioned so far are rather involved and often have runtimes that, although polynomial, make them difficult to use practically, k-Means often gets tackled by heuristics. These are algorithms that do not have any theoretical guarantee but can perform well in practical settings. We do not aim to provide a complete overview but rather name the most famous and important heuristics.

Probably the most-used and most well-known among these is Lloyd's Algorithm. It is often even referred to as *the k-means algorithm*. Originally developed by Stuart P. Lloyd in 1957, it was not formally published until 1982 [80]. It mainly exploits Corollary 68 by starting with arbitrary points as centers and then iteratively re-computing centroids and updating assignments. Due to its significance and its importance to the design of other algorithms (including our own work), it has its own dedicated section (see section III.1.2).

The method known as MacQueens Algorithm [81] uses a similar approach. Rather than looking at the whole dataset, it goes through the set point by point and, every time it computes a centroid, it does so based on only the points seen so far. Therefore, it also works in an online or streaming setting. On the other hand, it is sensitive to the input ordering and therefore rather unstable.

Another well-known heuristic is the Hartigan-Wong-Algorithm, which is used as the default method in the `kmeans` method of the programming language `R`. It starts with an arbitrary partition of the point set and then checks for every point and every cluster if assigning the point to that cluster would decrease the cost. More formally, for any point

p from cluster C_i, and every cluster $C_j, j \neq i$, it evaluates

$$\frac{|C_i|}{|C_i|-1} \cdot \|p-\mu(C_i)\|^2 - \frac{|C_j|}{|C_j|+1} \cdot \|p-\mu(C_j)\|^2.$$

For the points p and the clusters C_i, C_j that maximize this expression, p is reassigned from C_i to C_j. While this method is computationally more complex than Lloyd's method, it can often avoid the type of local minima that Lloyd's algorithm runs into, and therefore outperform it in terms of cost (cf. Telgarski and Vattani [90]).

Practical Approximation Algorithms Finally, there are also practical algorithms which give provable approximation guarantees, even if those guarantees are worse than those of the algorithms mentioned above. An important milestone in the field of algorithms for k-Means was the introduction of `k-means++` by Arthur and Vassilvitskii in [14]. Since this algorithm plays an important role in the design of our own algorithm, it has its own dedicated section (cf. section III.1.3). This improvement on the Lloyd heuristic yields an $\mathcal{O}(\log k)$-approximation in expectation and, more importantly, performs very well in practice. The algorithm `LS++` introduced by Lattanzi and Sohler in [77] combines ideas from `k-means++` and local search to obtain a constant factor approximation, albeit with a large constant. This algorithm is also discussed in more detail in section III.1.4.

More recently, Beretta et al. [24] proposed *Multi-Swap k-Means++*, an adaptation of `LS++` where multiple centers can be swapped out at once. They show that their approach achieves an approximation factor of $(9+\varepsilon)$ which is (as mentioned previously) best possible for a local search approach. Additionally, they show that their algorithm performs better than `LS++` in experiments as well.

In Chapter III.2, we describe our algorithm *Foresight LocalSearch++* (`FLS++`), first introduced in [41]. This algorithm also builds on `LS++` but modifies a different aspect of the local search procedure. It preserves the theoretical guarantees of `LS++` and outperforms it practically.

III.1.2. Lloyd's Algorithm

As mentioned before, Lloyd's Algorithm is so widely used that it is often referred to as *the k-Means algorithm.* Its idea is based on Corollary 68 and can be described as follows.

1. Pick the initial set of centers $\{c_1^0, c_2^0, \ldots, c_k^0\} \subset P$ arbitrarily.
2. Compute closest center assignment $\sigma(p) = \arg\min_{i\in[k]} d(p, c_i^0)$ for all $p \in P$.
3. For each emerging cluster $\sigma^{-1}(c_i^0)$, compute the new centroid $c_i^1 := \mu(\sigma^{-1}(c_i^0))$.
4. Repeat 2. and 3. until stopping criterion is met.

The idea of this algorithm is quite intuitive. Given any set of centers, we compute clusters by assigning each point to its closest center. As Corollary 68 shows, moving

Algorithm 10: `Lloyd's Algorithm`

```
  Input: A point set $P \subset \mathbb{R}^d$, a number $k \in \mathbb{N}$
  Output: A set $\mathcal{C} = \{c_1, \dots, c_k\} \subseteq P$ of centers, an assignment $\sigma : P \to \mathcal{C}$
1 Choose set $\mathcal{C} \subset \mathbb{R}^d$ of $k$ centers arbitrarily
2 forall $p \in P$ do
3     $\sigma(p) \leftarrow$ NULL
4 while Stopping criterion is not met do
5     forall $p \in P$ do
6         $\sigma(p) \leftarrow \arg\min_{c \in C} \|p - c\|$
7     forall $c \in \mathcal{C}$ do
8         $c \leftarrow \mu(\sigma^{-1}(c))$
9 return $\mathcal{C}, \sigma$
```

the centers to the cluster centroids minimizes the cost for that specific assignment. But changing the center locations might change the closest center for some points, so we recompute the assignment. Now we can again compute the centroids of the resulting clusters and iterate.

One can immediately observe that, in each iteration, the cost of the solution can only decrease. Naturally, the question when to stop this procedure arises. An obvious answer would be to stop as soon as no points are reassigned. In that case, the algorithm has converged to a local (not necessarily global) minimum. One can easily see that this stopping criterion is always reached after finitely many steps.

Observation 69 ([88])**.** *After at most $O(k^n)$ iterations, the set of centers does not change, i.e. $\{c_1^t, \dots, c_k^t\} = \{c_1^{t+1}, \dots, c_k^{t+1}\}$.*

Proof. Whenever we compute the closest center assignment, there are two possible outcomes. Either the center set does not change and we are already done, or the center set does change. But since we obtained the new centers by computing the closest center assignment, the cost must have strictly decreased. If the cost can only decrease, we can never end up with the same center set twice (up until the very last step). Each center set along with the closest center assignment defines a partition of the point set. Therefore, we can trivially upper bound the number of iterations by the number of possible partitions of n points into k subsets. This is k^n. ☐

Of course, this upper bound is somewhat pessimistic. Intuitively, it does not seem very likely that the algorithm will produce *all* possible partitions. But as it turns out, the lower bound is exponential as well, albeit with a constant base instead of k; there are instances where the number of iterations needed to converge is exponential in the size of the input n. In [63], Har-Peled and Sadri show that for $d = 1$ and $k = 2$ Lloyd's Algorithm needs $\Omega(n)$ iterations in the worst case. Subsequently, Arthur and Vassilvitskii [13] gave an instance with constant spread on which Lloyd's needs $2^{\Omega(\sqrt{n})}$

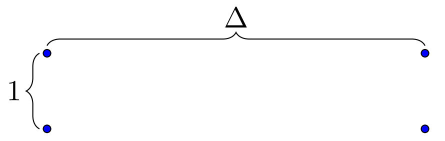
iterations. In their construction, the initial centers are chosen adversarially, but they show that even in the case of centers being chosen uniformly at random, the number of iterations is still superpolynomial with high probability. Vattani [92] showed that even for $d = 2$ there exists an instance on which the algorithm needs $2^{\Omega(n)}$ iterations. On the other hand, the fact that Arthur et al. [12] showed that Lloyd's algorithm has polynomial smoothed running time[3] hints at the fact that those worst case instances are rather fragile.

Stopping Criteria In its general form (cf. Algorithm 10), Lloyd's Algorithm does not specify a stopping criterion. The simplest approach is to limit the number of iterations beforehand, i.e. make the number of iterations a parameter of the algorithm. This has the disadvantage that, on different datasets and for different values of k, the number of iterations needed to converge (or get close to converging) can differ vastly. A more informed and adaptive strategy is to introduce a tolerance parameter for the cost. Then the algorithm is stopped as soon as the cost decrease in two consecutive iterations of the `while`-loop is below that threshold. This means that it is implicitly assumed that once the changes in cost are small, they will not get bigger again, which in general is not true. But in practice it is often the case that the algorithm ends with thousands of iterations where almost negligible improvements are made. To make sure that the algorithm behaves independently of the magnitude of the numbers in the dataset, this tolerance parameter can be chosen in relation to either the initial cost or the current cost. Another approach is to introduce a tolerance for the *movement* of centers. Then the algorithm terminates as soon as the center position does not change significantly between consecutive iterations.

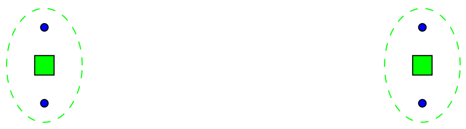
Quality of Solutions As discussed above, the runtime of Lloyd's algorithm is exponential in n in the worst case. But what about the quality of the solutions it provides? As mentioned earlier, it is a heuristic that does not yield any approximation guarantee at all. An easy example in two dimensions shows this. As illustrated in Figure III.1.2a, we have an instance consisting of 4 points. For $k = 2$, the optimum solution is obvious and has a cost of 1, as shown in Figure III.1.2b. But if we initialize the centers arbitrarily, we might choose the two points in the left cluster (see Figure III.1.2c). In this case, computing the closest center assignments and then computing the centroids leads to the solution shown in Figure III.1.2c, which has a cost of Δ^2. This is a local minimum, so the algorithm will terminate with this solution. Since Δ can be chosen arbitrarily large and the argument still works, we can conclude that the cost of the solution computed by Lloyd's Algorithm can be arbitrarily far off from the optimum.

Observation 70. *Lloyd's Algorithm does not yield an α-approximation for* Euclidean k-*Means for any* $\alpha \in \mathbb{R}_{\geq 0}$.

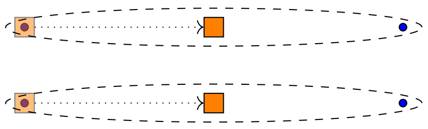
[3]In this framework, worst case instances are not constructed fully adversarially, but instead are subjected to slight random perturbations.



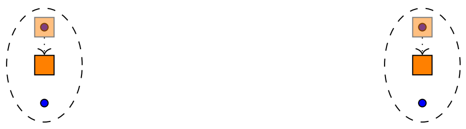
(a) An instance consisting of 4 points, forming two natural clusters (left and right). The two points within each cluster are at distance 1, but the two clusters are at distance $\Delta >> 1$ from each other.



(b) The obvious optimal 2-means solution forms two clusters of cost $1/2$ each, leading to an overall cost of 1.



(c) If the initial centers for Lloyd's are chosen poorly, it converges to a solution with cost Δ^2.



(d) If the initial centers are chosen favorably, Lloyd's converges to the optimum solution.

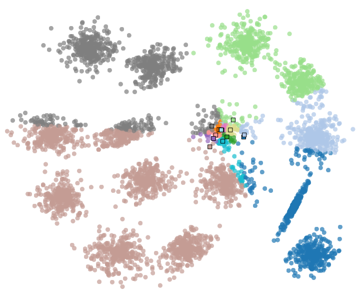
Figure III.1.2.: An example illustrating that Lloyd's Algorithm does not yield any approximation at all.

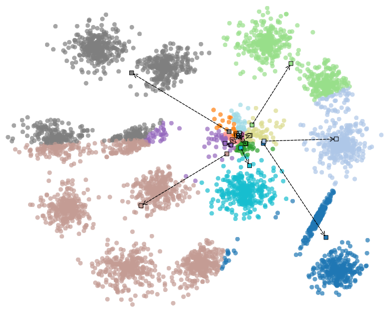
But, as Figure III.1.2d shows, with a good initial center choice, Lloyd's Algorithm *can* actually converge to the optimum solution.

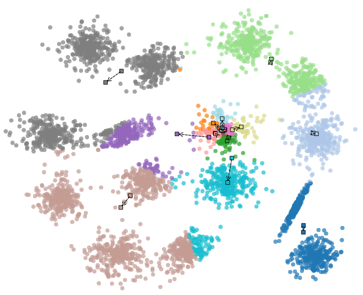
Even on less artificial datasets, a careless initialization can lead to bad local optima. Consider the example shown in Figure III.1.3 which shows an instance in 2 dimensions with 15 ground truth clusters. Here, the first 15 points listed in the dataset are chosen as initial centers, which are unfortunately very close to each other. Even though in the first iterations there are significant improvements, the final solution is visibly very far from optimal. But looking at the dataset, one might get the impression that having an initial center set where each ground truth cluster contains one cluster might again lead to Lloyd's Algorithm computing an optimal (or near-optimal) solution. We can therefore conclude that choosing a sensible initialization method is of high importance.

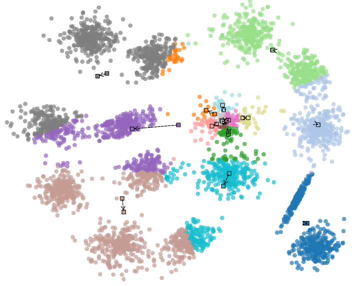
III.1.3. k-means++

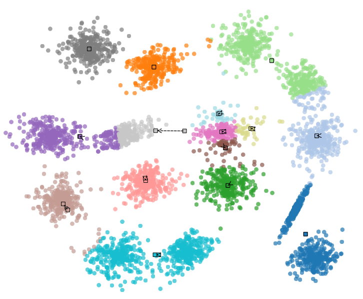
Since Lloyd's algorithm is so dependent on initialization, the question arises how to obtain good initial solutions. In [14], Arthur and Vassilvitskii proposed the randomized method of *D^2-sampling*. Before describing this method in detail, let us take another look at the bad example in Figure III.1.2 to get some intuition. The reason why the final solution is so bad is the fact that, although there are two clusters in the optimal solution, and they are very well separated, only one of them is "hit" by the initial center set - both points are located in the left optimal solution cluster. Lloyd's algorithm is then not able to "repair" this fault and converges to a very bad solution.


(a) An instance with 15 ground truth clusters. The initial centers are all located very close to each other.


(b) In the first iteration, some of the centers move a big distance.


(c) In the second iteration, there is less movement, but more centers move “outwards”.


(d) This trend continues in the third iteration.


(e) After 14 iterations, the last significant change happens.

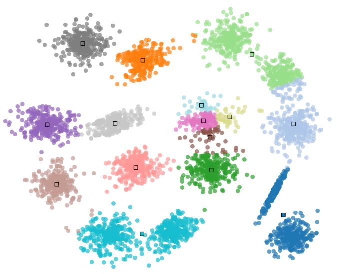

(f) In the final solution, some of the ground truth clusters have been recovered, but 3 of them do not have their own center.

Figure III.1.3.: An example illustrating the dependence of Lloyd’s algorithm on initialization.

It therefore seems desirable to obtain an initial solution where every optimal solution cluster is hit. Of course, there is no way of knowing the ground truth clusters beforehand. But Arthur and Vassilvitskii [14] introduce randomness to obtain a center set which, in expectation, hits many of the optimal clusters. They begin by sampling the first center c_1 uniformly at random from the input point set. In iteration $i > 2$, let $\mathcal{C}_{i-1} = \{c_1, \dots, c_{i-1}\}$ be the set of centers chosen so far. For each point $p \in P$, they compute a probability

$$\operatorname{prob}(p, \mathcal{C}_{i-1}) = \frac{\min_{c \in \mathcal{C}_{i-1}} \|c - p\|^2}{\sum_{q \in P} \min_{c \in \mathcal{C}_{i-1}} \|c - q\|^2}. \tag{III.1.1}$$

Note that this is exactly the relative contribution of p to the overall cost with respect to the current center set. They then sample a new center c_i according to these probabilities, add it to $\mathcal{C}_{i-1}$ to obtain $\mathcal{C}_i$ and continue with the next iteration. In other words, points that are far away from the current center set have a higher probability of being sampled than those that are closer. For our example in Figure III.1.2 this means that, after choosing an initial point uniformly at random, sampling the second center from the other optimal solution cluster is a lot more likely than sampling the second point from the same cluster. In Figure III.1.4, we see an exemplary run of the `KM++` initialization on the dataset from Figure III.1.3. The resulting center set is much more spread out among the dataset and provides a better starting point for Lloyd's algorithm.

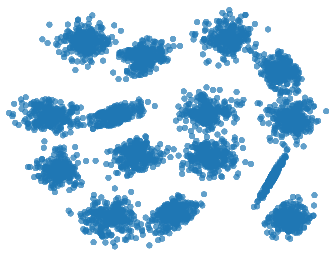
Algorithm 11: `SampleCenter`$(P, \mathcal{C})$

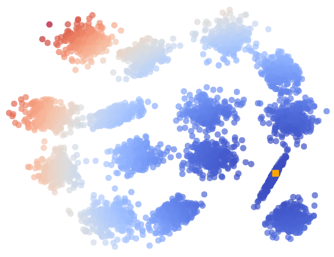
```
Input: A point set $P \subset \mathbb{R}^d$, a center set $\mathcal{C} \subset \mathbb{R}^d$
Output: A center $c \in P$
1 if $C = \emptyset$ then
2     Sample $c \in P$ uniformly at random
3 else
4     for $p \in P$ do
5         $\operatorname{prob}(p, \mathcal{C}) \leftarrow \frac{\min_{c \in \mathcal{C}} \|c-p\|^2}{\sum_{q \in P} \min_{c \in \mathcal{C}} \|c-q\|^2}$
6     Sample $c$ according to these probabilities
7 return $c$
```

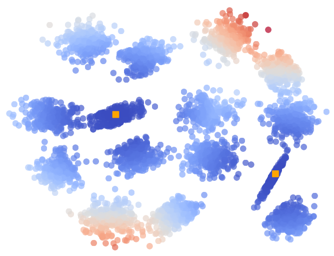
Algorithm 12: D^2`-sampling`(P, k)

```
Input: A point set $P \subset \mathbb{R}^d$, a number $k \in \mathbb{N}$
Output: A set $\mathcal{C} \subseteq P$ of centers
1 for $i = 1, \dots, k$ do
2     $c_i \leftarrow$ SampleCenter$(P, \mathcal{C})$
3     $\mathcal{C} \leftarrow \mathcal{C} \cup \{c_i\}$
4 return $\mathcal{C}$
```

Algorithm 11 implements the sampling of a single center, given a point set P and the current center set $\mathcal{C}$. Algorithm 12 constructs the complete initial center set by calling Algorithm 11 k times. The intuition that this might be a sensible strategy is backed up by the following result shown in [14].


(a) A 2D dataset with $k = 15$ ground truth clusters. An initial center will be chosen uniformly at random.


(b) The probabilities with respect to the first center are computed, where darker shades of blue indicate lower and darker shades of red higher probabilities.


(c) After a second center is sampled from these probabilities, the distribution is updated with respect to the new center set.

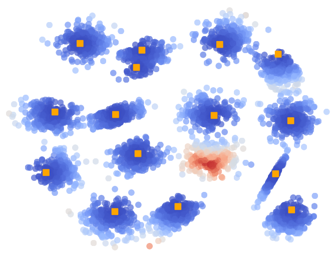

(d) The resulting center set after completing the initialization. All clusters except for one are hit.

Figure III.1.4.: An example run of the k-means++ initialization.

Theorem 71. *[14] The expected cost of the center set computed by D^2-sampling is at most* $(8 \ln k + 2)$ *times the cost of an optimum solution.*

Of course, it is not guaranteed that we sample a point from a new ground truth cluster in each iteration. Even in the example in Figure III.1.2, the probability of sampling the second center from the same ground truth cluster is larger than zero. But Theorem 71 tells us that the *expected* cost of the center set obtained by D^2-sampling is off by at most a factor of $\mathcal{O}(\log k)$.

