Coresets and Streaming Algorithms for the k-means Problem and Related Clustering Objectives

Dissertation

zur Erlangung des Grades eines
Doktors der Naturwissenschaften
der Technischen Universität Dortmund
an der Fakultät für Informatik
von

Melanie Schmidt

Dortmund
Dezember 2014

Tag der mündlichen Prüfung:	17.12.2014
Dekan:	Prof. Dr. Gernot Fink
Gutachter:	Prof. Dr. Christian Sohler,
	Prof. Dr. Johannes Blömer

Abstract

The continuing technological advances in different areas represent a challenge for researchers in computer science and in particular in the area of algorithms and theory. The gap between processing speed and data volume increases constantly, even though the performance of computers and their central processing units increases at a fast rate. This is because the data that surrounds us multiplies at an even more rapid pace. One example for the phenomenon is the Large Hadron Collider (CERN) that generates more than half a gigabyte of data every second. Even algorithms with linear running time are here too slow if they need random access to the data. Data stream algorithms are algorithms that only need one pass over the data to (approximately) solve a problem. Their memory usage is usually polynomial in the logarithm of the input size. Ideally, a data stream algorithm can process the data directly while it is created.

In my thesis, I consider *k-means clustering*. Given n points in the d-dimensional Euclidean space $\mathbb{R}^d$, the k-means problem is to compute k centers which minimize the sum of the squared distances of all points to their closest center. The centers can be chosen arbitrarily from $\mathbb{R}^d$. For a given solution, i. e., a set of k centers, we say that the sum of the squared distances is the *k-means cost* of this solution. The k-means problem has been studied for sixty years and often occurs in machine learning, also as a subproblem.

In the context of data streams, a popular technique to solve the k-means problem is the computation of *coresets*. A coreset for a point set P is a (usually much smaller) point set S which has approximately the same cost as P for any possible solution. More precisely and defined for the k-means problem, a $(1+\varepsilon)$-coreset for an $\varepsilon \in (0,1)$ is a set S that satisfies that the cost of S for any set of k centers C is at least an ε-fraction off the cost of P with the same centers C.

A coreset computation is often first designed as a polynomial algorithm with random access to the data. Then, the algorithm is converted into a data stream algorithm by using a technique which is known as Merge-and-Reduce. By using Merge-and-Reduce, the memory usage of the algorithm is usually increased by a factor which is polynomial in $\log n$. In joint work with Hendrik Fichtenberger, Marc Bury (né Gillé), Chris Schwiegelshohn and Christian Sohler, I developed a data stream algorithm for the k-means problem which does not use Merge-and-Reduce. It processes the input points one by one and directly inserts them into an appropriate data structure. We use a data structure which is used in BIRCH (Zhang, Ramakrishnan, Livny, 1997), an algorithm which is very popular in practical applications. By analyzing and improving the data structure, we could develop an algorithm which computes a $(1+\varepsilon)$-coreset in the data stream model and that uses pointwise updates. Our algorithm is named BICO as a combination of BIRCH and the term coreset. The memory usage of BICO is bounded by $\mathcal{O}(k \cdot \log n \cdot \varepsilon^{-(d+1)})$ if the dimension of the input points is a constant. We implemented a slightly modified version of BICO and combined it with an algorithm for the k-means problem which is known for its good results in practical applications. In an experimental study, we verified that the combined implementation computes solutions with high quality while it is much faster than other implementations that compute solutions of high quality. Our work was published at the

European Symposium on Algorithms 2013.

Joint work with Dan Feldman and Christian Sohler led to an important result on the k-means problem which was published at the ACM-SIAM Symposium on Discrete Algorithms 2013. It is a dimensionality reduction. If data is high dimensional, then reducing the dimension is as useful as reducing the number of points. We showed that it is possible to reduce the dimension of any k-means input to $\mathcal{O}(k/\varepsilon^2)$ while preserving the cost function up to an ε-fraction. The number of dimensions is in particular independent of the number of input points. The dimensionality reduction uses the singular value decomposition, a tool which is often used in practical applications. By combining the dimensionality result with known coreset constructions, we developed a construction that computes coresets of a size which is independent of the dimension *and* of the number of input points. To achieve this, we extended the usual coreset definition by allowing to store an additional constant. This does not increase the memory usage by much. This extension has the potential to allow for powerful coreset constructions. We develop an additional independent coreset construction for the k-means problem which also computes coresets of a size which is independent of the dimension and number of input points. The size of the computed coresets is larger, but it is applicable to at least one additional clustering problem which can (probably) not be solved by applying the singular value decomposition.

The results described so far are the core of the first part of this thesis. The second part contains results on related objective functions, including a so-called projective clustering problem. Additionally, the second part considers a probabilistic k-median problem which is joint work with Christiane Lammersen and Christian Sohler. The k-median problem is defined with Euclidean distances that are not squared, other than that it is identical to the k-means problem. In the probabilistic version we assume that the input points can appear at different locations. The probability for a point to appear at a certain location is given by a discrete probablity distribution. The task is to compute k centers that minimize the expected k-median cost. We develop a definition of a coreset for probabilistic clustering and show how to compute a coreset. This result was published at the Workshop on Approximation and Online Algorithms 2012.

Zusammenfassung

Die technologischen Weiterentwicklungen in verschiedensten Bereichen bringen neue Herausforderungen für die Informatik und insbesondere auch für die Algorithmik und die theoretische Informatik. Obwohl Prozessorgeschwindigkeiten und rechnerinterne Kommunikationsgeschwindigkeiten schnell anwachsen, vergrößern sich die Datenmengen um uns herum so viel rasanter, dass Algorithmen, die man klassischerweise als effizient bezeichnen würde, ungeeignet werden. Ein Beispiel dafür ist der Large Hadron Collider (‚Großer Hadronen-Speicherring', CERN), der über ein halbes Gigabyte an Daten pro Sekunde produziert. Selbst Algorithmen mit linearer Laufzeit können zu langsam sein, wenn sie wahlfreien Zugriff auf die Daten benötigen. Datenstromalgorithmen sind Algorithmen, die für die (approximative) Lösung eines Problems nur einen einzigen Durchlauf durch die gegebenen Daten benötigen, und deren Speicherplatz im Vergleich zur verarbeiteten Datenmenge sehr klein ist, üblicherweise ein Polynom im Logarithmus der Eingabegröße. Im Idealfall können Daten mit einem Datenstromalgorithmus direkt während der Entstehung verarbeitet werden.

In meiner Dissertation beschäftige ich mich mit sogenanntem *k-means Clustering*, unter anderem im Datenstrommodell. Für eine Menge von Punkten im d-dimensionalen Euklidischen Raum $\mathbb{R}^d$ ist die Aufgabe, k Zentren zu finden, so dass die Summe der quadrierten Distanzen aller Punkte zu den ihnen jeweils nächsten Zentren minimiert wird. Die Zentren können dabei frei aus dem $\mathbb{R}^d$ gewählt werden. Für eine mögliche Lösung, also eine Menge von k Zentren C, bezeichnen wir die Zielfunktion im Folgenden auch als k-means Kosten von P mit Zentren C. Das k-means Problem wird seit über sechzig Jahren untersucht und tritt im maschinellen Lernen häufig auf, z.B. auch als Teilproblem.

Eine häufig verwendete Technik, um das k-means Problem im Datenstrommodell zu lösen, ist die Berechnung von *Kernmengen*. Eine Kernmenge für eine Punktmenge P ist eine (normalerweise deutlich kleinere) Punktmenge S, die approximativ für jeden möglichen Lösungskandidaten die gleichen Kosten hat wie P. Genauer gesagt und spezifisch für das k-means Problem formuliert ist eine $(1+\varepsilon)$-Kernmenge für ein $\varepsilon \in (0,1)$ eine Menge S, für die die k-means Kosten von S für jede mögliche Zentrenmenge C aus k Zentren höchstens um einen ϵ-Anteil von den k-means Kosten von P mit denselben Zentren C abweichen.

Für die Berechnung einer Kernmenge wird häufig erst ein polynomieller Algorithmus entworfen, der die Daten auch mehrfach lesen kann. Dieser wird dann mit Hilfe einer Technik, die als *Merge-and-Reduce* bekannt ist, in einen Datenstromalgorithmus umgewandelt. Dadurch erhöht sich in der Regel der Speicherbedarf um einen Faktor, der polynomiell in $\log n$ ist. Zusammen mit Hendrik Fichtenberger, Marc Bury (geb. Gillé), Chris Schwiegelsohn und Christian Sohler habe ich einen Datenstromalgorithmus für das k-means Problem entwickelt, der ohne die Merge-and-Reduce Technik auskommt. Dieser verarbeitet die Punkte im Eingabestrom einzeln und nimmt sie direkt in eine geeignete Datenstruktur. Wir verwenden eine Datenstruktur, die bereits in einem in der praktischen Anwendung sehr beliebten Algorithmus, BIRCH (Zhang, Ramakrishnan, Livny, 1997), ihre Schnelligkeit demonstriert hat. Durch eine Analyse und Weiterentwicklung dieser Datenstruktur waren wir in der Lage, einen Algorithmus zu entwickeln, der im Datenstrommodell eine $(1+\varepsilon)$-

Kernmenge berechnet und nur punktweise Aktualisierungen der Datenstruktur benötigt. Der Name unseres Algorithmus ist BICO, was eine Kombination aus BIRCH und dem englischen Wort für Kernmenge, *coreset*, ist. Der Speicherbedarf von BICO ist für Eingaben mit konstanter Dimension durch $\mathcal{O}(k \cdot \log n \cdot \varepsilon^{-(d+1)})$ beschränkt. Wir haben BICO in einer leichten Abwandlung implementiert und mit einem Algorithmus für das k-means Problem, der für seine guten Ergebnisse in praktischen Anwendungen bekannt ist, kombiniert. In einer experimentellen Studie konnten wir belegen, dass die kombinierte Implementierung Lösungen von hoher Qualität berechnet und sehr viel schneller ist als andere bekannte Implementierungen von Algorithmen, die Lösungen von hoher Qualität berechnen. Die Arbeit wurde auf dem European Symposium on Algorithms 2013 veröffentlicht.