In their paper, Artur and Vassilvitskii also give an instance on which the expected cost of `kmeans++` is $\Omega(\log k)$ times the optimum value. So this approximation factor is (asymptotically) tight. However, Aggarwal et al. [5] argue that this is somewhat misleading, as even though the expected cost of `kmeans++` on this point set is $\Omega(\log k) \cdot$ OPT, it yields a constant factor approximation with high probability. In other words, the reason for the approximation factor being $\mathcal{O}(\log k)$ on that instance is not that the algorithm returns an $\mathcal{O}(\log k)$-approximate solution with high probability, but rather that it returns a constant factor approximation with high probability, and in the unlikely case that it does not return a constant factor approximation, the cost is *very* high, distorting the expected value. They therefore conjecture that `kmeans++` yields a constant factor approximation with constant probability. However, Brunsch and Röglin [30] show that this is not the case. More precisely, they show that there exists a class of instances on which `kmeans++` has an approximation guarantee of $\Omega(\ln k)$ with probability arbitrarily close to 1. Since their instance and the one in [14] both have dimension $\Theta(k^2)$, they pose the question whether the approximation guarantee might actually be $\mathcal{O}(\log d)$. In [26], Bhattacharya et al. settle this question by giving a two-dimensional instance on which `kmeans++` only achieves an approximation ratio of $\mathcal{O}(\log k)$ with probability exponentially small in k.

Contrasting this with the fact that Lloyd's Algorithm on its own does not yield any approximation guarantee whatsoever, $\log k$ is a big improvement [4]. But contrasting this with the best known approximation algorithm for k-means, which is currently 5.912 (due to [38]), this seems a bit weak. Probably the biggest advantages of `k-means++` are its simplicity and speed. The known constant-factor approximations rely on complex primal-dual algorithms, whose runtimes include rather large polynomials. In contrast, D^2-sampling can be implemented to run in $\mathcal{O}(kn)$, and Lloyd's Algorithm often converges quickly. Additionally, there has been a series of works on heuristically speeding up Lloyd's Algorithm and adjusting the initialization to get better results (cf. III.1.3.2).

III.1.3.1. Greedy k-Means++

In the conclusion of [14], the authors claim that during their experiments, `k-means++` performed better when employing a slight variant of D^2-sampling. In this variant, there is an additional parameter $\ell \in \mathbb{N}_{>1}$, and in each iteration we sample ℓ center candidates according to the probabilities given in Equation III.1.1. We then greedily choose the one that decreases the cost the most (see Algorithm 13). This immediately raises the question whether this apparent improvement can be backed up by theoretical results. As it turns out, this is not the case. In fact, Bhattacharya et al. [25] show that the approximation guarantee actually gets worse as ℓ increases. More precisely, they show that for every $k \geq 4$ and every $\ell \geq 1$ there exists a point set $X_{k,\ell}$ on which the expected approximation guarantee of greedy k-means++ is $\Omega\left(\min\{\ell, k/\log k\} \cdot \log k\right)$. In [60], Grunau

[4]It should also be noted that Agarwal et al. [5] show that sampling $\lceil 16k + \sqrt{k} \rceil$ centers with $D^2 -$ `sampling` yields a constant factor approximation in terms of cost. They then proceed to show how to pick a subset of exactly k centers among these that give a proper $\mathcal{O}(1)$ approximation.

Algorithm 13: `Greedy` D^2`-sampling`

```
   Input: A point set $P \subset \mathbb{R}^d$, a number $k \in \mathbb{N}$
   Output: A set $\mathcal{C} = \{c_1, \ldots, c_k\} \subseteq P$ of centers
1  $\mathcal{C} \leftarrow \emptyset$
2  Sample $\{c_{1,1}, \ldots, c_{1,\ell}\} \subseteq P$ uniformly at random
3  Let $c_1 = \arg\min_{i=1}^{\ell}$ kMeans$(\{c_\ell\}, P)$
4  $\mathcal{C} \leftarrow \mathcal{C} \cup \{c_1\}$
5  for $i = 2, \ldots, k$ do
6      $\mathcal{C}_i \leftarrow \emptyset$
7      for $j = 1, \ldots, \ell$ do
8          $c_{i,j} \leftarrow$ SampleCenter$(P, \mathcal{C})$
9          $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \{c_{i,j}\}$
10     Let $c_i = \arg\min_{c \in \mathcal{C}_i}$ kMeans$(P, \mathcal{C} \cup \{c\})$
11     $\mathcal{C} \leftarrow \mathcal{C} \cup \{c_i\}$
12 return $C$
```

et al. subsequently improve that lower bound to $\Omega\left(\ell^3 \log^3 k / \log^2(\ell \log k)\right)$, while also showing an upper bound of $\mathcal{O}(\ell^3 \log^3 k)$.

Nonetheless, this heuristic seems to perform quite well in practice, just as Arthur and Vassilvitskii claimed. For more details on this, see III.2.2.3. It is also currently the default initialization method used by the popular `scikit-learn` library [87]. In this implementation, ℓ can be passed as input argument and its default value is $\lceil 2 + \ln k \rceil$.

III.1.3.2. Heuristic runtime improvements for Lloyd's Algorithm

Due to its continued popularity, there exist many heuristic strategies to speed up Lloyd's algorithm. They mostly aim at reducing the number of distance calculations in the reassignment step (cf. line 6 in Algorithm 10). Implemented naively, the algorithm has to compute the distance between every point and every center in every iteration. Especially for high-dimensional data, distance calculations can be costly, so some algorithmic overhead might be acceptable if it will likely avoid many such calculations.

A classic result was given by Elkan [50] and utilizes the triangle inequality. The heuristic maintains a lower and an upper bound on the distance to the closest center for each point. These values are updated lazily (i.e. not in each iteration but only when it becomes necessary) and often allow for a significant number of distance computations to be skipped. In the experiments, Elkan shows that the number of distance calculations needed by the standard algorithm can be up to 3 powers of magnitude larger than those needed by the heuristic. Especially for larger k the difference becomes highly noticeable.

A somewhat complimentary heuristic was proposed by Hamerly in [61]. It also works with upper and lower bounds on the closest center distance for each points, especially with an improved lower bound. In their experiments, Hamerly found that, especially on low dimensional data, the method avoids up to 80% of the executions of the for-loop in Line 5 of Algorithm 10. However, on high dimensional data, Elkan's method still tends to be superior.

A third method of reliably reducing distance computations was given by Xia et al. [94]. Their method, called *Ball k-means*, is based on two intuitive concepts. Like the heuristics by Elkan and Hamerly, it exploits the fact that, even if a point changes cluster membership, it is not going to change to a cluster that is very far away. This is used to create a notion of neighborhood between clusters, and for each point we only compute distances to centers of neighboring clusters. The second insight that gets exploited is the fact that points that are very close to their center are very unlikely to get reassigned in the next iteration. This is used to divide each cluster into a *stable* and an *active* region, where points in the stable region are ignored during distance recomputations. In their experiments, they show that their method often outperforms other heuristics, especially for large k. However, for very high dimensions, it becomes increasingly likely that a large percentage of cluster are neighbors, negatively impacting the speed-up.

III.1.4. Local-Search++

The algorithm `Local-Search++` (abbreviated as `LS++`) was introduced by Lattanzi and Sohler in [77]. It can be viewed as `k-means++` with an extended initialization phase. As the name suggests, it incorporates elements of the local search paradigm. Generally speaking, this is an approach where a notion of neighborhood between solutions is introduced. In our specific case, two centers sets are neighboring when they differ in exactly one center. A local search algorithm starts with some initial solution and checks whether there exists a neighboring solution that would improve the solution quality. This is repeated until it converges to a (not necessarily global) minimum. In `LS++`, this unfolds as follows. After choosing a set of k centers $\mathcal{C} = \{c_1, \ldots, c_k\}$ with D^2-`sampling`, it samples a $(k+1)$-th center c_{k+1}, where the probabilities are computed in the same way as before (with respect to $\mathcal{C}$). For $i \in \{1, \ldots, k\}$, let $\mathcal{C}'_i = \mathcal{C} \setminus \{c_i\} \cup \{c_{k+1}\}$ be the center set arising from $\mathcal{C}$ by exchanging c_i for the newly sampled c_{k+1}. We now consider all $\mathcal{C}'_i$ for $i \in [k]$ and keep the one with the lowest cost. We refer to the process of exchanging one c_i for c_{k+1} as a *swap.* Trying all k swaps for a set $\mathcal{C}$ is called a *local search step.* Note that a local search step does not need to result in the center set changing; if the sampled center c_{k+1} is bad, or the present centers are very good, none of the swaps would actually decrease the cost.

The algorithm performs a specified number of local search steps and subsequently calls Lloyd's algorithm on the resulting center set. This algorithm is of special interest for this thesis, as the algorithm `FLS++`, presented in Chapter III.2, directly builds and improves upon `LS++`. To describe it formally (see Algorithm 14), we assume that we have a function `Lloyd`$(P, \mathcal{C}, s)$, which, given a point set P, a center set $\mathcal{C}$ and a number of iterations s, returns the center set emerging from $\mathcal{C}$ by running s iterations of Lloyd's algorithm. It receives a set P of points from d-dimensional Euclidean space, a number k of centers and a number s of iterations for Lloyd's algorithm. Additionally, it receives a

Algorithm 14: `LocalSearch++`

```
   Input: Point set $P \subset \mathbb{R}^d$, numbers $k, s \in \mathbb{N}$
   Parameters: $Z \in \mathbb{N}_0$
   Output: Center set $\mathcal{C} \subset \mathbb{R}^d$
1  $\mathcal{C} = D^2$-sampling$(P, k)$
2  for $i = 1$ to $Z$ do
3      $c_{\text{new}} =$ SampleCenter$(P, \mathcal{C})$
4      $\mathcal{C}^{\min} = \mathcal{C}$
5      for $c_{old} \in \mathcal{C}$ do
6          $\mathcal{C}' = (\mathcal{C} \setminus \{c_{\text{old}}\}) \cup \{c_{\text{new}}\}$
7          if kMeans$(P, \mathcal{C}') <$ kMeans$(P, \mathcal{C}^{\min})$ then
8              $\mathcal{C}^{\min} = \mathcal{C}'$
9      $\mathcal{C} = \mathcal{C}^{\min}$
10 $\mathcal{C}, \sigma =$ Lloyd$(P, \mathcal{C}, s)$
11 return $\mathcal{C}, \sigma$
```

parameter Z specifying the number of desired local search steps. Lattanzi and Sohler [77] show that, for $Z \geq 100\,000k \log\log k$, the expected cost of the resulting center set yields a constant-factor approximation. Additionally, they perform experiments which indicate that already for values of Z as small as 25, `LS++` yields significant advantages over `kmeans++`. Subsequently, Choo et al. [37] improve the analysis to show that, for $\varepsilon \in (0, 1]$, performing εk local search steps leads to a cost of $\mathcal{O}(1/\varepsilon^3)$ with probability at least $1 - e^{-\Omega(k^{0.1})}$.

We also include `LS++` in our empirical analysis in section III.2.2. In these experiments, we can validate the claims from [77] that `LS++` already yields good results for very small values of Z.

III.2. Foresight LocalSearch++

The algorithm `Foresight-LocalSearch++` (or `FLS++`) was introduced in *Local Search k-means++ with Foresight* (SEA'24 [41], ArXiv [42]) by Conrads, Drexler, Könen, Schmidt and Schmidt. In his Master's Thesis [40], Conrads already explored a preliminary version of the algorithm. It can be viewed as a natural extension of `LS++` where the decision whether a center gets swapped out is — in a sense — more *informed*. In the following, we give an intuitive motivation for the algorithm, then describe it in detail, before we finally look at an experimental evaluation. We compare `kmeans++`, `LS++` and `FLS++` and their greedy versions on datasets of different sizes and dimensions. For a subset of datasets where we know actual optimum solutions, we explore how close the algorithms get to these solutions. Finally, we briefly discuss the question of clusterability of datasets and how it can impact the performance of algorithms.

III.2.1. Idea and Description

The motivation for `FLS++` stems from the following observation. Consider any local search step in `LS++`. Let $\mathrm{c}_{\mathrm{new}}$ be the point that gets sampled in this iteration and let $\mathcal{C} = \{c_1, \ldots, c_k\}$ be the current center set. Ideally, we want $\mathrm{c}_{\mathrm{new}}$ to be part of a ground truth cluster that has not been hit by $\mathcal{C}$ yet. But it might happen that, even when we actually do hit a new cluster, the sampled point lies on the fringes of that cluster, i.e. rather far away from the desired cluster center. Since the algorithm only samples from input points, it might not even have the chance to sample a point close to the actual center. Note that this is, of course, also true for the initial k centers chosen by D^2`-sampling`. Thus, a sampled point might *seem* like a much worse candidate than it actually is. But we can use Corollary 68 to our advantage. Any set of centers along with the closest center assignment defines a partition – i.e., a clustering – of the point set. And for this particular clustering, we *do* know the optimal centers, namely the centroids. So whenever we consider a swap in a local search step, we update the centroids and judge the swap by the cost with respect to these updated centers.

A more formal description of `FLS++` is given in Algorithm 15. In line 5, it performs one Lloyd step (i.e. computing the centroids of the current partition). In the following `for`-loop, it swaps every old center with the newly sampled one and performs one Lloyd step for all resulting center sets. Finally, it keeps the center set where the cost decreased the most.

Algorithm 15: `Foresight-LS++`

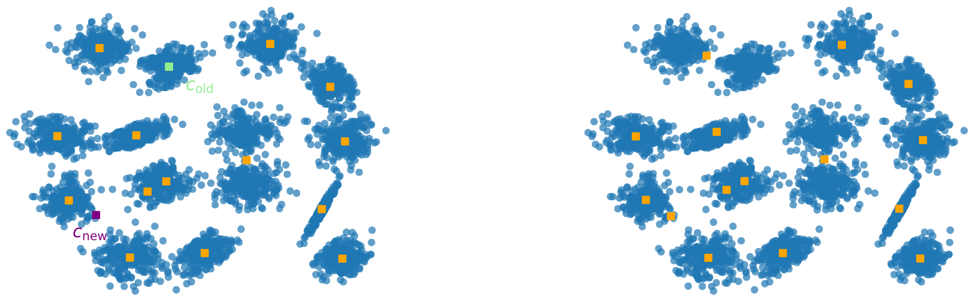
```
Input: Point set $P \subset \mathbb{R}^d$, numbers $k, s \in \mathbb{N}$
Parameters: $Z \in \mathbb{N}_0$
Output: Set of $k$ centers $\mathcal{C} \subset \mathbb{R}^d$
1  $\mathcal{C} = d^2$-sampling$(P, k)$
2  $\mathcal{C}, \sigma =$ Lloyd$(P, \mathcal{C}, 1)$
3  for $i = 1$ to $Z$ do
4      $c_{\text{new}} =$ SampleCenter$(P, \mathcal{C})$
5      $\mathcal{C}^{\min}, \sigma^{\min} =$ Lloyd$(P, \mathcal{C}, 1)$
6      for $c_{old} \in \mathcal{C}$ do
7          $\mathcal{C}', \sigma' =$ Lloyd$(P, (\mathcal{C} \setminus \{c_{\text{old}}\}) \cup \{c_{\text{new}}\}, 1)$
8          if kMeans$(P, \mathcal{C}', \sigma') <$ kMeans$(P, \mathcal{C}^{\min}, \sigma^{\min})$ then
9              $\mathcal{C}^{\min}, \sigma^{\min} = \mathcal{C}', \sigma'$
10     $\mathcal{C} = \mathcal{C}^{\min}$
11 $\mathcal{C}, \sigma =$ Lloyd$(P, \mathcal{C}, s)$
12 return $\mathcal{C}, \sigma$
```

Figure III.2.1 shows an example of a swap in a local search step where an unlucky center c_{new} was sampled. After swapping out the first c_{old} and performing one Lloyd iteration, the resulting center set is much worse than the initial one. In fact, every consecutive swap for this particular c_{new} will not yield a good solution, as it is located in a cluster that is already served by a center. Figure III.2.2, on the other hand, depicts a swap where c_{new} was sampled in a favorable location. The ground truth cluster it is located in is currently lacking its "own" cluster. Additionally, the c_{old} that is being swapped out is in a cluster that already has another center. So this is exactly the type of swap that we are looking for.

One can view `FLS++` not only as a modified `LS++` algorithm, but also as a modified Lloyd's algorithm. Viewed as a modified Lloyd's algorithm one can think of `FLS++` as performing Lloyd's algorithm but doing a center swap in every iteration to avoid running into local optima. Viewed as a `LS++` modification, one can think of it as running Lloyd's algorithm for one step between the local search improvement steps. Why do we run exactly one step of Lloyd's algorithm? The reason is that this can be done very efficiently. Running multiple Lloyd's steps for *each* c_{old} would incur a running time of $O(ndk^2)$ for *each* local search step. But recomputing the centroids can be done cleverly for all points together such that one local seach step in Algorithm 15 takes $\mathcal{O}(ndk)$ just as for `LS++`.

Lemma 72. *One iteration of the For-Loop in Lines* 3-13 *of Algorithm 15 can be implemented such to run in time* $\mathcal{O}(ndk)$.

Proof. First we compute the distances of every point p to its closest center $\sigma_1(p)$ and second-closest center $\sigma_2(p)$ in time $\mathcal{O}(ndk)$. With this information, we can sample a point c_{new} with d^2-sampling in time $\mathcal{O}(n)$. Next, we compute the distances of all points to the sample point c_{new}. Then we check all k current centers whether we want to swap



(a) The red center c_{new} was sampled and is swapped in for the green center c_{old}.

(b) Afterwards, one iteration of Lloyd's is performed on the changed center set.

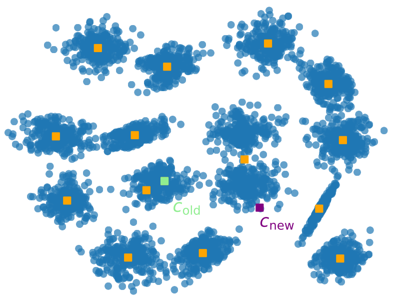
Figure III.2.1.: An example of an `FLS++` swap in a local search step where c_{new} was sampled unfavorably. The swap does not improve the solution and is therefore rejected.

it out. For one candidate c_{old}, we run through all n points p, and determine which is its closest center. We can do this in time $\mathcal{O}(1)$ in the following way: if $\sigma_1(p) \neq c_{\text{new}}$, then we compare $d(p, \sigma_1(p))$ with $d(p, c_{\text{new}})$. If $\sigma_1(p) = c_{\text{new}}$, then we compare $d(p, \sigma_2(p))$ with $d(p, c_{\text{new}})$. After doing this for all c_{old} and all p, we have determined the cluster membership of all points in time $\mathcal{O}(kn)$. Finally, we want to recompute the centroids and costs of all clusters which we can do in time $\mathcal{O}(nd)$. It is important to note that we do not recompute the reassignment of all points to the new set of centers as this would increase the overall running time $O(ndk^2)$. This is delayed until the next iteration and thus only done for the chosen $\mathcal{C}^{\min}$. □

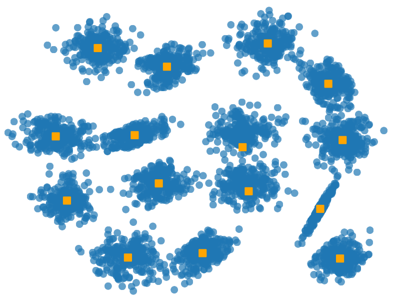
It should be noted that, in initial tests, we tried out more Lloyd iterations per local search steps. In his master's thesis [40], Theo Conrads even tried out a variant where Lloyd is run until convergence every time. But as it turns out, the computational overhead of doing more than one Lloyd's step per local search iteration outweighs the benefits. A nice side result is the insight that the approximation guarantee that `LS++` gives is also not affected.

Corollary 73. *FLS++ with $Z \geq 100\,000 k \log\log k$ computes a constant factor approximation.*

Proof. [77] shows that for any $\mathcal{C}$ with cost larger than $500 \cdot OPT$ and a c_{new} sampled by d^2-sampling, there is a c_{old} such that the cost of $(\mathcal{C} \setminus \{c_{\text{old}}\}) \cup \{c_{\text{new}}\}$ compared to the cost of $\mathcal{C}$ is smaller than that of $\mathcal{C}$ by a fraction of $(1/100k)$ with probability $1/1000$ (Lemma 3 in [77]). The analysis of the algorithm follows from this fact. Lloyd's steps are always improving, since the centroid is always the best center for a cluster. Thus Lemma 3 in [77] still holds and thus does the approximation guarantee. □



(a) The red center c_{new} was sampled and is swapped in for the green center c_{old}.