Ein grundlegendes Ergebnis zum k-Means Problem ist in Zusammenarbeit mit Dan Feldman und Christian Sohler entstanden und wurde auf dem ACM-SIAM Symposium on Algorithms 2013 veröffentlicht. Unser erstes Resultat ist eine Dimensionsreduktion. Im Falle von hochdimensionalen Daten ist es nicht nur hilfreich, die Anzahl der Punkte zu reduzieren, sondern noch viel mehr, ihre Dimension zu verkleinern. Wir zeigen, dass man die Dimension einer k-Means Eingabe auf $\mathcal{O}(k/\varepsilon^2)$ reduzieren kann, während sich die Zielfunktion maximal um einen ε-Anteil ändert. Die Anzahl der Dimensionen ist insbesondere unabhängig von der ursprünglichen Dimension. Die niedrigdimensionalen Punkte kann man mit einer Singulärwertzerlegung berechnen, ein Werkzeug, das auch in der praktischen Anwendung häufig verwendet wird. Durch eine Kombination mit vorherigen Kernmengenkonstruktionen konnten wir einen Algorithmus entwerfen, der eine Kernmenge berechnet, deren Größe unabhängig von der Anzahl *und* der Dimension der Eingabepunkte ist. Wir erweitern dazu die übliche Kernmengendefinition, indem wir das Speichern einer zusätzlichen Konstanten erlauben. Dadurch entsteht kein signifikanter zusätzlicher Speicheraufwand. Das Potential dieser Erweiterung demonstrieren wir auch durch den Entwurf einer zweiten vollkommen unabhängigen Konstruktion für das k-means Problem, die ebenfalls eine Kernmenge mit zusätzlicher Konstante berechnet, deren Größe unabhängig von der Anzahl und Dimension der Eingabepunkte ist. Die Größe dieser Kernmenge ist größer als die aus der ersten Konstruktion, dafür lässt sich die Konstruktion auf mindestens ein anderes verwandtes Clusteringproblem übertragen, für das die Anwendung einer Singulärwertzerlegung (wahrscheinlich) nicht möglich ist.

Die bisher beschriebenen Ergebnisse sind der Kern des ersten Teils meiner Arbeit. Im zweiten Teil beschreibe ich Ergebnisse für verwandte Zielfunktionen, unter anderem aus dem Bereich des sogenannten projektiven Clusterings. Außerdem enthält dieser Teil ein gemeinsames Resultat mit Christiane Lammersen und Christian Sohler für das probabilistische k-median Problem. Das k-median Problem verwendet Euklidische Distanzen, die nicht quadriert werden, und ist ansonsten identisch zum k-means Problem. In der probabilistischen Version nehmen wir an, dass die Eingabepunkte keine feste Koordinaten haben, sondern als diskrete Wahrscheinlichkeitsverteilung über mögliche Positionen gegeben sind. Das Problem ist nun, k Zentren zu berechnen, für die die erwarteten k-median Kosten minimal sind. Wir entwickeln eine Kernmengendefinition für diesen Fall und zeigen, wie man eine Kernmenge berechnen kann. Diese Arbeit haben wir auf dem Workshop on Approximation and Online Algorithms 2012 veröffentlicht.

Acknowledgements

First of all, I would like to thank my advisor Prof. Dr. Christian Sohler from whom I have learned a lot about how to choose reseach questions and how to solve them. Thank you for advising and supporting me, for pointing me to plenty interesting research questions and for giving me the opportunity to visit summer schools and other researchers.

During the majority of my PhD time, I worked on the project 'A practical theory for clustering algorithms' in the DFG Priority Programme 1307 'Algorithm Engineering'. I thank the DFG for the funding. It was a joint project with the group of Prof. Dr. Johannes Blömer in Paderborn. I thank him and Kathrin Bujna, Dr. Daniel Kuntze and Dr. Marcel R. Ackermann for a creative environment and an enjoyable collaboration. Prof. Dr. Johannes Blömer also agreed to assess my thesis and I thank him for that, too.

I thank Marc Bury, Hendrik Fichtenberger, Dr. Martin Groß, Dr. Jan-Philipp Kappmeier, Dr. Christiane Lammersen, Dr. Rainer Penninger, Arseny Pyatikhatka, Dr. Daniel Schmidt, Magdalena Schmidt, Chris Schwiegelshohn and Dr. Christine Zarges for proofreading parts of my thesis. Your comments were invaluable. Special thanks go to Martin and Daniel for reading (nearly) all of my thesis and all the helpful feedback, to Jan for preparing elaborate comments despite writing his own thesis at the same time, and to Marc for checking the math in Chapter 3.

The results in this thesis are based on joint work with Marc Bury, Dr. Dan Feldman, Hendrik Fichtenberger, Dr. Christiane Lammersen, Chris Schwiegelshohn and Prof. Dr. Christian Sohler. Thank all of you for the superb collaborations.

My thanks for fruitful discussions and a nice working atmosphere go to all my colleagues in Dortmund. While writing this thesis, my colleagues helped me out to reduce the additional workload, in particular Chris and Marc. Christiane always had the right words to cheer me up, both while we were working together but still when she had already left. I thank Chris, Christiane, Christine and Marc for their friendship and support. I also owe thanks for their encouragement to Dr. Thomas Jansen and Prof. Dr. Petra Mutzel.

Besides working on the content of this thesis, I had a wonderful time doing research on network flows with Daniel, Jan and Martin. Thanks belong to the three of you and to Prof. Dr. Martin Skutella and his group for welcoming us in Berlin and making this research project even more fun. Additional cooperations outside of my thesis topic were with Dr. Frank Hellweg and Dr. José Verschae. Thank you for the great time and fruitful collaboration.

I thank my parents and my sisters Katharina and Magdalena for their love and support. To my husband Daniel I owe thanks for more reasons than I can name here.

Finally, I wish to document my thankfulness to Prof. Dr. Ingo Wegener. Ingo died before I started to work on my PhD, but he is the reason why I did it. His memory accompanies me and I am very thankful for that.

Contents

1 Introduction

Summarizing is a natural concept. Every encyclopedia contains the population size, the gross domenstic product or the area of a country. Events are summarized in end-of-the-year reviews, books and movies are reviewed in magazines or on websites. Even this thesis contains a short summary of its contents.

We summarize data for different reasons. In a condensed form, it is easier to grasp the essentials of a topic. A summary structures data and makes it more accessible, easier to understand. The next reason is that we want to save space. In this context, summarizing means compressing. Most digital music files are stored in a compressed way to save disk space. In the context of algorithm design, there is another important aspect. Reducing the amount of input data decreases the running time of most algorithms. This is particularly true if the data is reduced to an amount that fits into the main memory, since reading data from the hard disk has significant impact on the running time. Algorithms falter because of the sheer amount of time it takes to just read the data. Lastly, there are also cases where we summarize because we simply cannot store the original data. For example, consider the data from the large hadron collidor which is so vast that it is directly filtered. Only a small fraction of the data is actually stored.

Good methods to compress data are increasingly important since we face more and more data that is automatically generated, not only from experiments in physics, but also from social media. This phenomenon has been called Big Data lately. We need new methods to compress data in a meaningful way in order to deal with terabytes and soon petabytes of data.

This thesis studies the concept of *coresets*. A coreset is defined in the context of an objective function. It summarizes input data with respect to this objective function. Notice that a summary can be good for one purpose and bad for another. A summary can never be similar to the original data in *every* aspect. It is sufficient if it resembles the data with respect to the intended use. When compressing music, we want to preserve everything that the human ear can perceive. When we compress input data for an optimization problem, we want that the summary has the same properties according to the given objective function.

The optimization problems studied in this thesis are geometric *clustering* problems. Clustering is a summary method in its own right. It means that we partition input data into subsets of points that we believe belong together. For geometric clustering, we measure whether points belong together by similarity or dissimilarity measures. A dissimilarity measure for Euclidean points would be the Euclidean distance. Assume that we partition points into subsets such that points in the same subset are close together, and points in different subsets are far away from each other. Then we can replace each subset by a representative point and obtain a smaller data set.

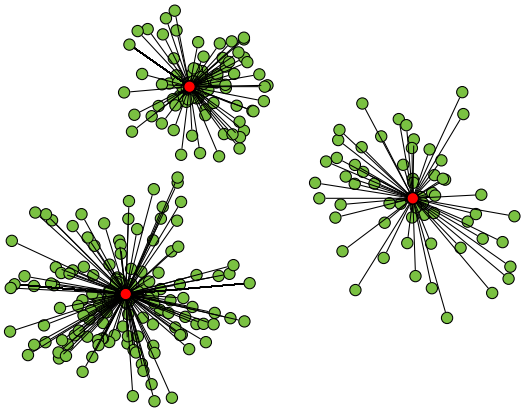


Figure 1.1: A two-dimensional input to the k-means problem and three well chosen centers. The lines indicate which point is assigned to which center.

One of the most widely studied clustering problems is the k-means problem. It consists of the following basic geometric question: Given a set P of points from the Euclidean space $\mathbb{R}^d$, find k centers that minimize the sum of the squared distances from every point to its closest center. A two-dimensional example for the k-means problem is depicted in Figure 1.1. We can imagine that the centers are potential locations for new stores that a big company will open, and that they want to minimize the customer's way to the nearest shops. Instead of shops it could also be telephone or internet exchange points, schools or public swimming baths[1].

The k-means problem has the nice property that for $k = 1$, the solution is the *centroid* of the point set, which is given by an explicit formula as the sum of the points divided by the number of points. This analytic solution to the 1-means problem implies that in an optimal solution for the k-means problem with $k > 1$, the input point set is partitioned into k subsets and all points in the same subset have the same center, namely the centroid of the subset. Using this observation, the k-means problem can be solved exactly by iterating through all partitionings of P into k subsets, computing the centroids of the subsets and keeping track of the solution with lowest cost. Of course this algorithm is highly inefficient. In fact, it turns out that the k-means problem is NP-hard even for $k = 2$ [ADHP09].