(b) Afterwards, one iteration of Lloyd's is performed on the changed center set.

Figure III.2.2.: An example for a good swap of `FLS++`. c_{new} hits a new cluster and c_{old} is exactly the center we want to get rid of. The solution after performing one Lloyd iteration on the new center set is a clear improvement.

III.2.2. Empirical Analysis

In this section, we present experimental results comparing `k-means++`, `LS++` and `FLS++` in various settings. Some experiments were done for the paper *Local Search k-Means++ with Foresight* [41], while the experiments in Section III.2.2.5 were done afterwards, specifically for this thesis.

We implemented all three algorithms in C++. The computations of the experiments in sections III.2.2.2, III.2.2.3 and III.2.2.4 were performed on an Intel(R) Xeon(R) E3-1240 running at 3.7Ghz and 8 cores, and 32GB RAM. The experiments in Section III.2.2.5 were performed on an AMD Ryzen 7 PRO 5850U with 32GB RAM. For some instances, we also computed optimal assignments with the *mssc-dca* algorithm presented in [89]. This algorithm combines column generation with dynamic constraint aggregation to reduce the number of constraints consideren in the column generation. These computations were performed on an AMD Ryzen 5 3400G with 80GB RAM.

The code and datasets used can be found at `https://github.com/lukasdrexler/flspp_code`. In the following, when necessary, we abbreviate `k-means++` with `KM++`. We refer to the greedy variant of `KM++` as `GKM++`. Since the results in III.2.2.3 suggest that the greedy variants for `LS++` and `FLS++` are superior to their non-greedy counterparts, we use the greedy variants by default. In most experiments, we simply refer to them as `LS++` and `FLS++` and only specifically write `GLS++` and `GFLS++` when comparing it to the non-greedy versions in section III.2.2.3.

dataset	n	d	source
rectangles	1 296	2	[54]
pr91	2 392	2	[86]
drilling	1 060	2	[89]
fl3795	3 795	2	[89]
fl417	417	2	[89]
666cities	666	2	[89]
D31	3 100	2	[54]
s1	5 000	2	[54]
s3	5 000	2	[54]
rl5934	5 934	2	[54]

dataset	n	d	source
A3	7 500	2	[54]
circles	10 000	2	[2]
close circles	10 000	2	[2]
Tower	4 915 200	3	[53]
Clegg	716 320	3	[53]
Frymire	1 235 390	3	[53]
Body measurements	507	5	[64]
Concrete strength	1 030	9	[95]
KDD Bio Train	145 751	74	[1]
KDD Phy Test	100 000	78	[1]

Table III.2.1.: Overview of datasets used in the experiments.

III.2.2.1. Datasets

Table III.2.1 lists all datasets used for evaluation in this section, along with their sources. In Appendix A, the 2-dimensional datasets are plotted for reference.

For some datasets, the choice of k is obvious, as there exists a clear ground truth clustering: *rectangles* (Figure A.2e) consist of $k = 36$ clearly separated clusters, *D31* (Figure A.2d) consists of 31 centers that are not as well separated, but still clearly visible; likewise, the two synthetic datasets *circles* and *close circles* consist of $k = 100$ separated clusters, as visualized in Figures A.2b and A.2c. But looking at *666cities* (Figure A.1a), it is less clear what k to choose. Also, for higher dimensional data it is not possible to "look" at the data for a clue as to what the optimal value for k is. Therefore, we evaluated several values of k for many of the datasets, which where chosen either because they were also chosen in other papers, and so we have something to compare our results to, or because the optimum solutions for these k are actually known (cf. section III.2.2.5). The choice of k and its impact on solution quality as well as the significance of optimal solutions is discussed in more detail in Section III.2.3.

III.2.2.2. Comparison of `kmeans++`, `LS++` and `FLS++`

In this section, we aim at comparing `FLS++` to `LS++` as its direct inspiration, and `KM++` as the most commonly used algorithm. The main goal, of course, is to find the cheapest solution. But we also take the computational effort needed to obtain these solutions into considerations. Therefore, we perform different types of experiments in order to get a better understanding of the trade-off between speed and solution quality. Since all algorithms are randomized, we let them run for a certain number of repetitions, each time with a different random seed. We analyze their minimum, mean, median and maximum cost over these runs. First, we take a look at the performance of these algorithms on the largest of our datasets. As the two local search algorithms–by design–have to do more computational work, we then change the setup a bit. We set a fixed time limit and then let all algorithms perform as many repetitions as they can finish in that time. This way, we get a better picture of what the algorithms can achieve in a certain amount of time, rather than within a certain number of repetitions. Finally, we take a look at how often any given algorithm was able to find the minimum solution among all three algorithms.

Overall Performance on Big Datasets. To get an idea of the algorithms' strengths and weaknesses, we compare the results of 50 runs of all four algorithms on three large datasets, *KDD Bio Train*, *KDD Phy Test*, and *Tower*. A *run* consists of 50 calls of each algorithm with a specific k on each dataset. The initial centers for `GLS++` and `GFLS++` are chosen by greedy d^2-sampling. We execute `GKM++`, `GLS++` and `GFLS++` with the same initial set of centers in a run.

Table III.2.2 shows the maximum, average and minimum cost that the algorithms produced on each dataset with $k = 100$. While the overall differences are not too big, `FLS++` consistently obtains the lowest values.

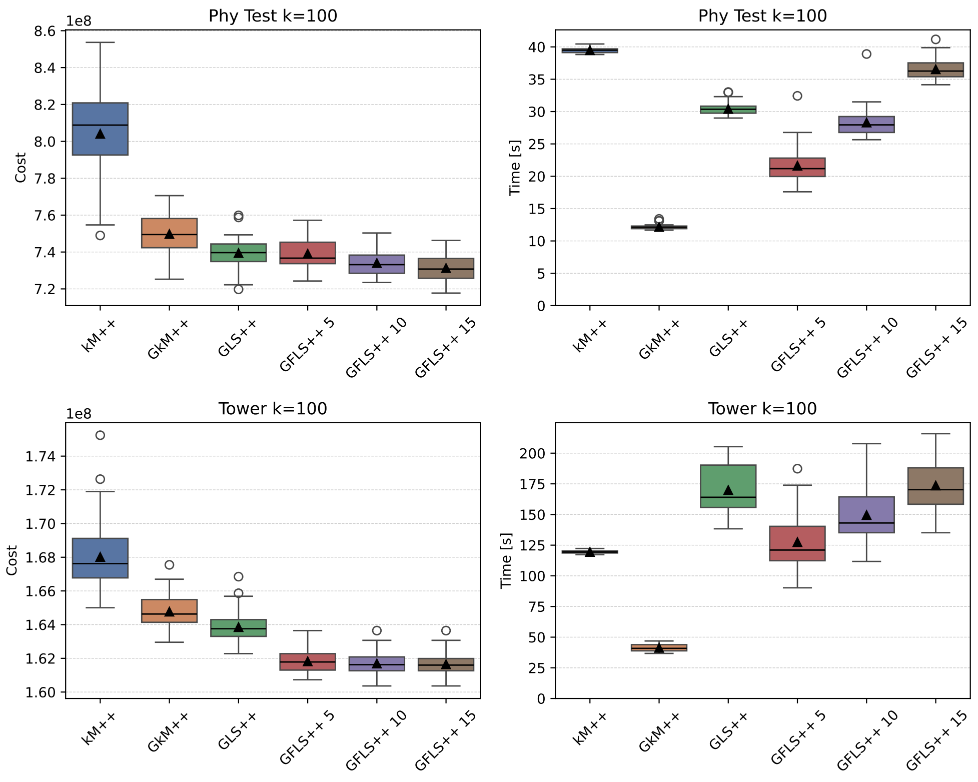


Figure III.2.3.: Comparison on two large datasets, for $R = 50$ repetitions. `GLS++` always performs 25 local search steps. For `GFLS++`, we display the results for $Z = 5, 10, 15$.

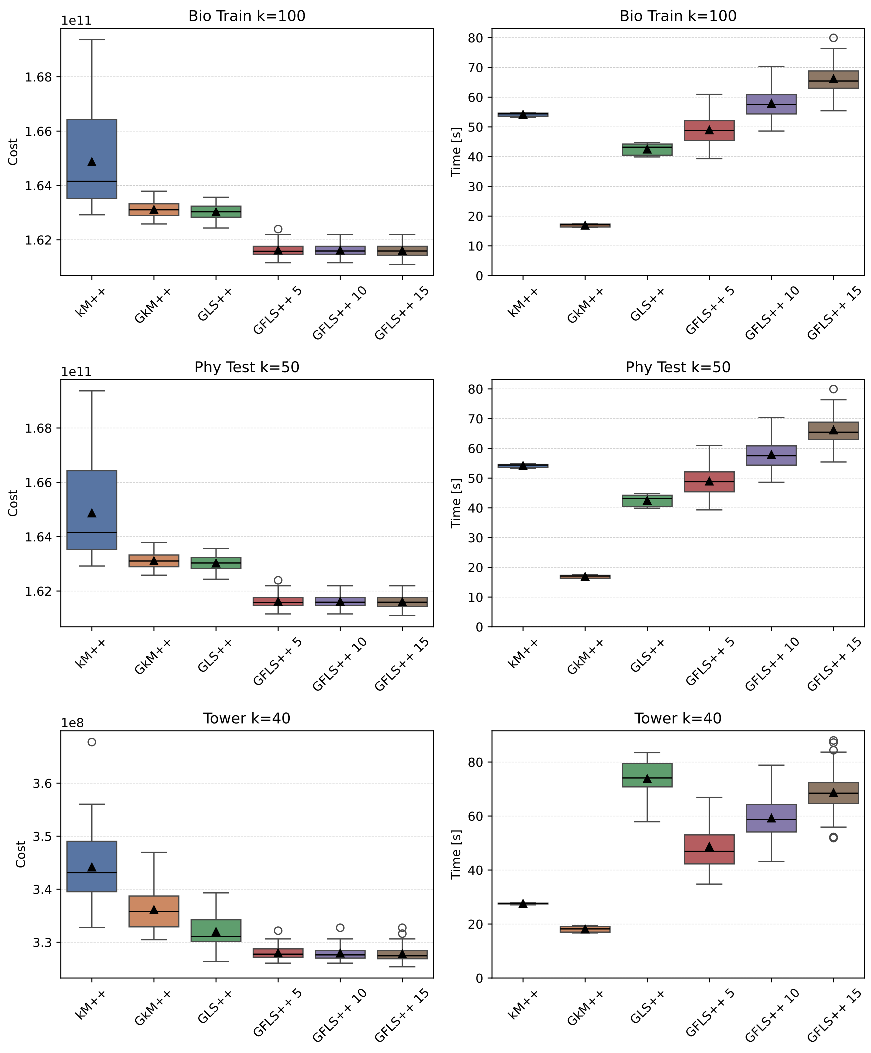


Figure III.2.4.: Performance boxplots for additional values of k on *Tower* and *KDD Phy Test*, as well as the additional dataset *KDD Bio Train*.

Datasets		Algorithms					
		KM++	GKM++	LS++	FLS++ 5	FLS++ 10	FLS++ 15
Bio Train	**Min**	1.6292E+11	1.6258E+11	1.6244E+11	1.6116E+11	1.6116E+11	**1.6110E+11**
	Mean	1.6487E+11	1.6311E+11	1.6302E+11	1.6162E+11	1.6162E+11	**1.6160E+11**
	Max	1.6937E+11	1.6379E+11	1.6357E+11	1.6240E+11	**1.6219E+11**	**1.6219E+11**
Phy Test	**Min**	7.4898E+08	7.2525E+08	7.1975E+08	7.2427E+08	7.2348E+08	**7.1773E+08**
	Mean	8.0406E+08	7.4972E+08	7.3942E+08	7.3905E+08	7.3400E+08	**7.3128E+08**
	Max	8.5367E+08	7.7058E+08	7.5988E+08	7.5721E+08	7.5032E+08	**7.4630E+08**
Tower	**Min**	1.6500E+08	1.6296E+08	1.6228E+08	1.6073E+08	**1.6036E+08**	**1.6036E+08**
	Mean	1.6802E+08	1.6477E+08	1.6385E+08	1.6182E+08	1.6169E+08	**1.6164E+08**
	Max	1.7524E+08	1.6755E+08	1.6685E+08	**1.6365E+08**	**1.6365E+08**	**1.6365E+08**

Table III.2.2.: Cost comparison on large datasets with $k = 100$ and 50 runs.

Dataset	$c_{\text{KM++}}$	$c_{\text{LS++}}$	$c_{\text{FLS++}}$	$C(\text{FLS++}, \text{KM++})$	$C(\text{FLS++}, \text{LS++})$
pr91	9.5637E+08	9.5276E+08	9.4696E+08	0.98%	0.61%
bio train features	2.3918E+11	2.3885E+11	2.3864E+11	0.23%	0.09%
concrete	3.2724E+06	3.1775E+06	3.1308E+06	4.33%	1.47%
circles	3.1913E+05	2.7164E+05	2.4773E+05	22.37%	8.8%

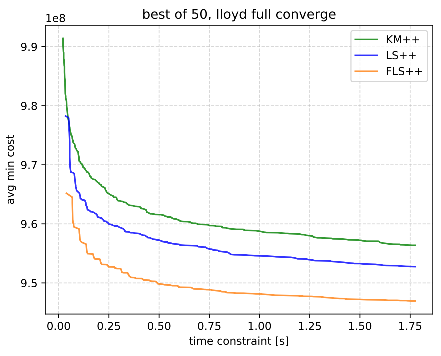
Table III.2.3.: Average cost. Last two columns show $(1 - c_{\text{FLS++}}/c_A) \cdot 100\%$ for $A \in \{\text{KM++}, \text{LS++}\}$.

In Figure III.2.3, we compare `KM++`, `GKM++`, `GLS++` with 25 local search steps (which is the number chosen in [77]) and `GFLS++` with 5, 10, and 15 steps. Since `GFLS++` performs one Lloyd iteration in every local search step, 25 such steps take longer than 25 steps in `GLS++`. Hence, we compare how `GFLS++` performs when using *fewer* local search steps. We take a more detailed look at the trade-off between runtime and cost further below. As expected, a larger value of Z increases runtime but decreases cost. On both *Phy Test* and *Tower*, `GFLS++` with $Z = 15$ obtains the overall minimum cost at the expense of a longer runtime, while outperforming `GLS++` slightly at $Z = 10$ in terms of cost *and* runtime. Figure III.2.4 shows similar results for *KDD Phy Test* with $k = 50$, *Tower* with $k = 40$ and *KDD Bio Train* with $k = 100$. For *Tower* with $k = 40$ `GFLS++` even outperforms `GLS++` at $Z = 5$ in terms of cost *and* runtime. In all cases, `GFLS++` found the best value among all algorithms.

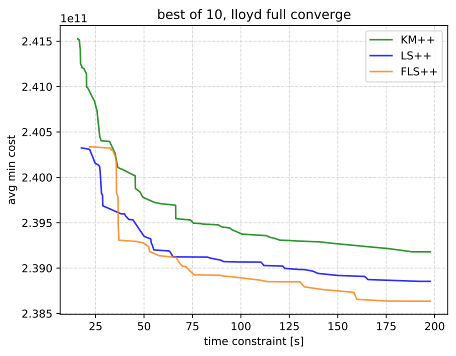
Best of Repeated Runs Within a Time Limit. As we have seen in the previous section, solutions computed by `GFLS++` have a smaller cost than `GKM++` or `GLS++` on average, but the algorithm needs more time to terminate. Since all algorithms are randomized, being able to execute one of them more often than others in a fixed amount of time might be an advantage. Therefore, we set some time limit and analyze if running `KM++` or `LS++` multiple times can yield better results than `GFLS++` in the same time. Thus, we repeat each algorithm until a time limit is reached and for each algorithm only the best solution is returned.

We choose some iteration value $B \in \mathbb{N}$ and report the best solution found by exactly B repetitions of `GFLS++`. We then repeat (G)`KM++` and `GLS++` as long as their respective elapsed time is less or equal to the time used by `FLS++`; again, we return the best solution found.

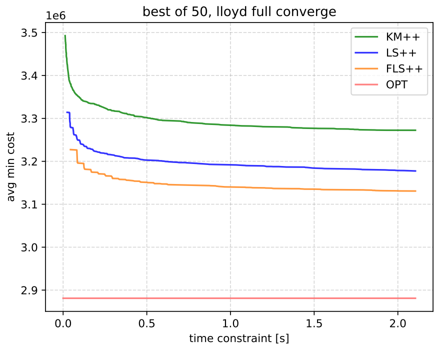
The result of this comparison is shown in Table III.2.3. It shows the numerical values of the final average cost of each algorithm. Additionally we evaluated the relative cost difference when compared to `GFLS++`. For algorithms $A_1, A_2 \in \{\text{KM++}, \text{LS++}, \text{FLS++}\}$ and their respective average costs $c_{A_1}, c_{A_2} \in \mathbb{R}_{>0}$ we define their respective percentage cost difference as $C(A_1, A_2) := (1 - c_{A_1}/c_{A_2}) \cdot 100\%$. On dataset *circles* the cost difference was the highest with approximately 22.37% and 8.8% when we compare the final average cost of `FLS++` to `KM++` and `LS++`. This large difference also shows that even if clusters are well separated, `LS++` might still fail to find an improvement if the optimal center centroids are not close to the actual data points. In contrast, `FLS++` can more often find an improvement because it evaluates the actual optimal centers for a new



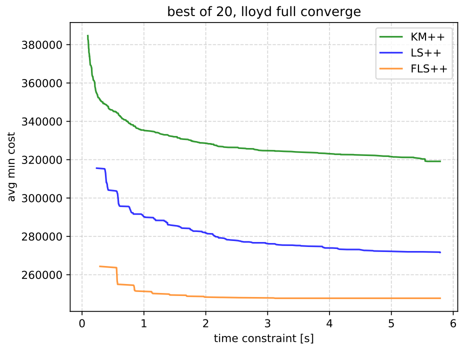
(a) *pr91*, $k = 50$



(b) *bio train features*, $k = 25$



(c) *concrete*, $k = 60$



(d) *circles*, $k = 100$

Figure III.2.5.: Comparison of average cost decrease depending on the runtime of `FLS++`.

choice of centers, which brings the new centers closer to the actual optimal centroids. For the other datasets the relative difference in cost is not large, but a positive amount which, if we take into account the number of times each algorithm returned the smallest cost, can be achieved with large success probability.

Performance over Time. We compare the best solutions found by all three algorithms and their average cost progression over the time. We test this procedure $R \in \mathbb{N}$ times, where the r-th run being defined as:

- Run `GFLS++` B times, let t_B^r be the used time until termination in run r.
- Repeat `GKM++` and `GLS++` each as long as their respective elapsed time is at most t_B^r.

For run $r \in [R]$ let ${}^r c_A^t$ for $A \in \{\text{KM++}, \text{LS++}, \text{FLS++}\}$ be the current minimum cost of algorithm A at time $t \in \mathbb{R}_{\geq 0}$ and run r. Let $t_{\min}$ be the first point in time when A did terminate in every run $r \in [R]$, i.e., ${}^r c_A^t$ is always defined for every time $t \geq t_{\min}$. The average cost of any algorithm A after R runs in time step $t \geq t_{\min}$ is defined as

Dataset	#Wins KM++	#Avg. iterations KM++	#Wins LS++	#Avg. iterations LS++	#Wins FLS++	#Avg iterations FLS++
pr91	5	160.49	8	64.42	87	50
bio train features	2	19.3	6	13.7	12	10
concrete	0	357.02	4	103.22	96	50
circles	0	103.73	0	29.13	94	20

Table III.2.4.: Number of wins and average number of repetitions within the time limit. We only count a win if the respective algorithm uniquely returned the smallest cost.

$\mathrm{AVG}(A,t) := \frac{1}{R}\sum_{r=1}^{R} {}^{r}c_A^t$. As before, each algorithm starts in the r-th run with the same set of centers from the initialization process.

Figure III.2.5 shows the progress of each algorithm over time. For LS++, we use $Z = 25$ as number of repeat steps, as in the original experiments in [77]. We use the same value for FLS++ in the following experiments if not specified otherwise. We repeat Lloyd steps, i.e., computing memberships of all points to their closest center and recomputing the new optimal centers for this clustering, until two consecutive iterations produce center sets $\mathcal{C}_1, \mathcal{C}_2$ with $1 - \frac{\mathtt{kmeans}(\mathcal{C}_2)}{\mathtt{kmeans}(\mathcal{C}_1)} < 0.0001$. All plots are averaged over 100 runs, i.e., $R = 100$, except for (b), which is averaged over 20 runs. In all plots, FLS++ achieves the smallest final average cost and often beats the other algorithms over the entire time frame. Similarly, LS++ is consistently better than KM++.

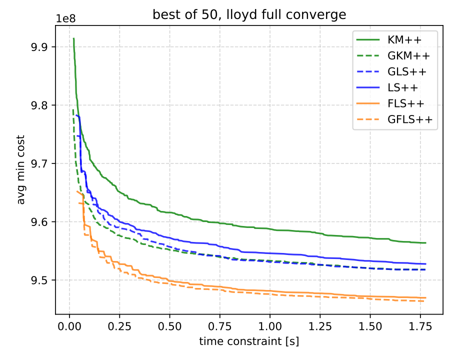
Number of Best Solutions Found Table III.2.4 shows how many times KM++, LS++ or FLS++ computed the overall best solution for a given dataset within the time limit; most of the time, FLS++ returned the smallest solution, despite FLS++ having the fewest average number of iterations due to its longer running time. In the remaining cases, the best solution was mostly found by LS++, while KM++, having the most number of average iterations for each dataset, could only report the smallest solution twice, namely in the large dataset *bio train features*.

III.2.2.3. Greedy vs. Non-Greedy D^2-Sampling

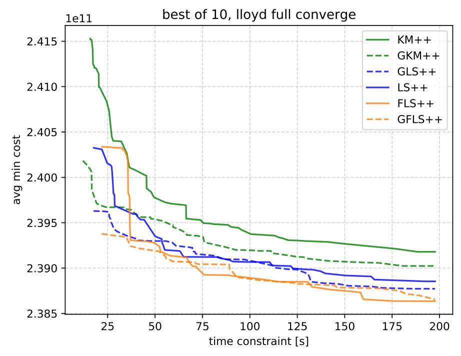
Although greedy D^2-sampling yields a worse theoretical approximation ratio (cf. III.1.3.1 and [25]), it is commonly used in practice. In the scikit-learn library [87], it is used as the default initialization method with parameter $\ell = 2 + \lfloor \ln(k) \rfloor$. In this section, we investigate whether the practical choice is justified and how LS++/FLS++ changes when using greedy sampling.

In Figure III.2.6 and III.2.7, we compare the average performance of all algorithms when using greedy D^2-sampling vs. standard D^2-sampling. We also used $\ell = 2 + \lceil \ln k \rceil$. Here, on almost all evaluated datasets, the average cost decrease is larger when using greedy D^2-sampling, regardless of which algorithm is chosen afterwards. Especially KM++ benefits from using greedy D^2-sampling. LS++ and FLS++ benefit from greedy D^2-sampling most of the time, and by roughly the same amount.

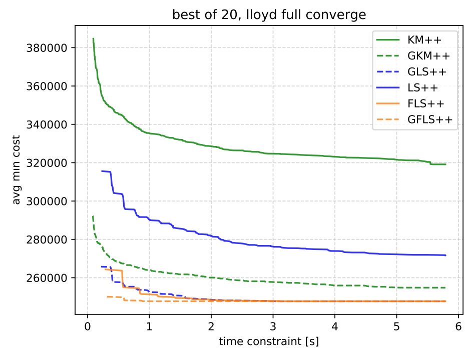
We also perform experiments with a given time limit for all algorithms to see whether a higher number of restarts can create an advantage for the other algorithms. In this



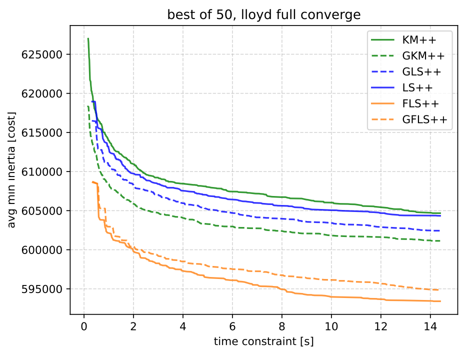
(a) *pr91*, $k = 50$



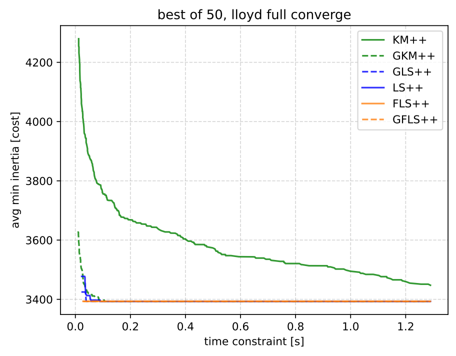
(b) *bio train features*, $k = 25$



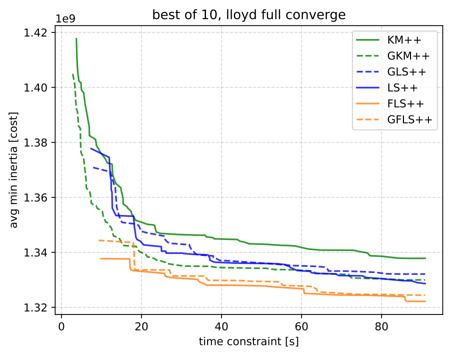
(c) *circles*, $k = 100$



(d) *close circles*, $k = 60$



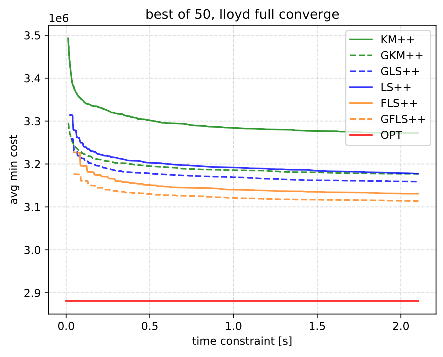
(e) *D31*, $k = 31$



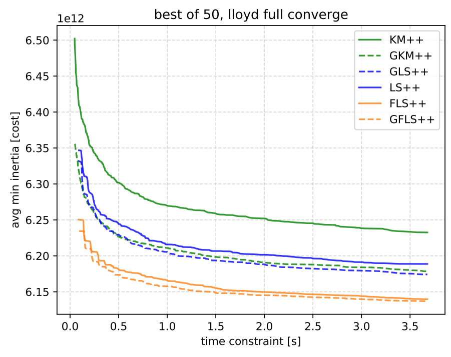
(f) *frymire*, $k = 20$

Figure III.2.6.: Comparison of average cost decrease depending on the runtime of `FLS++`. Dotted lines correspond to the greedy variant of the original algorithm.

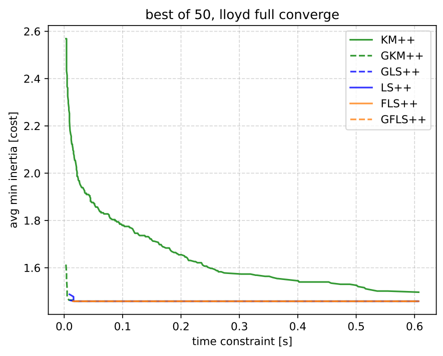
case we use the time limit given by `FLS++` for $B = 25$ for all algorithms. We also fixed $Z = 25$ for `LS++` and `GLS++` but varied the number of local search steps for `FLS++` and `GFLS++` as $Z \in \{5, 10, 15, 20\}$. Even with fewer local search steps in `FLS++` or `GFLS++` we



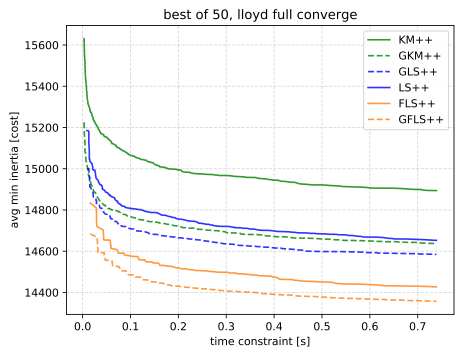
(a) *concrete*, $k = 60$



(b) *s3*, $k = 15$



(c) *rectangles*, $k = 36$



(d) *body measurements*, $k = 50$

Figure III.2.7.: Comparison of average cost decrease depending on the runtime of `FLS++`. Dotted lines corresponds to the greedy variant of the original algorithm.

get a smaller cost compared to `LS++` or `GLS++` on average. For two considered datasets *concrete* and *pr91*, choosing $Z = 15$ is always enough to get approximately the same cost. For *pr91*, even choosing $Z = 5$ results in a better solution than `LS++` or `GLS++` on average for the specified time limit.

Now we want to consider how using greedy D^2-sampling improves the average costs after reaching the time limits.

For $A \in \{\texttt{FLS++}, \texttt{LS++}, \texttt{KM++}\}$ we define their average costs c_A, c_A^G where the superscript G indicates if we use greedy d^2-sampling. In our case we compare both average costs over all runs $r \in [R]$ when using time limit t_B^r. Lastly, similar to the analysis above, we define the improvement factor as $(1 - \frac{c_A^G}{c_A}) \cdot 100\%$. As we can see in Table III.2.5, for most datasets and algorithms the improvement is not below 0% and most of the time larger than 0%.

At last, we analyze the number of times each algorithm did report the strictly best solution, the number of iterations, average costs and cost factor when compared to GFLS++. Like in the case when using normal D^2-sampling we can see in Tables III.2.6

Dataset	$C(\text{GKM++}, \text{KM++})$	$C(\text{GLS++}, \text{LS++})$	$C(\text{GFLS++}, \text{FLS++})$
pr91	0.48%	0.1%	0.06%
bio train features	0.07%	0.03%	−0.01%
concrete	2.91%	0.58%	0.54%
circles	20.16%	8.8%	0%
close circles	0.59%	0.32%	−0.24%
D31	1.58%	0%	0%
frymire	0.59%	−0.26%	−0.17%
rectangles	2.56%	0%	0%
s3	0.87%	0.23%	0.04%
Body measurements	1.72%	0.46%	0.49%

Table III.2.5.: Average cost decrease by using *greedy D^2-sampling*.

and III.2.7 that GFLS++ still on average computes the solution with the smallest cost and does so with large probability. In case of the dataset *circles*, now GLS++ and GFLS++ computed the optimal solution after the time limit in every round. But we can see in Figure III.2.7 that GFLS++ does so faster than GLS++.

Dataset	#Wins GKM++	#Avg. repetitions GKM++	#Wins GLS++	#Avg. iterations GLS++	#Wins GFLS++	#Avg. repetitions GFLS++
pr91	5	160.39	8	62.4	87	48.14
bio train features	2	20.9	6	12.95	12	10.1
concrete	0	295.71	0	95.44	100	48.25
circles	0	109.96	0	29.33	0	19.86
close circles	3	127.14	2	55.47	95	48.46
D31	0	250.55	1	69.75	0	47.3
frymire	5	41.15	2	13.55	13	10
rectangles	0	442.31	0	83.12	0	48.12
s3	4	131	6	57.38	89	48.38
Body measurements	0	336.77	0	101.81	100	47.72

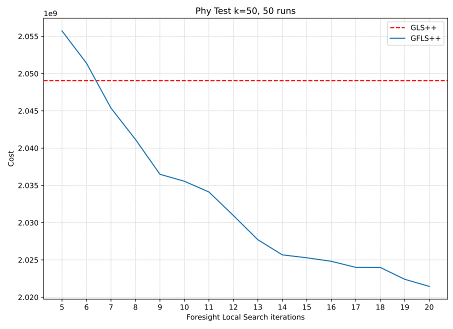
Table III.2.6.: Number of wins and average number of iterations, corresponding to the results shown in Figure III.2.6.

Dataset	$c_{\text{GKM++}}$	$c_{\text{GLS++}}$	$c_{\text{GFLS++}}$	$C(\text{GFLS++}, \text{GKM++})$	$C(\text{GFLS++}, \text{GLS++})$
pr91	9.518E+08	9.5177E+08	9.4638E+08	0.57%	0.57%
bio train features	2.3902E+11	2.3877E+11	2.3866E+11	0.15%	0.05%
concrete	3.1772E+06	3.1591E+06	3.1138E+06	2%	1.43%
circles	2.5481E+05	2.4773E+05	2.4773E+05	2.78%	0%
close circles	6.0113E+05	6.0244E+05	5.9485E+05	1.04%	1.26%
D31	3393.26	3393.26	3393.26	0%	−0%
frymire	1.33E+09	1.3321E+09	1.3244E+09	0.42%	0.58%
rectangles	1.46	1.46	1.46	0%	0%
s3	6.1786E+12	6.1742E+12	6.137E+12	0.67%	0.6%
Body measurements	14637.51	14584.68	14357.35	1.91%	1.56%

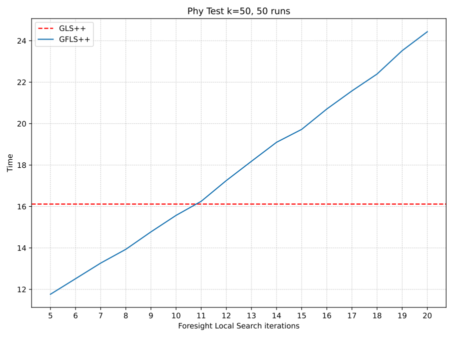
Table III.2.7.: Respective average cost of each algorithm. Last two columns show $(1 - c_{\text{GFLS++}}/c_A) \cdot 100\%$ for $A \in \{\text{GKM++}, \text{GLS++}\}$.

III.2.2.4. Varying the Number of Local Search Steps

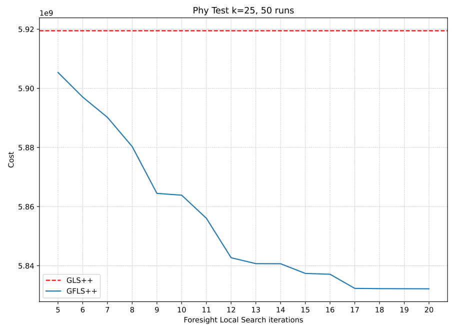
Running `LS++` and `FLS++` with the same number of local search steps expectedly leads to `FLS++` performing better in terms of cost, but worse in terms of runtime. But Figures III.2.3 and III.2.4 already hint at the possibility that using fewer local search steps in FLS++ might lead to better cost *and* better runtime. To this end, we fix the number of local search iterations of LS++ to 25 (guided by the experiments in [77]), while trying out different values for FLS++. More precisely, we let the algorithms perform 50 runs in total, where one run consists of the following calls. First, we call LS++ with $Z = 25$, and afterwards FLS++ for all $Z \in \{5, \ldots, 20\}$. Importantly, we again ensure that each call in the run starts with the same set of initial centers. For both algorithms, we let Lloyd's



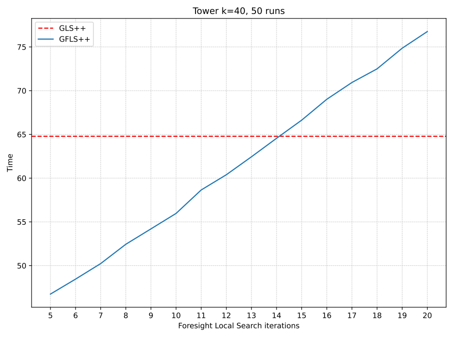
(a) *KDD Phy Test* Cost, $k = 50$



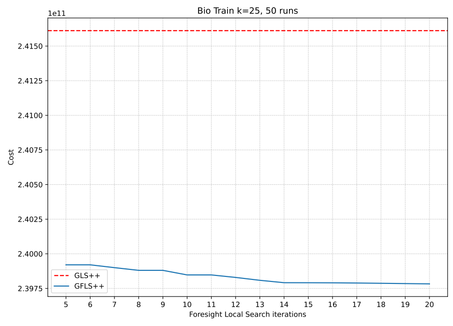
(b) *KDD Phy Test* Time, $k = 50$



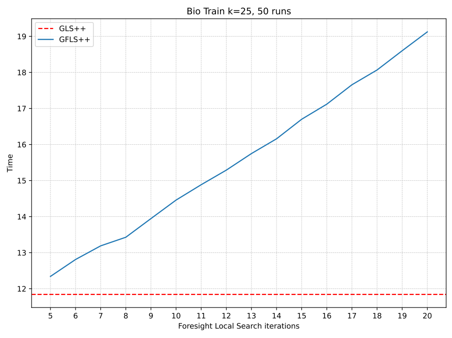
(c) *KDD Phy Test* Cost, $k = 25$



(d) *KDD Phy Test* Time, $k = 25$



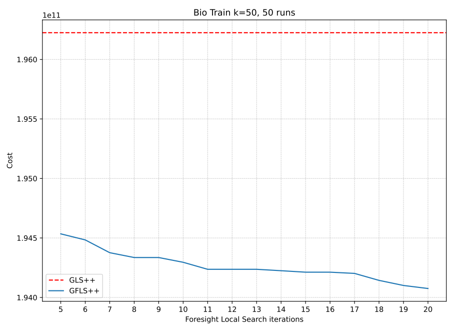
(e) *KDD Bio Train* Cost, $k = 25$

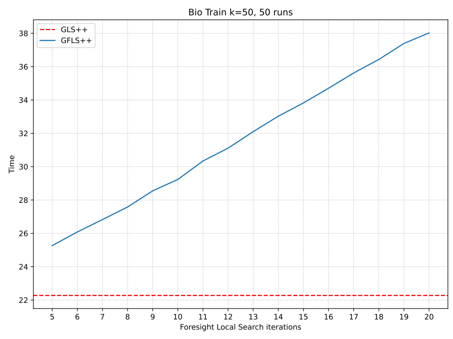


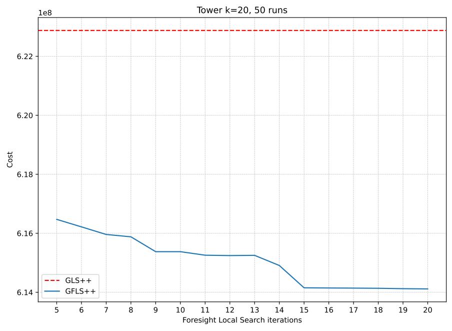
(f) *KDD Bio Train* Time, $k = 25$

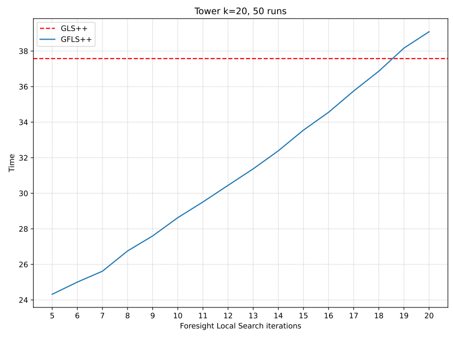
Figure III.2.8.: Impact on cost and time of increasing the number of local search steps in FLS++. The dashed red line shows average cost (resp. time) of 50 runs of LS++ with 25 local search steps.

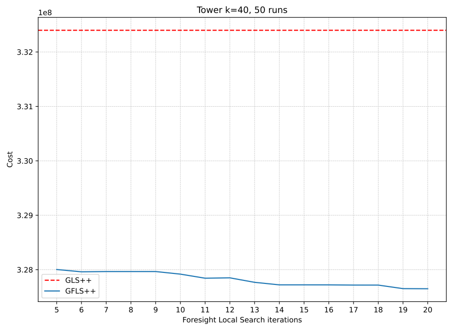
Algorithm run until convergence after performing the local search. After completing all 50 runs, we compute the average cost and runtime of the `LS++` calls across all runs. For `FLS++`, we compute the averages for every value of Z over all calls separately.