The most popular algorithm for solving the k-means problem is Lloyd's algorithm [Llo82], a local improvement heuristic developed more than fifty years ago. Lloyd's algorithm starts with an initial solution of k centers, often chosen uniformly at random from the input point set. Then it calculates the partition induced by the center set by assigning each point to

[1]Intuitivelly, one might also model the above examples with non-squared distances. In this case, the problem is called *k-median* problem. Both problems have many applications. The k-means problem punishes larger distances more severely, which often makes a lot of sense: A slightly larger average way to school is certainly preferable to some children living in an insurmountable distance. We discuss differences between k-means and k-median in more detail later.

its closest cluster, thus forming k point sets. Because the optimal solution for a point set is its centroid, the algorithm replaces the center belonging to each point set by the centroid of the point set. This can only improve the solution. The new centers induce a new partitioning, and so the algorithm iterates until convergence.

There are also various approximation algorithms known for the k-means problem, and many of the newer algorithms are based on the computation of coresets. A coreset for the k-means problem for an input set P is a (smaller) weighted set S with the property that for each choice of k centers, the sum of the squared distances between the points in P and the centers is approximately the same as the sum of the squared distances between the weighted points in S and the centers. Notice that a coreset does not only preserve the cost of an optimal or nearly optimal solution. It approximates the cost function for *every* choice of centers. This is very convenient. When we run an algorithm on S, it will behave similarly as it would when run on P. Coresets have a theoretical guarantee, in fact, it is ensured that the cost of any solution is $(1+\varepsilon)$-approximated by the coreset. Thus, we can use coresets to speed up approximation algorithms at the expense of an additional $(1+\varepsilon)$-factor in the quality guarantee.

Coresets can also be used for distributed computations or when data is given in a streaming setting. The latter means that the data can only be read once, in a given order that cannot be influenced by the algorithm. The constant production of data by the large hadron collidor is a prime example for a streaming setting. The algorithm cannot store all of the data but only a small fraction of it.

The topic of this thesis is the study of coresets and related summaries. The thesis has two parts. The first part is devoted to the k-means problem, while the second part discusses additional clustering objectives.

Chapter 2 introduces the k-means problem, gives a brief outline of its history, its approximability, and its complexity. The k-median problem, which is closely related to the k-means problem, is also introduced. Furthermore, the chapter contains a short introduction to Euclidean geometry and basic techniques for the development of algorithms in the context of the k-means problem.

Chapter 3 addresses dimensionality reductions for the k-means problem. If data is high dimensional, summarizing it can also mean that we reduce the dimension instead of the number of points. The chapter contains a review of methods that reduce the dimension of the points but preserve the k-means objective function up to a factor of $(1+\varepsilon)$. Then, a new method is presented that is able to reduce the dimension of an input point set to $\mathcal{O}(k/\varepsilon^2)$ while the k-means objective is changed by at most an ε-faction.

Chapter 4 introduces coresets for the k-means problem, reviews their history and known coreset results. Additionally, two important methods to construct coresets are discussed. One of them is sampling. The chapter concludes with two new coreset constructions. Both

have the property that the number of points in the coreset is independent of the number of input points and it is also independent of the dimension of the input points.

Chapter 5 discusses the computation of coresets in data streams. It starts with an introduction to the streaming setting and explains the Merge-and-Reduce technique which can be used to embed coreset constructions into streaming. Then, existing streaming algorithms for the k-means problem are reviewed, and one of the coreset constructions from Chapter 4 is embedded into Merge-and-Reduce. The chapter concludes with the presentation of BICO, a coreset construction for the k-means problem in the streaming setting which has a theoretical guarantee but is also easy to implement and fast in praxis.

Chapter 6 is the first chapter of the second part of the thesis. It considers the kernel k-means problem which is a generalization of the k-means problem for instances that are not linearly separable. One of the constructions from Chapter 4 is observed to work in so called inner product spaces. This insight is then used to construct coresets for the kernel k-means problem.

Chapter 7 considers a probabilistic clustering problem. Here, points can appear at different positions within a given metric space. Coresets are constructed for a probabilistic k-median problem, both in the Euclidean space and in finite metric spaces. The chapter also contains approximation results for both problems.

Chapter 8 introduces projective clustering problems. This generalization is the task to cluster points with affine subspaces. Notice that points are affine subspaces of dimension zero. A coreset construction for clustering with one j-dimensional subspace is presented, followed by a coreset construction for clustering with k subspaces of dimension j.

The results in this thesis arose from collaborations with Marc Bury, Dan Feldman, Hendrik Fichtenberger, Christiane Lammersen, Chris Schwiegelshohn and Christian Sohler. Chapters and sections which present original research refer to the publications and name the co-authors. For all publications with n authors, I have contributed at least $1/n$ of the work.

Part I

The k-means problem

2 Introduction to the k-means problem

Clustering is an old[1] and broad topic and the term itself is hard to define concisely. Jain [Jai10] states that the goal of clustering is 'to discover the *natural* grouping(s) of a set of patterns, points or objects' and that the aim of cluster analysis is to find an 'automatic algorithm' to do so. This statement describes the inherent motivation of clustering, and it tells us that a clustering partitions an input into subsets, also called *clusters*.