(a) *KDD Bio Train* Cost, $k = 50$


(b) *KDD Bio Train* Time, $k = 50$


(c) *Tower* Cost, $k = 20$


(d) *Tower* Time, $k = 20$


(e) *Tower* Cost, $k = 40$

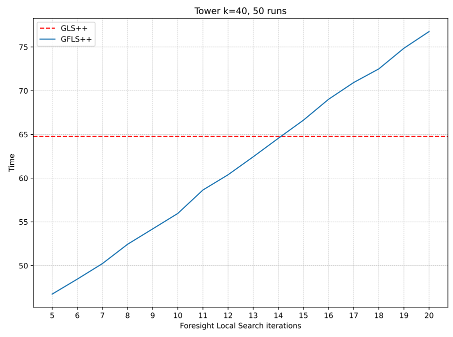

(f) *Tower* Time, $k = 40$

Figure III.2.9.: Impact on cost and time of increasing the number of local search steps in `FLS++`. The dashed red line shows average cost (resp. time) of 50 runs of `LS++` with 25 local search steps.

The results of running these experiments on datasets *KDD Phy Test, KDD Bio Train* and *Tower* for two different values of k each are shown in Figures III.2.8 and III.2.9. The red line indicates the average cost (resp. time) of `LS++` with $Z = 25$ over all 50 runs. It should be emphasized that the blue curves do *not* depict the temporal evolution of a single execution of `FLS++`. Instead, each point on a curve represents the average cost (resp. runtime) over 50 runs for a specific value of Z. Consequently, the cost curves are not always strictly decreasing, even though they are decreasing within an individual run of the algorithm, as the cost can only decrease between two consecutive local search steps. We see that for `FLS++`, even very small values of Z already suffice to beat `LS++` in terms of average cost. In fact, `FLS++` already produces a better solution for $Z = 5$ than `LS++` for $Z = 25$ in most cases. The only exception is *KDD Phy Test* with $k = 50$, where the solution becomes better at $Z = 7$. The runtime of `FLS++` expectedly grows linearly with Z. On *KDD Phy Test* and *Tower*, the runtime for small values of Z stays below that of `FLS++`, only overtaking it at $Z = 11$ and $Z = 15$ for *KDD Phy Test* with $k = 50$ and $k = 25$, and at $Z = 19$ and $Z = 15$ for *Tower* with $k = 20$ and $k = 40$. On *KDD Bio Train*, the runtime is larger than that of `LS++` even for $Z = 5$, but not by much. So choosing $Z = 5$ gives a better solution with a comparable runtime.

III.2.2.5. Instances with Known Optimal Solutions

After seeing that `FLS++` outperforms the other algorithms in terms of cost, a natural question is to ask how well it performs with respect to the optimum solution. As k-Means is NP-hard, obtaining optimum solution values, even for medium-sized instances, is practically infeasible. There exist some solvers that can handle instances of a few hundred up to 4000 points in two dimensions (e.g. [9], [89]). We use MSSC-DCA, introduced by Sudoso and Aloise [89], to obtain the actual optimum assignments in addition to the cost for some of the instances and for selected values of k (cf. Table A.1).

First, we compare `FLS++` with $Z = 25$ to `LS++` (also with $Z = 25$), `k-means++` and `Greedy k-means++`. We use a similar experiment structure as before, repeating every algorithm on every dataset for every k for 100 runs, ensuring that the initialized center set is identical for all four algorithms within one run. Finally, we let Lloyd's algorithm run until convergence. Then we check whether the algorithms were able to find the optimum solution. If c_A denotes of the best solution found by algorithm A over 100 runs for a specific value of k, and c_{opt} the cost of an optimal solution for that k, we let $\alpha_k = c_A/c_{\text{opt}}$ be the approximation factor of that best solution. So if the optimum was found, we have $\alpha_k = 1$. Additionally, we count how often the returned solutions were within a certain approximation factor τ. This is especially interesting if no algorithm was able to find the exact optimum. By default, we use $\tau = 0.001$.

Small Values of k Table III.2.8 shows the results of these experiments on datasets *666cities*, *pr91* and *fl417* for small values of k. On the first two datasets, we see an apparent increase in difficulty for the algorithms with increasing k. For $k = 4$, all four algorithms were able to find the optimum on both datasets. On *pr91*, `FLS++` found a solution within a factor of 1.001 every run, and the other algorithms almost every run. On *666cities*, `FLS++` was able to return such a solution in 50 out of 100 runs. Interestingly, on *666cities* all algorithms performed better for $k = 6$ than for $k = 4$,

Dataset	k		Algorithms			
			`KM++`	`GKM++`	`LS++` 25	`FLS++` 25
666cities	4	α_k	1	1	1	1
		$\mathbf{N}_\tau$	15	41	43	50
	6	α_k	1	1	1	1
		$\mathbf{N}_\tau$	33	52	72	100
	10	α_k	1.00006	1.00009	1.00006	1.00009
		$\mathbf{N}_\tau$	3	2	2	6
pr91	4	α_k	1	1	1	1
		$\mathbf{N}_\tau$	98	96	99	100
	8	α_k	1.00003	1.0000004	1.0000004	1
		$\mathbf{N}_\tau$	2	5	15	17
	10	α_k	1.00001	1.00003	1.000003	1.0000008
		$\mathbf{N}_\tau$	4	4	14	10
fl417	16	α_k	1	1	1	1
		$\mathbf{N}_\tau$	1	3	16	75

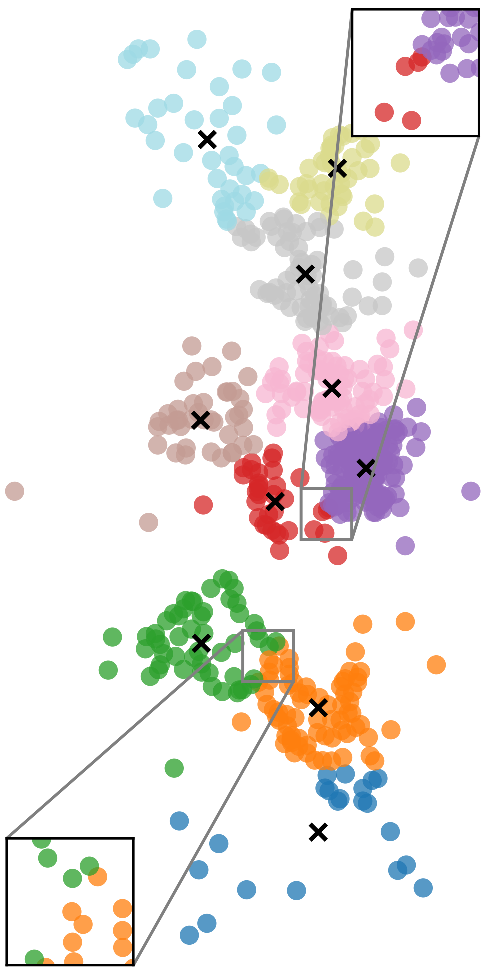
Table III.2.8.: Results for *666cities*, *pr91* and *fl417* over 100 runs for $\tau = 0.001$. $\alpha_k = 1$ means that the optimal solution was found at least once. N_τ indicates the number of runs that returned a solution within $(1 + \tau)\textsc{Opt}$.

while on *pr91* the jump from $k = 4$ to $k = 8$ caused a significantly weaker performance. Only `FLS++` was able to find the optimum solution in that case. However, the best found solutions for all algorithms are very close to the actual optimum. For $k = 10$ none of the algorithms was able to find the exact optimum solution on either dataset. In fact, the number of approximately optimal solutions found is quite low across all algorithms. On the other hand, the best found solutions are very close to the optimum. Figure III.2.10 illustrates how similar the solutions are.

Only two points are assigned differently. So even if the optimum was not found, the best found solution is almost identical. A reason why the algorithms fall just short of finding the global optimum might be the fact that the optimum solution is not very well separated. The cluster borders often run among very densely populated regions, making the optimum solution rather “ambiguous”.

Figures III.2.11a and III.2.11b show the corresponding boxplots for $k = 10$ on both datasets. We see that, as indicated by the values in Table III.2.8, all algorithms are able to find optimum or near-optimum solutions. Staying mainly in line with the experiments seen so far, we can see that `Greedy k-means++` performs better than `k-means++`, `LS++` performs better than `Greedy k-means++`, and `FLS++` performs better than `LS++`. Mean and median values as well as variance decrease from left to right.

Dataset *fl417*, for which we only have an optimal solution for $k = 16$, seems to be less problematic. All four algorithms were able to find the optimum solution within 100


(a) Best solution found by `LS++`.

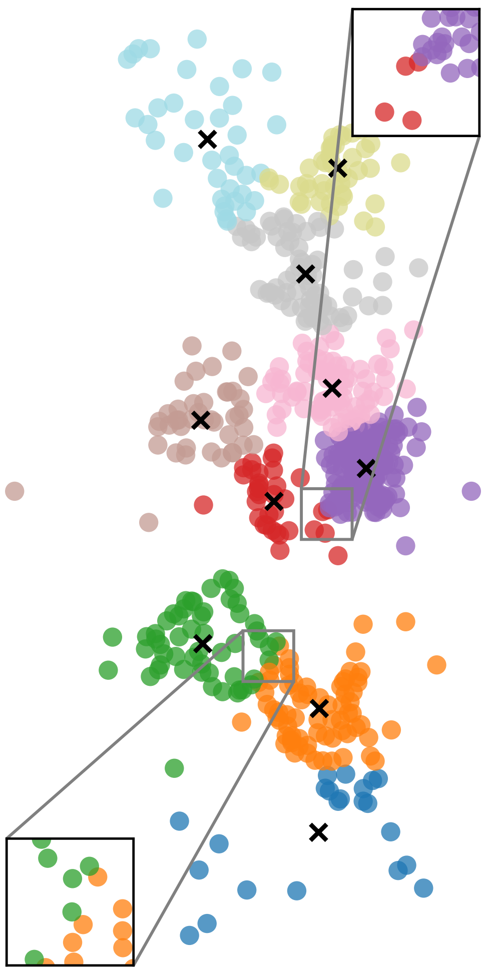
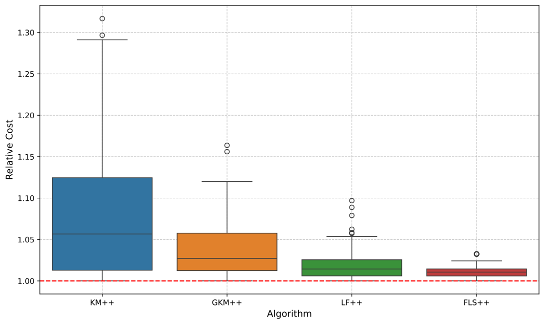

(b) The optimum solution.

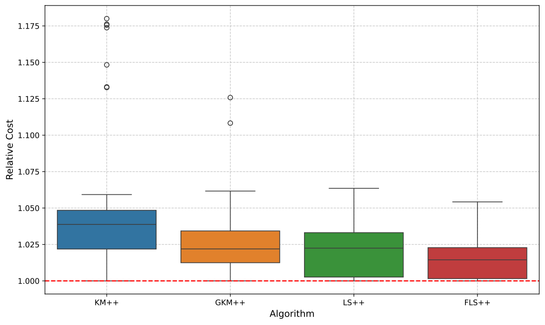
Figure III.2.10.: A comparison of the best solution returned by any of the 4 algorithms with the global optimum for *666cities* and $k = 10$. The two are almost indistinguishable. Only two points are assigned differently.

runs, with non-greedy `k-means++` only finding it once. Furthermore, 75 out of 100 runs of `FLS++` returned a solution within 0.1% of the optimum. The corresponding boxplots in Figure III.2.11c also show the expected behavior of all four algorithms. The worst solution of `FLS++` is still only off by roughly 0.3%.

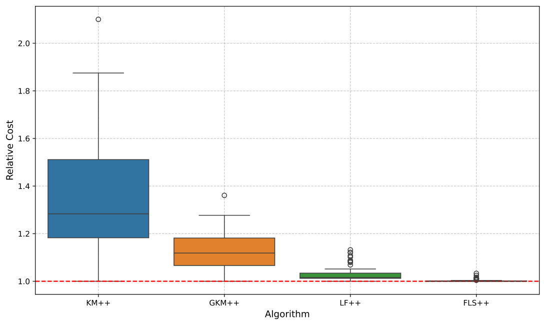
Larger Values of k As it turns out, the solvers that compute optimum solutions are good at handling rather small values of k (e.g. under 10) or rather big values of k (e.g. 100 and above), but struggle to compute optimum solutions for intermediate k in



(a) *666cities*, $k = 10$.



(b) *pr91*, $k = 10$.



(c) *fl417*, $k = 16$.

Figure III.2.11.: Relative cost comparison.

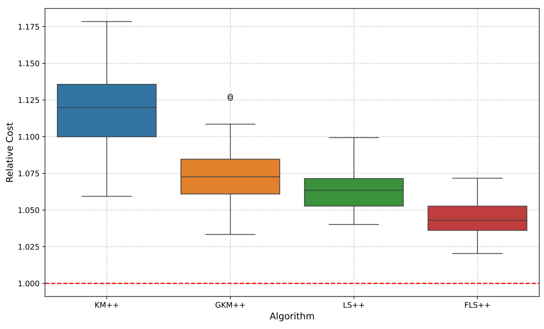
reasonable time (and memory). Therefore, we have a rather large gap in the sequence of k values that we consider. For *pr91* we were able to get optimum assignments for $k \in \{100, 200, 250, 300\}$, for *drilling* and *u2152* with $k \in \{100, 200\}$, for *fl3975* with $k = 300$, and for *rl5934* with $k = 100$. We therefore also let the four algorithms run with these configurations. Table III.2.9 shows the approximation factors of the best found solutions. The trend that was already hinted at in the experiments for smaller k continues here. None of the algorithms was able to find a solution within 0.1% of the optimum. In fact, almost all solutions are off by more than 2%, with *rl594* being the exception. Here, `GKM++` and `LS++` were roughly within 1.38% and 1.48%, while `LS++` was within 0.65%. Overall, `FLS++` performed the best on all datasets.

The boxplots for these experiments are depicted in Figures III.2.12, III.2.13, III.2.14 and III.2.19. On the datasets where we tried multiple values of k, we can observe that larger k leads to worse performance by all algorithms. For *pr91* with $k = 300$ and *drilling* with $k = 200$, the best found solution by `FLS++` is off by more than 5%. On dataset *rl5934*, the algorithms achieve the best results.

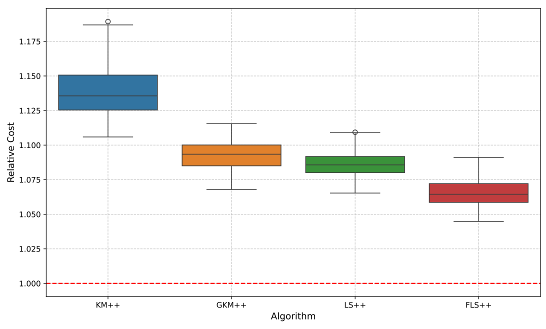
Dataset	k		Algorithms			
			`KM++`	`GKM++`	`LS++`	`FLS++ 25`
fl3795	100	α_k	1.1072	1.0559	1.0519	1.039
		$\mathbf{N}_\tau$	0	0	0	0
pr91	100	α_k	1.0594	1.0333	1.0402	1.0204
		$\mathbf{N}_\tau$	0	0	0	0
	200	α_k	1.1059	1.0679	1.0653	1.0448
		$\mathbf{N}_\tau$	0	0	0	0
	250	α_k	1.1273	1.0897	1.0850	1.0562
		$\mathbf{N}_\tau$	0	0	0	0
	300	α_k	1.1633	1.0777	1.0799	1.0555
		$\mathbf{N}_\tau$	0	0	0	0

Dataset	k		Algorithms			
			`KM++`	`GKM++`	`LS++`	`FLS++ 25`
rl5934	100	α_k	1.0226	1.0138	1.0148	1.0065
		$\mathbf{N}_\tau$	0	0	0	0
drilling	100	α_k	1.0956	1.0518	1.0499	1.0254
		$\mathbf{N}_\tau$	0	0	0	0
	200	α_k	1.1947	1.1042	1.0886	1.0589
		$\mathbf{N}_\tau$	0	0	0	0
u2152	100	α_k	1.0545	1.0347	1.038	1.0252
		$\mathbf{N}_\tau$	0	0	0	0
	200	α_k	1.1095	1.0641	1.0633	1.0434
		$\mathbf{N}_\tau$	0	0	0	0

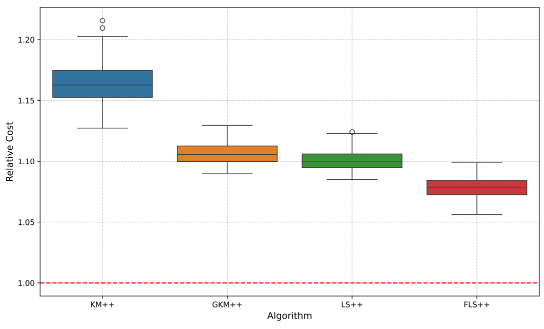
Table III.2.9.: Results for several datasets for $\tau = 0.001$. None of the algorithms found an optimal solution, or even one that is within 0.1% of the optimum.



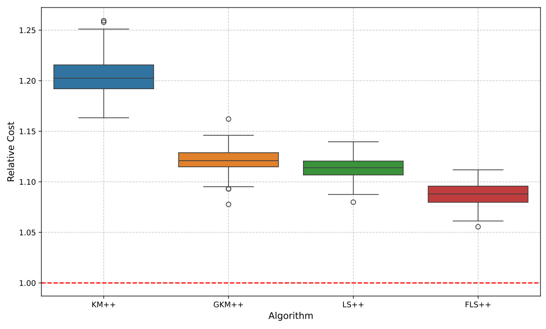
(a) $k = 100$



(b) $k = 200$

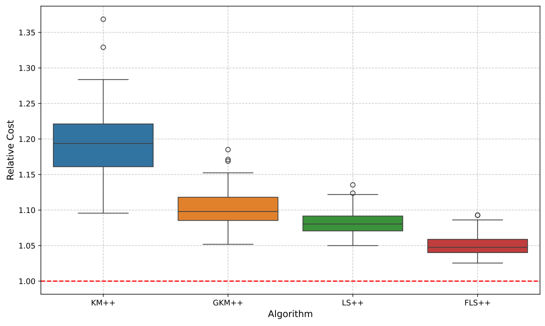


(c) $k = 250$

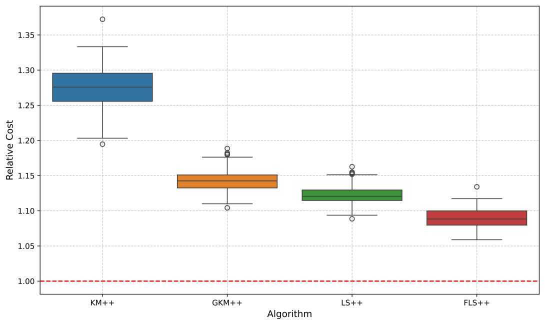


(d) $k = 300$

Figure III.2.12.: Relative cost comparison on dataset *pr91* for larger values of k.