Clustering is an *unsupervised* method, meaning that it tries to 'discover' the structure hidden in the data without external guidance. This differentiates clustering from supervised learning methods, where parts of the data include class labels.

Given Jain's broad notion of clustering, it is natural that clustering has applications in many very different areas. Indeed, Jain [Jai10] names image segmentation / computer vision, information access and retrieval, grouping customers for efficient marketing, workforce management and planning, and the study of genome data in biology as examples. We can imagine that clustering plays a role whenever objects shall be grouped into similarly behaving subgroups, whenever structure is searched for in not yet understood data, and whenever large data sets need to be compressed into a set of meaningful representatives.

The different application areas also mean that there is not 'the' clustering method. Consider Figure 2.1 which shows an example by Jain (approximately reproduced from [Jai10]). Jain uses it as a visualization of different cluster shapes and densities and states that 'Although these clusters are apparent to a data analyst, none of the available clustering algorithms can detect all these clusters.'

Even more, we notice that this example also illustrates that the 'natural' clustering (and thus, the right choice of the clustering objective) often lies in the eye of the beholder. Depending on the application and the overall structure of the data, we might well consider the orange, gray and purple cluster as one cluster. Consider Figure 2.2 which shows two point sets including these these three sets in a different context. In the right picture of Figure 2.2 we would probably indeed identify these three point sets as different clusters. However, in the left picture we might favour to see them as one cluster.

Because of the large number of different applications and thus also large number of different goals, various clustering problems and algorithms exist in the literature. Several

[1] Hansen and Jaumard [HJ97] claim that clustering dates back to Aristotle. In fact, in his text 'Categories' [Ari], Aristotle derives ten categories to classify objects and concepts, which shows an understanding of the concept of partitioning options due to their properties. Additionally, the Stanford Encyclopedia of Philosophy [Len11] also names Aristotle the 'originator of the scientific study of life' because 'his zoological writings provide a theoretical defense of the proper method for biological investigation; and they provide a record of the first systematic and comprehensive study of animals', which is based on several of his other writings. While discussing whether Aristotle really worked on clustering is besides the point, we conclude that the desire to classify and categorize is indeed an old one.

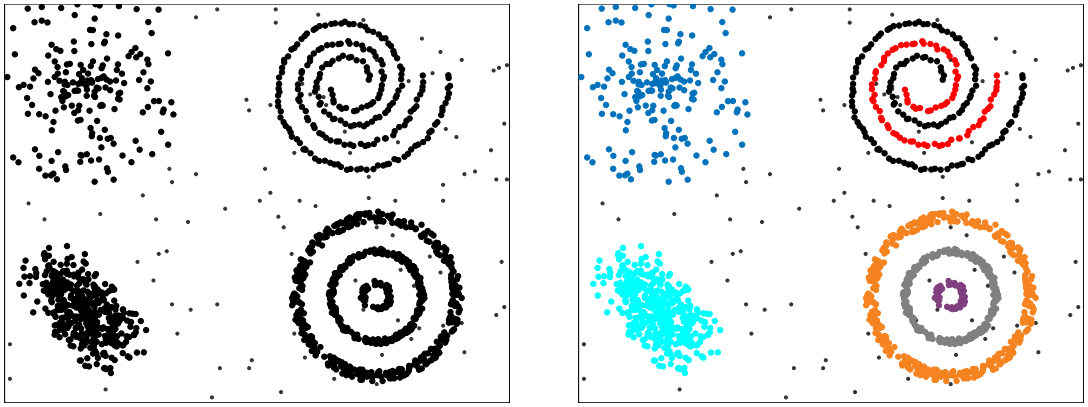


Figure 2.1: Example reproduced after Figure 2 in [Jai10]. The left side presents the input. In the right picture, the points are colored according to the desired clustering.

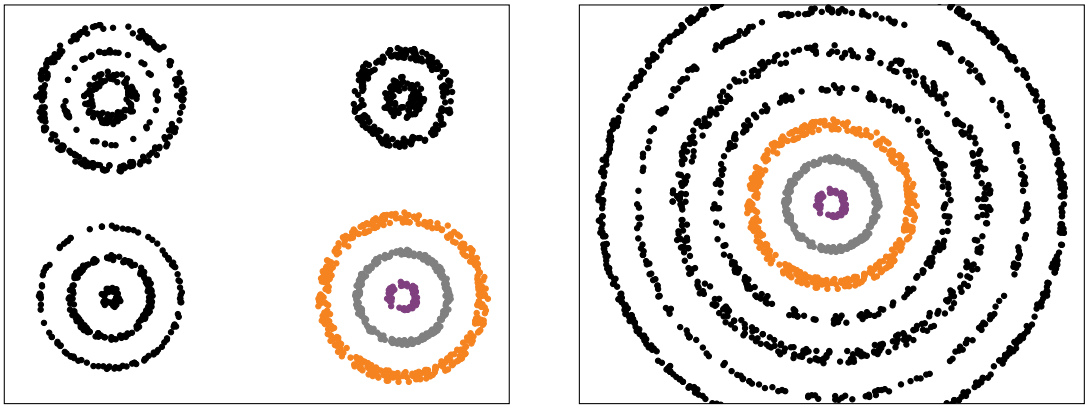


Figure 2.2: Two examples including the orange, gray and purple cluster from Figure 2.1.