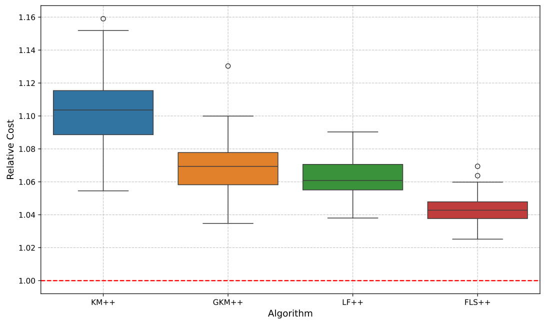


(a) $k = 100$

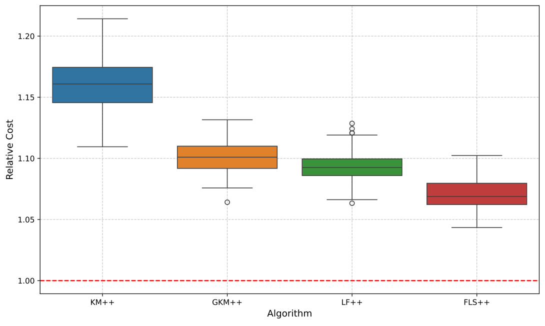


(b) $k = 200$

Figure III.2.13.: Relative cost comparison on dataset *drilling* for larger values of k.

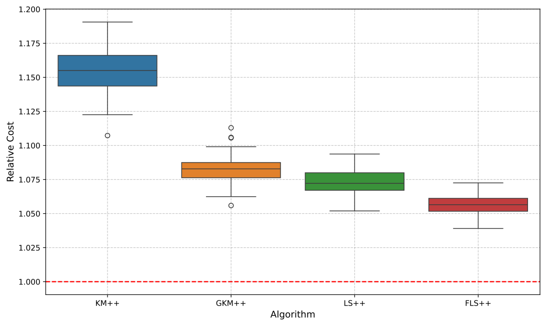


(a) $k = 100$

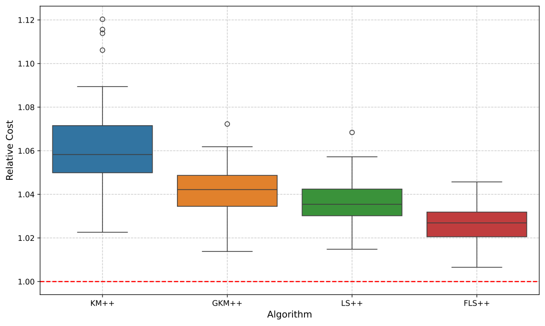


(b) $k = 200$

Figure III.2.14.: Relative cost comparison on dataset *u2152* for larger values of k.



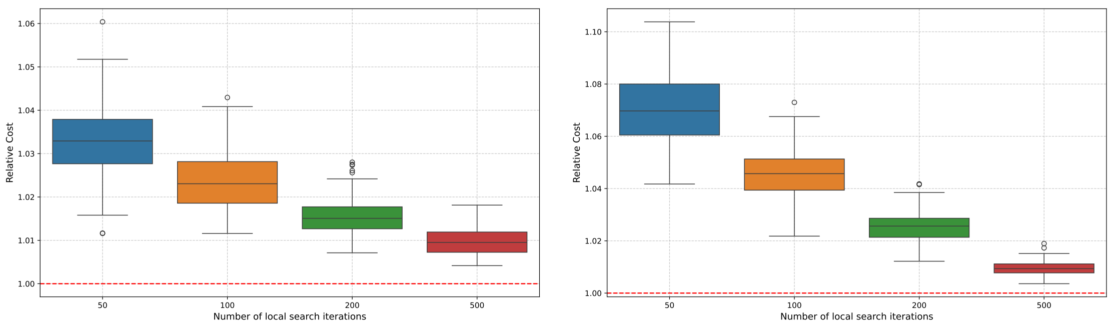
(a) *fl3795*, $k = 300$



(b) *rl5934*, $k = 100$

Figure III.2.15.: Relative cost comparison on datasets *fl3795* and *rl5934* for larger values of k.

Increasing Z to Approach the Optimum In section III.2.2.4, we already saw that, unsurprisingly, more local search iterations lead to better solutions. This immediately raises the question whether performing more such iterations can get us closer to the optimum solution. To this end, we performed experiments similar to those in section III.2.2.4 for $Z \in \{50, 100, 200, 500\}$. Figures III.2.16 and III.2.17 show the corresponding boxplots for datasets *drilling* and *pr91*.



(a) $k = 100$ (b) $k = 200$

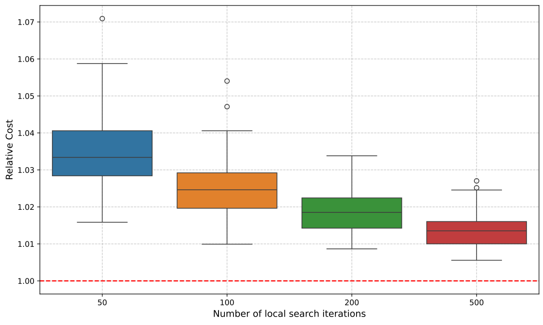
Figure III.2.16.: Cost of `FLS++` relative to the optimum on dataset *drilling* for $Z \in \{50, 100, 200, 500\}$.

We see that with larger Z the solutions indeed get better. Not only do the best and median solutions improve, but also the variance gets visibly smaller. On the *drilling* dataset, the best found solutions for $Z = 500$ are well below a factor of 1.01 compared to the optimum. But even here it does not get as close as 1.001.

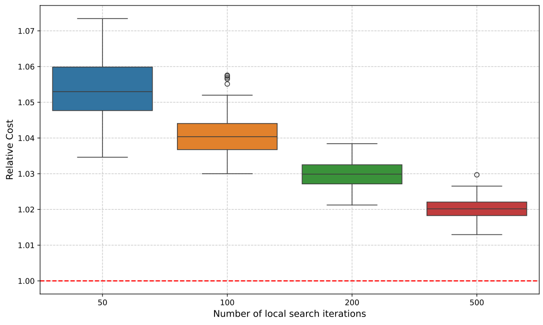
Dataset *pr91* seems to pose even more of a problem. For $Z = 50$ the best found solution is almost 5% off, and even for $Z = 500$ it does not get below 1%.

However, even for $Z = 500$, `FLS++` is not able to find an optimum solution or even a solution within the tolerance of 0.1%. Instead, the best found solution can still be off by more than 1%, as can be seen in Table III.2.10. Finally, we were also able to obtain an optimal assignment for the dataset *fl3795* with $k = 300$ and for *rl5934* with $k = 100$. As can be seen in Figure III.2.19 (a), *fl3795* seems to be hard to approximate as well. While the variance and the worst found solutions are not as bad as e.g. for *pr91* with $k = 100$, the best found solution is off by almost 2%. *rl5934* on the other hand is the only dataset where we actually do find an optimum solution for a large value of k.

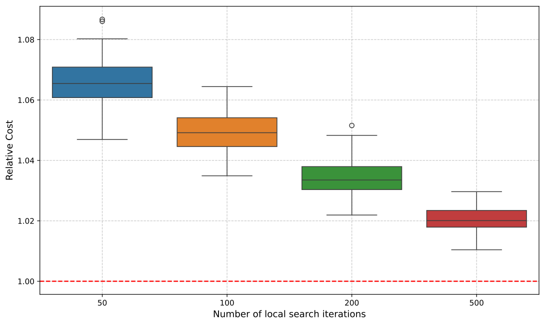
Increasing Z Even Further As there are still visible improvements when increasing Z from 200 to 500, we might want to take a look at even larger values of Z and see whether the optimum can be found. Since k is rather large, even on these relatively small datasets, performing tens or even hundreds of thousands of local search iterations takes considerable time. So instead of performing multiple repeats, we identify the runs that yielded the best result for $Z = 500$ and simply let local search run for longer with that same seed.



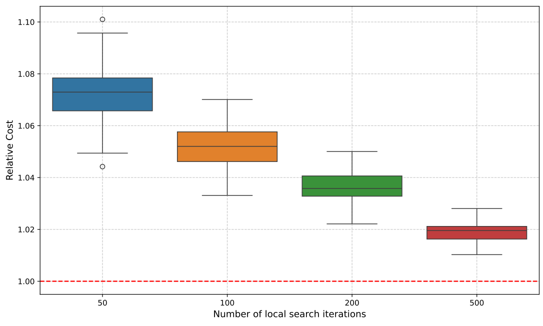
(a) $k = 100$



(b) $k = 200$

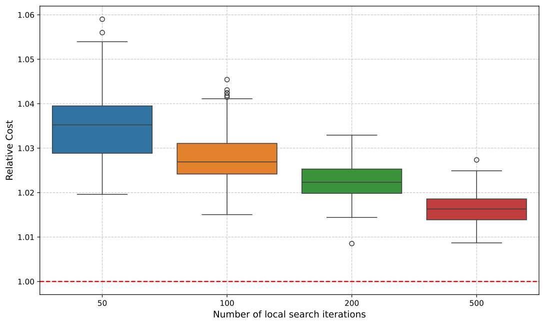


(c) $k = 250$

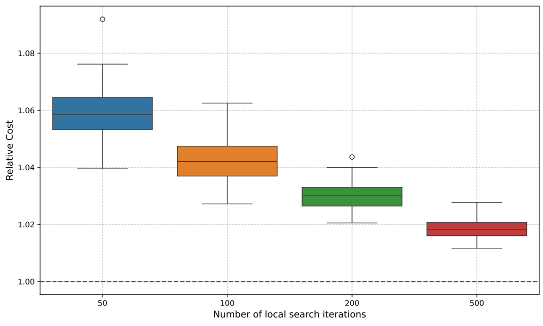


(d) $k = 300$

Figure III.2.17.: Cost of `FLS++` relative to the optimum on dataset *pr91* for $Z \in \{50, 100, 200, 500\}$.



(a) $k = 100$



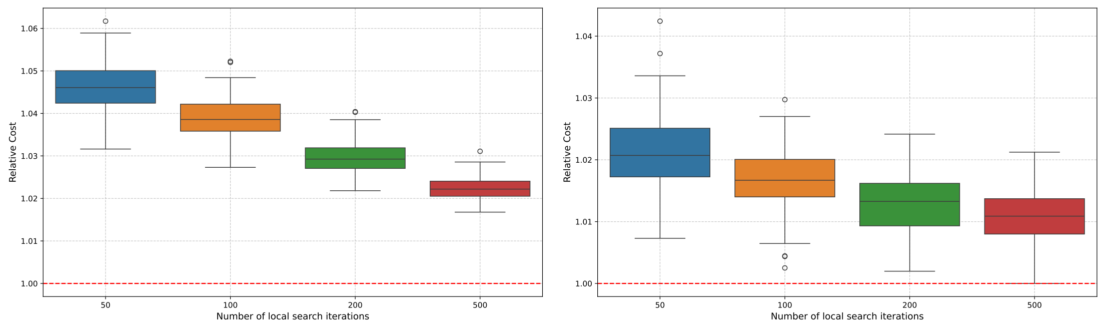
(b) $k = 200$

Figure III.2.18.: Cost of `FLS++` relative to the optimum on dataset *u2152* for $Z \in \{50, 100, 200, 500\}$.

We performed these experiments on *pr91, u2152, drilling* with $k = 100$ and $k = 200$, and on *fl3795* with $k = 300$. For every dataset and value of k, we let the algorithm run with $Z \in \{500, 10\,000, 50\,000, 100\,000, 500\,000\}$, except for *fl3795*, where $Z = 500\,000$ was omitted. Thus, the following plots do not show averages over multiple runs. Instead, they

Dataset	k	Approximation Factor
pr91	100	1.005578
	200	1.012953
	250	1.010423
	300	1.010264
drilling	100	1.004164
	200	1.003590
u2152	100	1.008692
	200	1.011664
fl3795	300	1.016777
fl5934	100	1

Table III.2.10.: Approximation factors of best found solutions after 500 local search iterations.



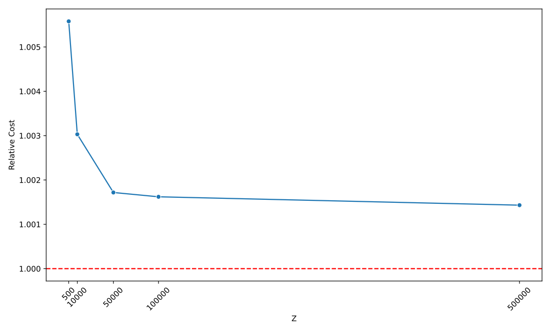
(a) *fl3795*, $k = 300$ (b) *rl5934*, $k = 100$

Figure III.2.19.: Relative cost development for increasing Z on two more datasets.

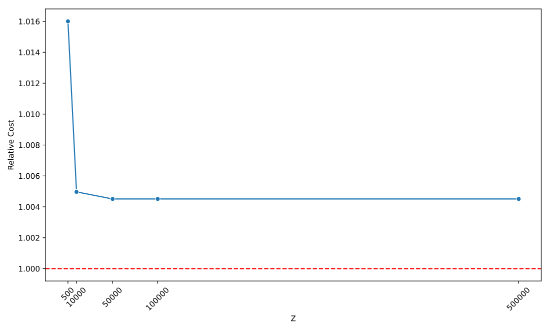
show for each value if Z the (relative) cost of the solution returned after Z local search iterations and then letting Lloyd's algorithm run until convergence. In Figure III.2.20, we see the results for *pr91* and *u2152* with $k = 100$ and $k = 200$. In all cases, we see similar behavior. Between 500 and 10 000 local search steps, there is a steep decline, which significantly flattens out between 10 000 and 50 000. Above 50 000 local search steps, there is barely any improvement at all; in the case of *u2152* and $k = 200$ there is none. Still, the solution does not get within 0.1% of the optimum in any of the cases. It does get quite close for *pr91* with $k = 100$ and *u2152* with $k = 200$, while for *pr91* with $k = 200$ it stagnates at around 0.4%.

In Figure III.2.21, the results for *drilling* with $k = 100$ and $k = 200$ as well as *fl3795* with $k = 300$ are shown. On *drilling*, the overall image remains the same: after a sharp initial drop in the first 10 000 steps, there is not a lot of improvement in the remaining

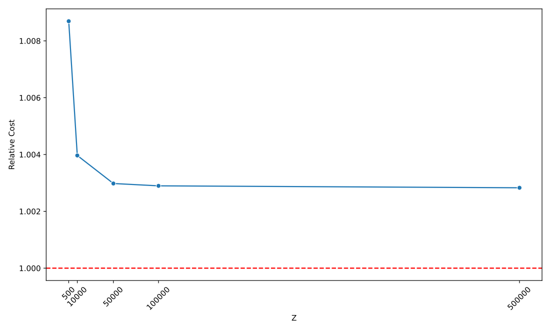
490000 steps. For $k = 100$, the cost actually drops below the 0.1% threshold. For *fl3795* and $k = 300$, running the algorithm takes considerably more time, which is why, for this dataset, the maximum value of Z is 100 000 instead of 500 000. We start with a solution of around $1.017 \cdot \textsc{Opt}$, but here we do not see any improvement between 500 and 10 000 steps. After 50 000 steps, we see a decrease to around $1.0075 \cdot \textsc{Opt}$, but no further improvement is made after 100 000 steps.



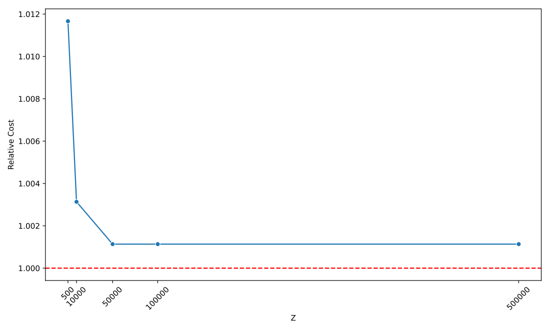
(a) *pr91*, $k = 100$



(b) *pr91*, $k = 200$



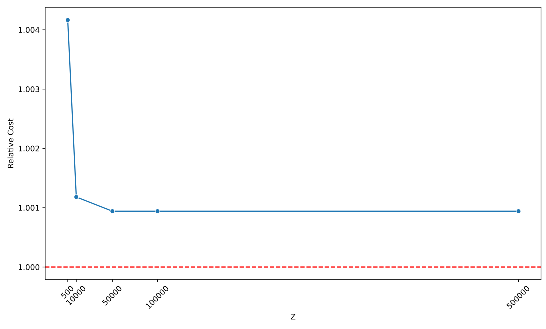
(c) *u2152*, $k = 100$



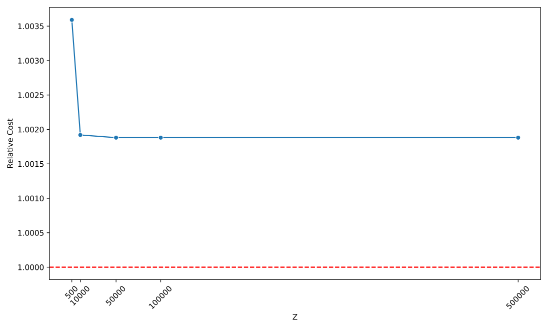
(d) *u2152*, $k = 200$

Figure III.2.20.: Improvement of (relative) cost for even larger values of Z. Above 50 000 local search steps, the improvements are negligible.

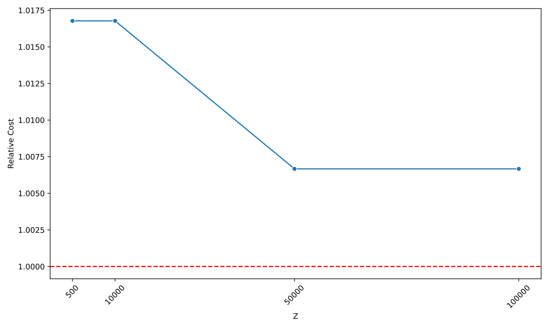
In conclusion, increasing Z beyond 500 seems to be of little value, as the improvements in cost do not outweigh the significant increase in runtime. The runtime scales linearly with Z, so even for the datasets where there is a visible improvement between $Z = 500$ and $Z = 10\,000$, one has to keep in mind that this comes at the price of essentially increasing the runtime by a factor of 20. In any case, actually recovering the optimum solution was not possible. In the next section, we will explore some ideas as to why this might be the case.



(a) *drilling*, $k = 100$



(b) *drilling*, $k = 200$



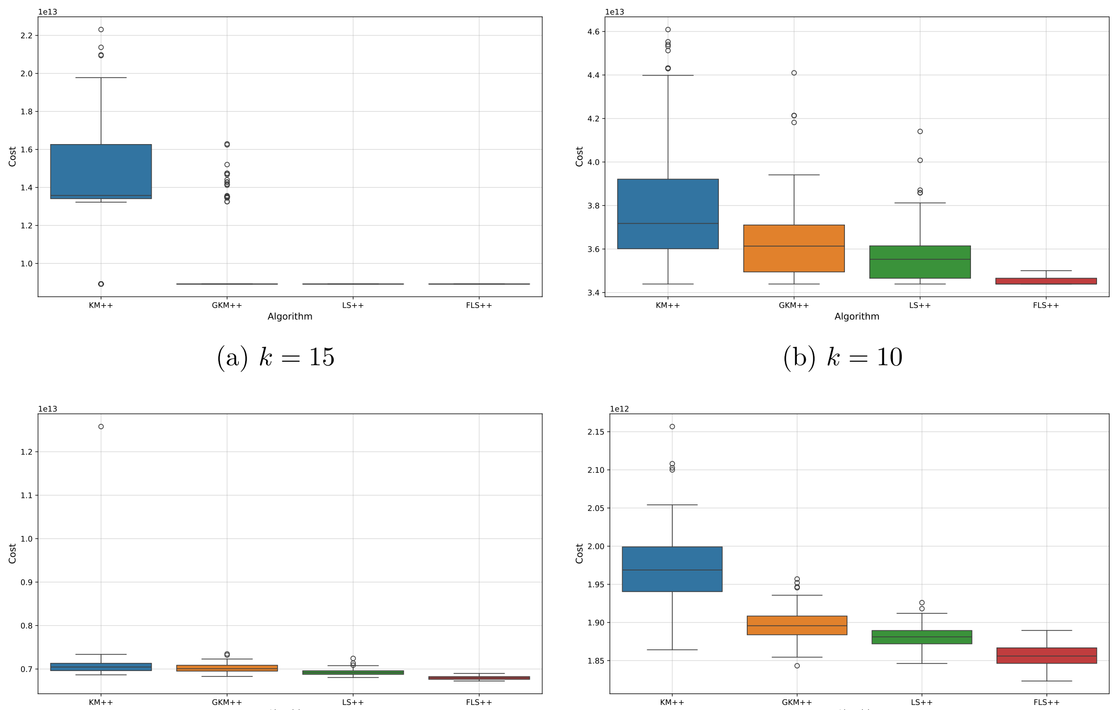
(c) *fl3795*, $k = 300$

Figure III.2.21.: Improvement of (relative) cost for even larger values of Z. Above 50 000 local search iterations the improvements are negligible.