surveys and books have been written to summarize and classify the different approaches. These include earlier books by Anderberg [And73] in 1973, by Hartigan [Har75] in 1975 and by Jain and Dubes [JD88] in 1988, newer publications like the review by Jain, Murty and Flynn [JMF99] in 1999 or the book by Everitt, Landau and Leese [ELL09] (first edition 2001), and even more recent publications by Xu and Wunsch [XW05] in 2005, Berkhin [Ber06] in 2006 or the data clustering survey by Jain [Jai10] in 2010. The latter survey has already been cited more than 850 times (according to Google Scholar [Goo14], accessed on 31st of January 2014)[2], which highlights the popularity of the topic even more.

While there is no commonly agreed upon specification, introductions to the field usually ask of a clustering that objects are only grouped together if they are in some way similar. For example, Berkhin [Ber06] says that 'clustering is a division of data into groups of similar objects'. Often, it is also required that objects that are not grouped together

[2]On the 2nd of May in 2015, the number of citations has reached 1668 already.

and are thus classified as different, are in some way dissimilar. Xu and Wunsch [XW05] write that 'most researchers describe clusters by considering the internal homogeneity and the external separation.' However, these goals might be hard to achieve simultaneously. The definition currently found on the online encyclopedia Wikipedia [Wik] gives a slight relaxation by demanding of a clustering that 'objects in the same group (called a cluster) are more similar (in some sense or another) to each other than to those in other groups (clusters).' For our purposes, we will keep the following stability-type idea in mind: A clustering is a partitioning of a set of objects into clusters such that every object satisfies that it is in some way more similar to the objects in its cluster than to the objects in other clusters.

There are various ways to specify what we mean by similar, and this depends on the application area. For example, assume we want to cluster strings of a fixed length from a finite alphabet. Then the similarity between two strings could be measured by the number of coinciding characters. If we normalize the similarity measure by dividing by the string lengths and consider binary strings, this measure is called (simple) matching coefficient (see for example Everitt, Landau, Leese [ELL09]). Yet, there are several other similarity measures for binary strings, and following the presentation in [ELL09], we see that this is due to different application areas. The expressiveness of a match can be low if the bit encodes the presence of something rare, for example a rare genetic defect. If the match encodes the absence of this defect, then it is disputable whether this really indicates a significant similarity. So, depending on the actual scenario, different measures are reasonable, and [ELL09] lists six different relatively simple similarity measures for binary strings alone.

Often, *dis*similarity measures provide a more natural modelling. If the objects are coordinates on a city map, then we probably associate similarity with closeness, or, in other words, low dissimilarity. As for similarity measures, there are several ways to define dissimilarities, even if we fix an underlying metric like the Euclidean distance measure. In the introduction, we already got to know the k-means problem, where the dissimilarity of points in the Euclidean space is measured by the Euclidian distance, but squared. This is a very common modelling, and the remainder of this chapter is devoted to the introduction to this problem. In Section 2.2, we look at the k-median problem, the close relative of the k-means problem where the distances are non-squared. Other dissimilarity measures related to the Euclidean space include l_p^q distances which are based on the q-th power of p-norms[3] and Mahalanobis distances[4].

Before we now take a closer look at the k-means problem, one final remark regarding the type of clustering that is done in this thesis. The clustering algorithms in this thesis belong to the field of *partitional* clustering. This means that they look for partitionings of the input data into subsets in a way that optimizes a clustering objective function. For example, algorithms for the k-means or k-median problem can be viewed as partitional clustering

[3]The l_p^q distance of two points $x = (x_1, \ldots, x_d)^T$ and $y = (y_1, \ldots, y_d)^T$ from $\mathbb{R}^d$ is $\left(\sum_{i=1}^d |x_i - y_i|^p\right)^{q/p}$.

[4]For a symmetric positive definite matrix $A \in \mathbb{R}^{d\times d}$, the corresponding Mahalanobis distance of two points x and y from $\mathbb{R}^d$ is $\sqrt{(x-y)^T A(x-y)}$.

algorithms. Every solution (a set of k centers) induces a partitioning (by assigning every point to its closest center) with a certain cost (the sum of the [squared] distances of all points to their centers). This cost is then optimized.

The wide field of clustering contains other approaches as well. For a systematic overview on different approaches and viewpoints, see for example [Ber06] and [JD88]. One popular example is *hierarchical* clustering. Hierarchical clustering computes a nested sequence of partitions, i. e., the input point set is subsequently subdivided, usually until a level where all subsets are singletons (only contain one point). The subdivision is done according to a splitting criterion based on similarity. A hierarchical clustering algorithm can also start with the set of singletons and subsequently merge these based on a similarity merging criterion. Our only brief contact with hierarchical clustering will be in Chapter 5.3 because the algorithm BIRCH described there uses a hierarchical clustering algorithm.