III.2.3. Clusterability of Datasets

As we have seen in the results of our experiments, there are datasets and choices of k where all algorithms are struggling to get close to the optimum solution, let alone find it exactly. When minimizing k-`Means` on a given dataset, we make implicit assumptions on the structural properties of said dataset. As was outlined in Chapter III.1, minimizing k-`Means` can be interpreted as MLE for a special case of Gaussian mixture models. We therefore implicitly assume that the data consists of k roughly spherical and well-separated clusters. If we take a look at some of our datasets, it becomes clear that this assumption is flawed. Taking *u2152*, *fl3795* or *rl5934* as examples, we see that the points form many elongated chains rather than convex, spherical clusters. Other clustering algorithms (e.g. SingleLinkage, DBSCAN) might be able to better recover these structures. Additionally, the number of clusters we chose is in no way reflected in the actual structure of the dataset. This leads to optimum solutions being rather ambiguous, and therefore possibly harder to retrieve. But also on datasets that do not exhibit such unfavorable shapes, we might encounter similar problems caused by the choice of k. As was already discussed in Figure III.2.10, the optimum solution on *666cities* for $k = 10$ is quite ambiguous, as the dataset does not actually consist of 10 clusters. There does not seem to be an “explanation” for why the optimum solution looks the way it does.

To emphasize this even more, we take a look at dataset *s1*, which clearly consists of 15 rather well-separated clusters. If we call all four algorithms 100 times for $k = 15$, we get the results illustrated in Figure III.2.22a. Save for `k-means++`, all algorithms consistently find the same low-cost solution[1], depicted in Figure III.2.23a. `LS++` and `FLS++` find it in every single run. But even on a dataset with such a clear structure, the wrong choice of k destabilizes the results. Especially for $k = 10$, the variance in solution quality is a lot bigger. If we take a look at the best found solution with 10 clusters Figure III.2.23b, we again notice that it looks somewhat arbitrary. Due to the fact that k is now smaller than the actual number of clusters, some neighboring clusters have to share a center, and the decision as to which ones have to share seems rather random. For $k = 25$ and $k = 100$ (Figure III.2.23c and d), we observe the opposite. Ground truth clusters are split up into multiple clusters, and the number of subdivisions as well as their shapes seem completely arbitrary.



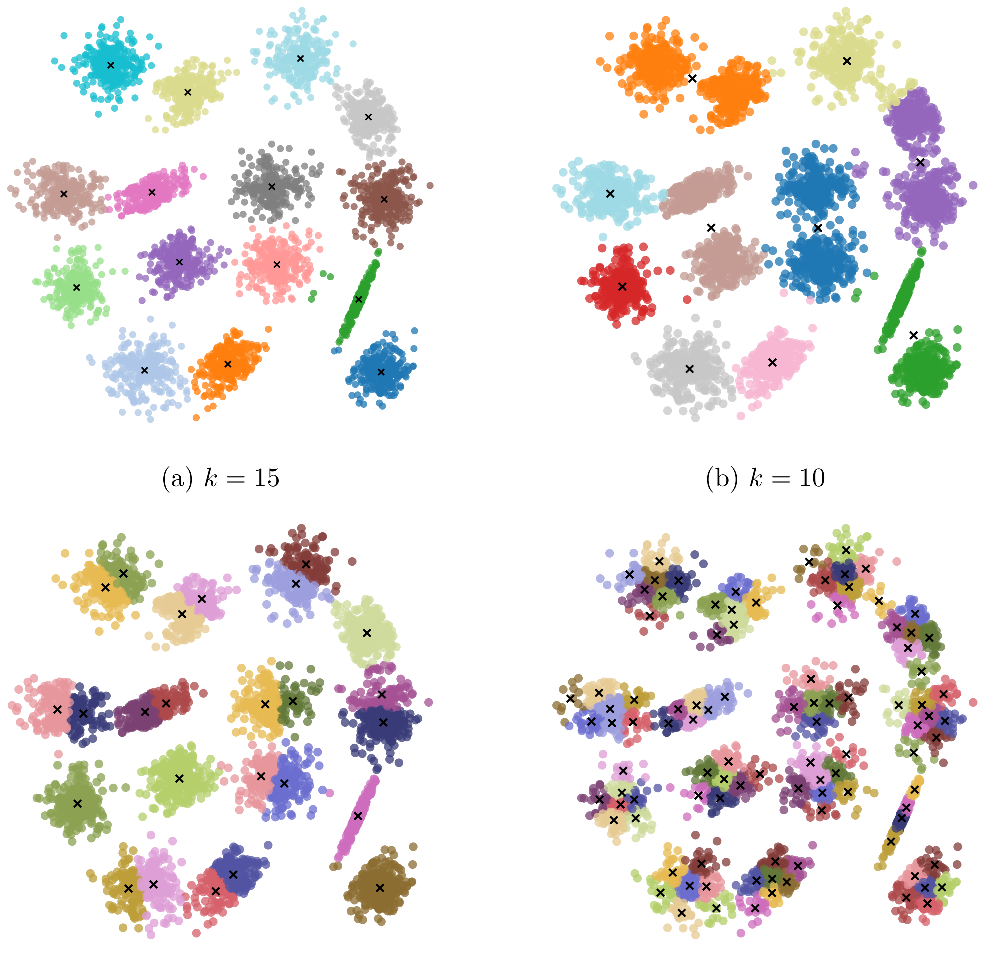
(a) $k = 15$ (b) $k = 10$

(c) $k = 25$ (d) $k = 100$

Figure III.2.22.: Boxplots for 100 runs on dataset *s1*, which actually consists of 15 clusters. If $k = 15$ is chosen, `LS++` and `FLS++` find the minimum cost solution in every single run. But if k is chosen differently, it becomes more unstable.

[1]This is not necessarily the global optimum solution, but it certainly *looks* – for the most part – sensible.

It is therefore a legitimate question whether finding exact (or almost) optimum solutions under these circumstances is a goal worth pursuing at all. Thinking back to one of the motivations for clustering given in the introduction – identifying *meaningful* structures inherent in data – and then looking at the assignment in Figure III.2.23d, it becomes evident that a low-cost solution is not necessarily a meaningful one. At the same time, there is an argument to be made that, from a purely theoretical perspective, it can very well be of value to investigate how algorithms perform in these cases. After all, the goal of approximation design is to derive guarantees for *all* possible inputs. There have been multiple attempts at bridging this gap between practical application



(a) $k = 15$ (b) $k = 10$

(c) $k = 25$ (d) $k = 100$

Figure III.2.23.: The best found solutions for each value of k.

and theory by defining so-called *clusterability assumptions*. These aim at formalizing the intuitive notion of an input being well "clusterable", and then showing that, on such inputs, better guarantees can be achieved. In [85], Ostrovsky et al. define a point set to be ε-separated for a certain k, if the cost of the optimal k-means solution is at most ε^2 times the optimal $(k-1)$-means solution. The motivation behind this definition stems from the heuristic method of trying out different consecutive values of k and looking at the cost development. For increasing k, cost should go down, and for the "right" k, there should – at least according to the assumption – be a noticeable leap. They give a randomized algorithm that, assuming ε-separatedness of the input, returns a constant-factor approximation with high probability in $\mathcal{O}(nkd + k^3d)$. They also show that their separation condition is equivalent to the condition that any two near-optimal solutions (for the correct k) disagree only on a small fraction of the input points. A somewhat similar notion is weak deletion stability, introduced by Awasthi et al. [15]. Here, an instance is called $(1+\alpha)$-stable, if deleting one center from an optimum solution and assigning all the points in that cluster to another center increases the cost by at least $(1+\alpha)$. Other attempts of defining clusterability include perturbation robustness. In [3], Ackerman and Ben-David introduce the notion of additive perturbation robustness. Here, an input set P is said to be ε-robust if an optimal k-clustering remains optimal after additively perturbing P by small amounts. Analogously, Bilu and Linial [27] introduce multiplicative perturbation robustness. Awasthi et al. [16] define center stability, where an instance is α-stable if, for any optimal clustering, all points are closer to their closest center than to any other center by a factor α.

While all of these notions sound reasonable, it is worth questioning to what extent they actually encapsulate the concept of clusterability, and if we can expect real world data to satisfy these conditions. In the survey paper [21], Ben-David points out that, for some notions, the values of the parameters (ε or α) needed to guarantee polynomial time exact algorithms are rather unrealistic. Additionally, Ackerman and Ben-David show in [3] that these notions are partially mutually exclusive. Returning to our "problematic" instances, it might be interesting to practically test whether they satisfy any of these conditions for non-trivial values of their respective parameters.

Ultimately, the inability of `FLS++` to find optimum solutions even for very large Z raises another potentially interesting question. All four algorithms that we compare in our experiments can ultimately be viewed as Lloyd's Algorithm with different initialization techniques: they compute a set of k centers and then run Lloyd's until convergence. So can we assume that, for the right choice of initial center set, a "Lloyd type" method will lead to the optimum solution? Or posed differently: are there datasets on which, for *any* set of initial centers that is chosen from the data points, Lloyd's Algorithm cannot converge to the optimum solution?

Acknowledgements

The proof of hardness of the assignment problem for k-MSR in Chapter II.1 is taken from [48], which is joint work with Jan Höckendorff, Joshua Könen and Kevin Schewior, and was written by me.

The results presented in Chapter II.2 as well as Observation 26 and Lemma 29 are mainly taken from [47], which is joint work with Annika Hennes, Abhiruk Lahiri, Melanie Schimdt and Julian Wargalla. The paper resulted from a long series of meetings in which we jointly worked out the details of the k-center approximation algorithm by Bădoiu et al. [31], checked which aspects carry over to the k-MSR setting and developed the idea of our k-MSR PTAS. The group that met during this development phase consisted of Lukas Drexler, Annika Hennes and Melanie Schmidt. Abhiruk Lahiri and Julian Wargalla joined the group in the final phase of the algorithm development. I significantly contributed to re-proving the correctness of the k-Center PTAS, in particular the multi-center case, developing the idea for the k-MSR PTAS and designing the final algorithm. My contribution to the paper roughly amounts to 25% of the overall workload. It should, however, be noted that the presentation of the results in Chapter II.2 differs significantly from the original paper. One reason for this is that the arguments presented in [47] contain a flaw that only got noticed after the publication. Hence, the results for mergeable constraints do not hold, and we only show a weaker version here.

The results presented in Chapter II.3 are mainly taken from [33], which is joint work with Lena Carta, Annika Hennes, Clemens Rösner and Melanie Schmidt. I began working on this topic in 2020, and I was involved in the development of the algorithmic idea and some central concepts (e.g. guessing the radii, modifying the distance function). After some setbacks, I abandoned the project for a while. It was later picked up by Annika Hennes, who contributed significantly in properly formalizing and fleshing out many parts of the idea. Additionally, the key insight that connected components of the access graph are fair (cf. Section II.3.4) was provided by her, which greatly reduced the approximation factor. I rejoined the project during the completion of the paper. Furthermore, I have fleshed out the descriptions of the special cases for lower bounds and 1:1-fairness (cf. Section II.3.6.1), which were only briefly mentioned in the original paper.

The results presented in Part III are partly taken from [41], which is joint work with Theo Conrads, Joshua Könen, Daniel Schmidt and Melanie Schmidt. The code used in this publication was jointly written by Joshua Könen and me, as was the formulation of the `Foresight-LS++` algorithm. We also jointly developed the pipeline to automate the experiments. The experiments on the general performance on large datasets and varying number of local search steps (both found in Section III.2.2) were conducted solely by me. The experiments concerning instances with known optima (Section III.2.2.5), as well as the discussion in Section III.2.3, were not part of that publication and were conducted by me explicitly for this thesis.

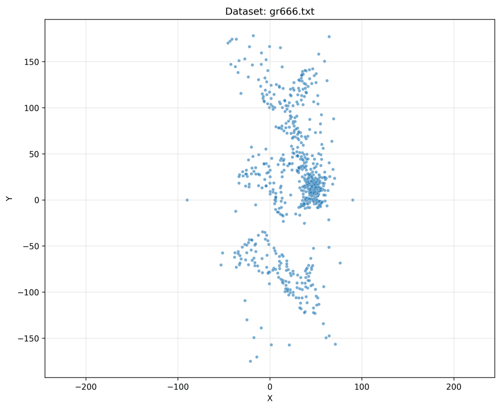
I would like to thank the DFG (Deutsche Forschungsgemeinschaft), as the work presented in this thesis was party funded by DFG project 456558332 ”Approximation Algorithms for Geometric Data Analysis and Their Practicability”.

I would like to thank my supervisor, Prof. Dr. Melanie Schmidt, for her guidance and support throughout my PhD. I am particularly grateful for her genuine concern for the well-being of her students and colleagues, for always being approachable, and for creating an exceptionally warm, pleasant and supportive working environment. Thanks to all of the Algorithms and Data Structures group for contributing to this environment. Special thanks go to Annika Hennes, not only for the great collaboration on two papers, but for proofreading large parts of my thesis and providing extremely useful feedback. Thanks to all co-authors mentioned above for the great collaborations. I would like to thank Prof. Dr. Heiko Röglin for agreeing to be the second reviewer of my thesis. In addition, his excellent lectures played a major role in my decision to focus on theory.

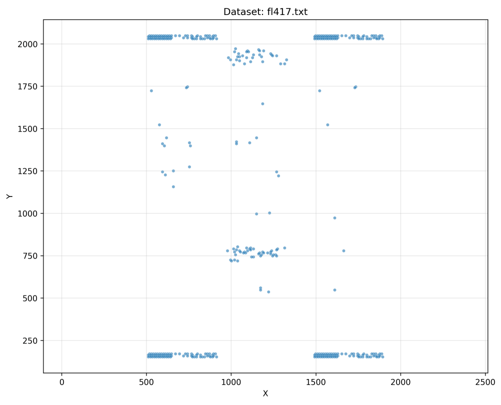
I thank my parents for always being there for me and supporting me in all my decisions. Finally, I thank Vicki and Ava Abresch. I don’t know if I could have done this without your love and support.

Appendix

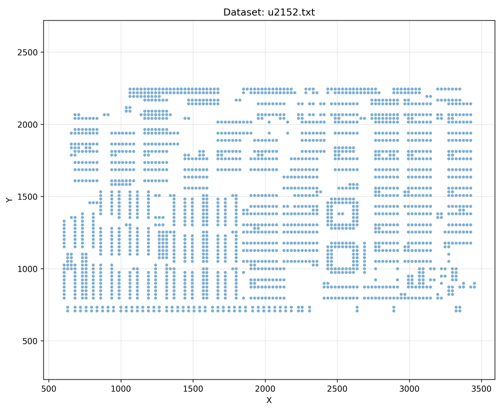
A. 2d Datasets and Optimum Solution Values



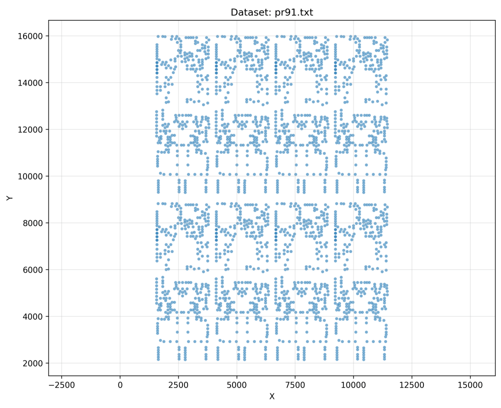
(a) 666cities



(b) fl417

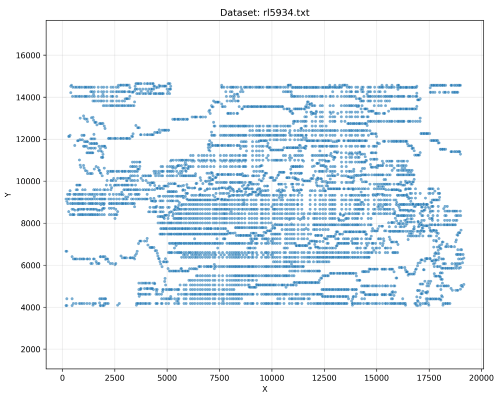


(c) u2152

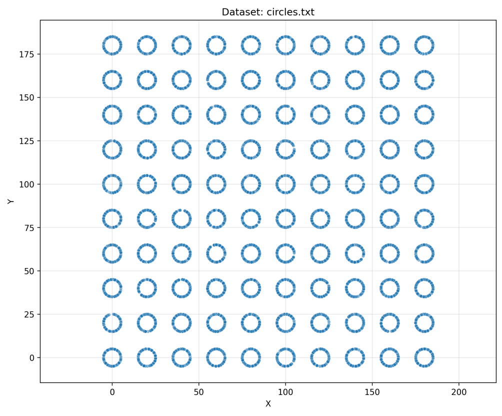


(d) pr91

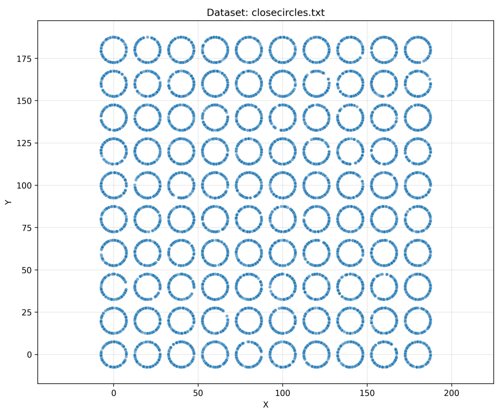
Figure A.1.: Selection of 2d datasets used for the experiments.



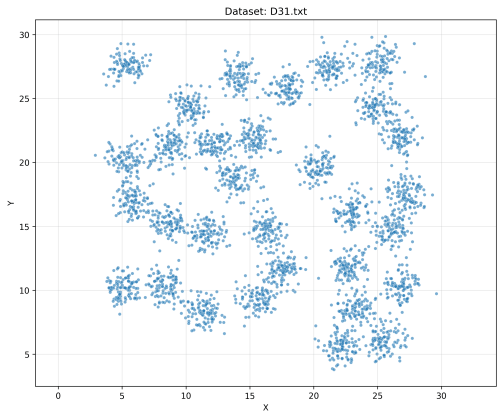
(a) *rl5934*



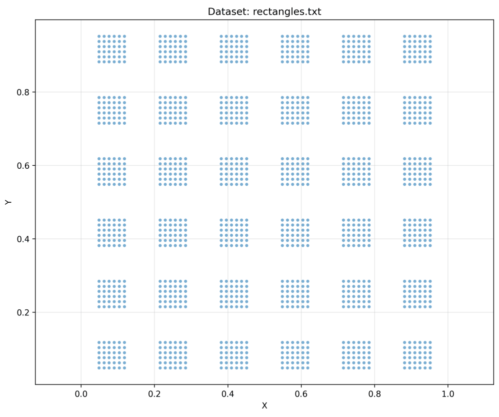
(b) *circles*



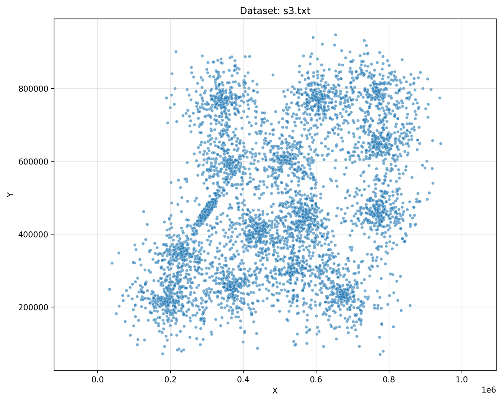
(c) *close circles*



(d) *D31*

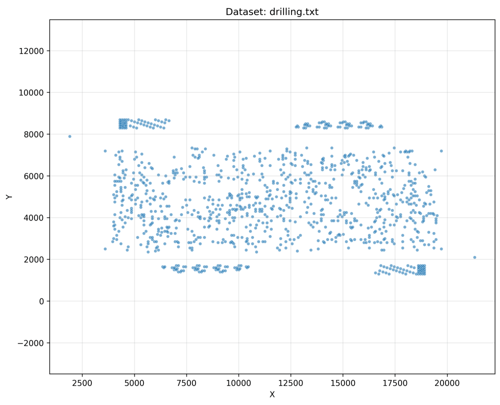


(e) *rectangles*

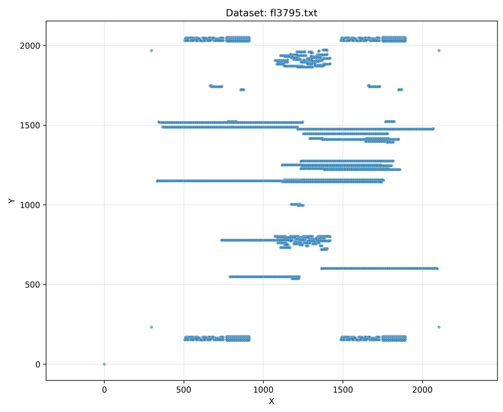


(f) *s3*

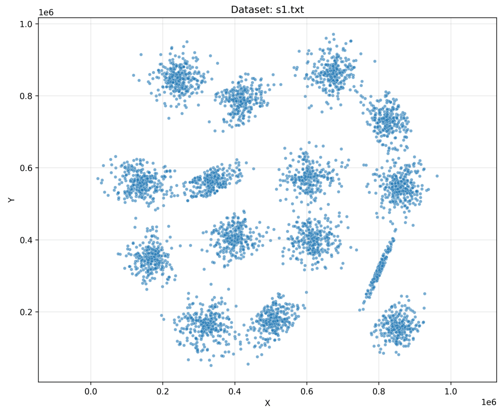
Figure A.2.: Selection of 2d datasets used for the experiments.



(a) drilling



(b) fl3795



(c) s1

Figure A.3.: Selection of 2d datasets used for the experiments.

Dataset	k	Opt
202cities	6	6764.88487
fl417	16	2017630.97
666cities	4	613995.08
	6	382676.87
	10	224183.98
pr91	4	14118367258
	8	7013383132
	10	5324924228
	100	404498401
	200	175431272
	250	132351731
	300	101568507
drilling	100	96317864
	200	36157288
u2152	100	11718395
	200	5163871
fl3795	300	562133.80
rl5934	100	1477892122

Table A.1.: Optimal solution values for different datasets and values of k.

Bibliography

[1] Kdd cup 2004 datasets, 2004. Available at https://www.kdd.org/kdd-cup/view/kdd-cup-2004/Data".

[2] Synthetic Circle Data Set. UCI Machine Learning Repository, 2024. DOI: https://doi.org/10.24432/C51909.

[3] Margareta Ackerman and Shai Ben-David. Clusterability: A theoretical study. In *Artificial intelligence and statistics*, pages 1–8. PMLR, 2009.

[4] Marek Adamczyk, Jarosław Byrka, Jan Marcinkowski, Syed M Meesum, and Michał Włodarczyk. Constant-factor FPT approximation for capacitated k-median. In *27th Annual European Symposium on Algorithms, ESA 2019*, 2019.

[5] Ankit Aggarwal, Amit Deshpande, and Ravi Kannan. Adaptive sampling for k-means clustering. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 15–28. Springer, 2009.

[6] Sara Ahmadian, Ashkan Norouzi-Fard, Ola Svensson, and Justin Ward. Better guarantees for k-means and euclidean k-median by primal-dual algorithms. *SIAM Journal on Computing*, 49(4):FOCS17–97, 2019.

[7] Sara Ahmadian and Chaitanya Swamy. Approximation Algorithms for Clustering Problems with Lower Bounds and Outliers. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, volume 55 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 69:1–69:15, 2016.

[8] Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat. NP-hardness of euclidean sum-of-squares clustering. *Machine learning*, 75:245–248, 2009.

[9] Daniel Aloise, Pierre Hansen, and Leo Liberti. An improved column generation algorithm for minimum sum-of-squares clustering. *Mathematical Programming*, 131:195–220, 2012.

[10] Helmut Alt, Esther M. Arkin, Hervé Brönnimann, Jeff Erickson, Sándor P. Fekete, Christian Knauer, Jonathan Lenchner, Joseph S. B. Mitchell, and Kim Whittlesey. Minimum-cost coverage of point sets by disks. In *Proceedings of the Twenty-Second Annual Symposium on Computational Geometry*, SCG '06, page 449–458. Association for Computing Machinery, 2006.

[11] Hyung-Chan An, Aditya Bhaskara, Chandra Chekuri, Shalmoli Gupta, Vivek Madan, and Ola Svensson. Centrality of trees for capacitated k-center. *Mathematical Programming*, 154(1):29–53, 2015.

[12] David Arthur, Bodo Manthey, and Heiko Röglin. Smoothed analysis of the k-means method. *Journal of the ACM (JACM)*, 58(5):1–31, 2011.

[13] David Arthur and Sergei Vassilvitskii. How slow is the k-means method? In *Proceedings of the twenty-second annual symposium on Computational geometry*, pages 144–153, 2006.

[14] David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, page 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[15] Pranjal Awasthi, Avrim Blum, and Or Sheffet. Stability yields a PTAS for k-median and k-means clustering. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 309–318. IEEE, 2010.

[16] Pranjal Awasthi, Avrim Blum, and Or Sheffet. Center-based clustering under perturbation stability. *Information Processing Letters*, 112(1-2):49–54, 2012.

[17] Pranjal Awasthi, Moses Charikar, Ravishankar Krishnaswamy, and Ali K Sinop. The hardness of approximation of euclidean k-means. In *31st International Symposium on Computational Geometry (SoCG 2015)*, 2015.

[18] Sayan Bandyapadhyay and Tianzi Chen. Improved FPT approximation for sum of radii clustering with mergeable constraints. In Alina Ene and Eshan Chattopadhyay, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2025)*, 2025.

[19] Sayan Bandyapadhyay, William Lochet, and Saket Saurabh. FPT Constant-Approximations for Capacitated Clustering to Minimize the Sum of Cluster Radii. In *39th International Symposium on Computational Geometry (SoCG 2023)*, volume 258 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:14, 2023.

[20] Babak Behsaz. *Approximation algorithms for clustering problems.* PhD thesis, University of Alberta, 2012.

[21] Shai Ben-David. Clustering is easy when.... what? *arXiv preprint arXiv:1510.05336*, 2015.

[22] Suman Bera, Deeparnab Chakrabarty, Nicolas Flores, and Maryam Negahbani. Fair algorithms for clustering. *Advances in Neural Information Processing Systems*, 32, 2019.

[23] Ioana O Bercea, Martin Groß, Samir Khuller, Aounon Kumar, Clemens Rösner, Daniel R Schmidt, and Melanie Schmidt. On the cost of essentially fair clusterings. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2019)*, pages 18–1, 2019.

[24] Lorenzo Beretta, Vincent Cohen-Addad, Silvio Lattanzi, and Nikos Parotsidis. Multi-swap k-means++. *Advances in Neural Information Processing Systems*, 36:26069–26091, 2023.

[25] Anup Bhattacharya, Jan Eube, Heiko Röglin, and Melanie Schmidt. Noisy, greedy and not so greedy k-means++. In *28th Annual European Symposium on Algorithms (ESA 2020)*, 2020.

[26] Anup Bhattacharya, Ragesh Jaiswal, and Nir Ailon. Tight lower bound instances for k-means++ in two dimensions. *Theoretical Computer Science*, 634:55–66, 2016.

[27] Yonatan Bilu and Nathan Linial. Are stable instances easy? *Combinatorics, Probability and Computing*, 21(5):643–660, 2012.

[28] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[29] Matteo Böhm, Adriano Fazzone, Stefano Leonardi, and Chris Schwiegelshohn. Fair clustering with multiple colors. *arXiv preprint arXiv:2002.07892*, 2020.

[30] Tobias Brunsch and Heiko Röglin. A bad instance for k-means++. *Theoretical Computer Science*, 505:19–26, 2013. Theory and Applications of Models of Computation 2011.

[31] Mihai Bădoiu, Sariel Har-Peled, and Piotr Indyk. Approximate clustering via coresets. In *Proceedings of the Thirty-Fourth Annual ACM Symposium on Theory of Computing*, STOC '02, page 250–257. Association for Computing Machinery, 2002.

[32] Moritz Buchem, Katja Ettmayr, Hugo K. K. Rosado, and Andreas Wiese. *A $(3 + \varepsilon)$-approximation algorithm for the minimum sum of radii problem with outliers and extensions for generalized lower bounds*, pages 1738–1765. 2024.

[33] Lena Carta, Lukas Drexler, Annika Hennes, Clemens Rösner, and Melanie Schmidt. FPT Approximations for Fair k-Min-Sum-Radii. In *35th International Symposium on Algorithms and Computation (ISAAC 2024)*, volume 322 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:18, Dagstuhl, Germany, 2024.

[34] Moses Charikar and Rina Panigrahy. Clustering to minimize the sum of cluster diameters. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, STOC '01, pages 1–10. Association for Computing Machinery, July 2001.

[35] Xianrun Chen, Dachuan Xu, Yicheng Xu, and Yong Zhang. Parameterized approximation algorithms for sum of radii clustering and variants. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 20666–20673, 2024.

[36] Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, and Sergei Vassilvitskii. Fair clustering through fairlets. *Advances in neural information processing systems*, 30, 2017.

[37] Davin Choo, Christoph Grunau, Julian Portmann, and Václav Rozhon. k-means++: few more steps yield constant approximation. In *International Conference on Machine Learning*, pages 1909–1917. PMLR, 2020.

[38] Vincent Cohen-Addad, Hossein Esfandiari, Vahab Mirrokni, and Shyam Narayanan. Improved approximations for euclidean k-means and k-median, via nested quasi-independent sets. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, page 1621–1628. Association for Computing Machinery, 2022.

[39] Vincent Cohen-Addad, Karthik C. S., and Euiwoong Lee. Johnson coverage hypothesis: Inapproximability of k-means and k-median in ℓ_p-metrics. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1493–1530, 2022.

[40] Theo Conrads. Lokale Such- und Samplingmethoden für das k-Means- und k-Median-Problem, 2021.

[41] Theo Conrads, Lukas Drexler, Joshua Könen, Daniel R. Schmidt, and Melanie Schmidt. Local search k-means++ with foresight. In *22nd International Symposium on Experimental Algorithms, SEA 2024, July 23-26, 2024, Vienna, Austria*, volume 301 of *LIPIcs*, pages 7:1–7:20, 2024.

[42] Theo Conrads, Lukas Drexler, Joshua Könen, Daniel R Schmidt, and Melanie Schmidt. Local search k-means++ with foresight. *arXiv preprint arXiv:2406.02739*, 2024.

[43] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2015.

[44] Marek Cygan, MohammadTaghi Hajiaghayi, and Samir Khuller. LP rounding for k-centers with non-uniform hard capacities. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 273–282. IEEE, 2012.

[45] Sanjoy Dasgupta. The hardness of k-means clustering. Technical report, Department of Computer Science and Engineering, University of California, 2008.

[46] Sanjoy Dasgupta and Philip M Long. Performance guarantees for hierarchical clustering. *Journal of Computer and System Sciences*, 70(4):555–569, 2005.

[47] Lukas Drexler, Annika Hennes, Abhiruk Lahiri, Melanie Schmidt, and Julian Wargalla. Approximating fair k-min-sum-radii in euclidean space. In *International Workshop on Approximation and Online Algorithms*, pages 119–133. Springer, 2023.

[48] Lukas Drexler, Jan Höckendorff, Joshua Könen, and Kevin Schewior. Clustering graphs of bounded treewidth to minimize the sum of radius-dependent costs. *arXiv preprint arXiv:2310.02130*, 2023.

[49] Petros Drineas, Alan Frieze, Ravi Kannan, Santosh Vempala, and Vishwanathan Vinay. Clustering large graphs via the singular value decomposition. *Machine learning*, 56:9–33, 2004.

[50] Charles Elkan. Using the triangle inequality to accelerate k-means. In Tom Fawcett and Nina Mishra, editors, *Proc. of the 20th ICML*, pages 147–153, 2003.

[51] Brian S. Everitt, Sabine Landau, Morven Leese, and Daniel Stahl. *Cluster analysis*. Wiley, 2011.

[52] Dan Feldman, Morteza Monemizadeh, and Christian Sohler. A ptas for k-means clustering based on weak coresets. In *Proceedings of the Twenty-Third Annual Symposium on Computational Geometry*, SCG '07, page 11–18. Association for Computing Machinery, 2007.

[53] Gereon Frahling and Christian Sohler. A fast k-means implementation using coresets. In *Proc. of the 22nd SoCG*, page 135–143, 2006.

[54] Pasi Fränti and Sami Sieranoja. K-means properties on six clustering benchmark datasets. *Appl. Intell.*, 48(12):4743–4759, 2018.

[55] Zachary Friggstad and Mahya Jamshidian. Improved Polynomial-Time Approximations for Clustering with Minimum Sum of Radii or Diameters. In *30th Annual European Symposium on Algorithms (ESA 2022)*, volume 244 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 56:1–56:14, 2022.

[56] Matt Gibson, Gaurav Kanade, Erik Krohn, Imran A Pirwani, and Kasturi Varadarajan. On metric clustering to minimize the sum of radii. *Algorithmica*, 57(3):484–498, 2010.

[57] Matt Gibson, Gaurav Kanade, Erik Krohn, Imran A. Pirwani, and Kasturi Varadarajan. On Clustering to Minimize the Sum of Radii. *SIAM Journal on Computing*, 41(1):47–60, January 2012. Publisher: Society for Industrial and Applied Mathematics.

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

[59] Fabrizio Grandoni, Rafail Ostrovsky, Yuval Rabani, Leonard J. Schulman, and Rakesh Venkat. A refined approximation for euclidean k-means. *Information Processing Letters*, 176:106251, 2022.

[60] Christoph Grunau, Ahmet Alper Özüdoğru, Václav Rozhoň, and Jakub Tětek. *A Nearly Tight Analysis of Greedy k-means++*, pages 1012–1070.

[61] Greg Hamerly. Making k-means even faster. In *SDM*, pages 130–140. SIAM, 2010.

[62] Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '04, page 291–300. Association for Computing Machinery, 2004.

[63] Sariel Har-Peled and Bardia Sadri. How fast is the k-means method? *Algorithmica*, 41:185–202, 2005.

[64] Grete Heinz, Louis J. Peterson, Roger W. Johnson, and Carter J. Kerk. Exploring relationships in body dimensions. *Journal of Statistics Education*, 11(2), 2003.

[65] Dorit S Hochbaum. When are np-hard location problems easy? *Annals of Operations Research*, 1(3):201–214, 1984.

[66] Wen-Lian Hsu and George L Nemhauser. Easy and hard bottleneck location problems. *Discrete Applied Mathematics*, 1(3):209–215, 1979.

[67] Mary Inaba, Naoki Katoh, and Hiroshi Imai. Applications of weighted voronoi diagrams and randomization to variance-based k-clustering. In *Proceedings of the tenth annual symposium on Computational geometry*, pages 332–339, 1994.

[68] Tanmay Inamdar and Kasturi Varadarajan. Capacitated sum-of-radii clustering: An fpt approximation. In *28th Annual European Symposium on Algorithms (ESA 2020)*, pages 62–1, 2020.

[69] Kamal Jain and Vijay V Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *Journal of the ACM (JACM)*, 48(2):274–296, 2001.

[70] Ragesh Jaiswal, Amit Kumar, and Jatin Yadav. FPT Approximation for Capacitated Sum of Radii. In Venkatesan Guruswami, editor, *15th Innovations in Theoretical Computer Science Conference (ITCS 2024)*, volume 287 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 65:1–65:21, 2024.

[71] Linus Källberg. *Minimum enclosing balls and ellipsoids in general dimensions.* PhD thesis, Mälardalen University, 2019.

[72] 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. In *Proceedings of the eighteenth annual symposium on Computational geometry*, pages 10–18, 2002.

[73] Samir Khuller and Yoram J Sussmann. The capacitated k-center problem. *SIAM Journal on Discrete Mathematics*, 13(3):403–418, 2000.

[74] Matthäus Kleindessner, Pranjal Awasthi, and Jamie Morgenstern. Fair k-center clustering for data summarization. In *International Conference on Machine Learning*, pages 3448–3457. PMLR, 2019.

[75] Bernhard Korte and Jens Vygen. *Combinatorial optimization: theory and algorithms*. Springer.

[76] A. Kumar, Y. Sabharwal, and S. Sen. A simple linear time (1 + /spl epsiv/)-approximation algorithm for k-means clustering in any dimensions. In *45th Annual IEEE Symposium on Foundations of Computer Science*, pages 454–462, 2004.

[77] Silvio Lattanzi and Christian Sohler. A better k-means++ algorithm via local search. In *Proc. of the 36th ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 3662–3671. PMLR, 09–15 Jun 2019.

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

[79] Nissan Lev-Tov and David Peleg. Polynomial time approximation schemes for base station coverage with minimum total radii. *Computer Networks*, 47(4):489–501, 2005.

[80] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[81] James MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, volume 5, pages 281–298. University of California press, 1967.

[82] Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. The planar k-means problem is np-hard. *Theoretical Computer Science*, 442:13–21, 2012.

[83] Jiří Matoušek. On approximate geometric k-clustering. *Discrete & Computational Geometry*, 24(1):61–84, 2000.

[84] Ramgopal R Mettu and C Greg Plaxton. Optimal time bounds for approximate clustering. *Machine Learning*, 56(1):35–60, 2004.

[85] Rafail Ostrovsky, Yuval Rabani, Leonard J Schulman, and Chaitanya Swamy. The effectiveness of lloyd-type methods for the k-means problem. *Journal of the ACM (JACM)*, 59(6):1–22, 2013.

[86] Manfred Padberg and Giovanni Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33(1):60–100, 1991.

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

[88] Melanie Schmidt. Lecture Notes on Cluster Analysis, February 2019. Rheinische Friedrich-Wilhelms-Universität Bonn.

[89] Antonio M Sudoso and Daniel Aloise. A column generation algorithm with dynamic constraint aggregation for minimum sum-of-squares clustering. *arXiv preprint arXiv:2410.06187*, 2024.

[90] Matus Telgarsky and Andrea Vattani. Hartigan's method: k-means clustering without voronoi. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 820–827. JMLR Workshop and Conference Proceedings, 2010.

[91] Andrea Vattani. The hardness of k-means clustering in the plane. Technical report, University of California, San Diego, 2009.

[92] Andrea Vattani. K-means requires exponentially many iterations even in the plane. In *Proceedings of the twenty-fifth annual symposium on Computational geometry*, pages 324–332, 2009.

[93] Emo Welzl. Smallest enclosing disks (balls and ellipsoids). In *New Results and New Trends in Computer Science: Graz, Austria, June 20–21, 1991 Proceedings*, pages 359–370. Springer, 2005.

[94] Shuyin Xia, Daowan Peng, Deyu Meng, Changqing Zhang, Guoyin Wang, Elisabeth Giem, Wei Wei, and Zizhong Chen. Ball k k-means: Fast adaptive clustering with no bounds. *IEEE transactions on pattern analysis and machine intelligence*, 44(1):87–99, 2020.

[95] I.-C. Yeh. Modeling of strength of high-performance concrete using artificial neural networks. *Cement and Concrete Research*, 28(12):1797–1808, 1998.

[96] E Alper Yildirim. Two algorithms for the minimum enclosing ball problem. *SIAM Journal on Optimization*, 19(3):1368–1391, 2008.