2.1 The k-means problem

Now, we consider the objective function on which the 'most commonly used partitional clustering strategy' [JD88] is based upon. The k-means objective function seeks to minimize the sum of the squared distances of all points to a set of k centers. It is also known as *square error criterion* or *sum of squared errors*, usually abbreviated by SSE.

The squared Euclidean distance is an old clustering objective. Bock [Boc07] traces the k-means problem back to 1950 to a paper by Dalenius [Dal50]. The still most popular algorithm for the k-means problem was already developed in 1956 by Steinhaus [Ste56] and independently in 1957 by Lloyd [Llo57]. The immense popularity of the k-means cost function becomes most apparent in the popularity of this algorithm. We discuss it below.

The k-means problem is specified with a *dis*similarity measure, the Euclidean distance. We will review the definitions and notations in Euclidean geometry below, for now it is only important that we denote the Euclidean space by $\mathbb{R}^d$ and the Euclidean norm by $||\cdot||$. In addition to the k-means objective function, we also define the k-median objective function. Firstly, we will consider (a probabilistic version of) the k-median problem in Chapter 7. Secondly, we will occasionally use the k-median problem for comparisons of properties of the k-means problem. We also define a popular variant of both problems, called *discrete* clustering.

Definition 2.1.1 (k-means and k-median)**.** *Let $P \subset \mathbb{R}^d$ be a finite set of points in the d-dimensional Euclidean space, let $k \in \mathbb{N}$ be positive integer. The* (Euclidean) k-median problem *is to find a set $C \subset \mathbb{R}^d$ of k points (called* centers*) that minimizes the sum of the Euclidean distances of the points in P to their closest center in C, $\sum_{x \in P} \min_{c \in C} ||x - c||$. The* (Euclidean) k-means problem *is to find a set $C \subset \mathbb{R}^d$ of k points that minimizes*

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

the sum of the squared Euclidean distances of the points to their closest center.

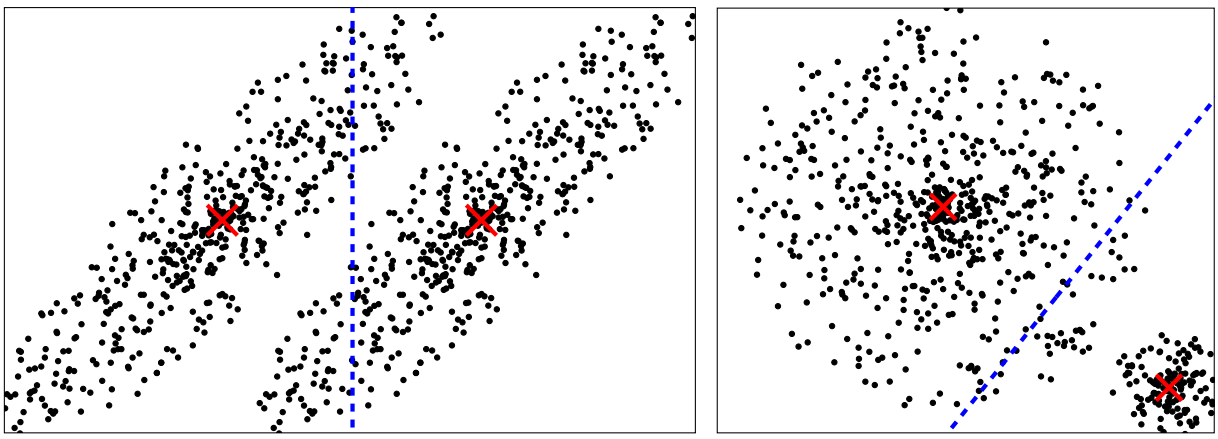


Figure 2.3: Examples of point sets that are to be partitioned into two clusters. The red points are chosen to lie at the center of the intuitive clusters. The blue lines show the partitioning created by assigning each point to its closest center. This partitioning is probably not the intuitive one.

If the point set is weighted by a function $w : P \to \mathbb{R}$, the weighted *k-means problem is the task to minimize $\sum_{p \in P} \min_{c \in C} w(x)||x - c||^2$ over the choice of $C \subset \mathbb{R}^d$ with $|C| = k$. The* discrete *(possibly weighted) Euclidean k-means or k-median problem is verbatim except that it does not optimize over $C \subset \mathbb{R}^d$, but over all choices of C from a finite set, e. g., from the input point set.*

Notice that discrete clustering problems are discrete with respect to the set of centers, not with respect to the input point set. Standard discrete clustering problems use the input point set as the candidate set for the centers. So, if we talk about 'the' discrete k-means or k-median problem, we mean that the candidate set is equal to the input point set. Complementary to the discrete case, k-means and k-median problems with $\mathbb{R}^d$ as the candidate space for the centers are also refered to as *continuous*.

For all these problems, we also call the value of the objective function the *cost* of P with a given center set C or the *optimal cost* of P for the optimal choice of C.

Limitations for finding 'natural' clusterings. As we discussed above, no (known) clustering objective finds every 'natural' clustering. The k-means objective is particularly well-suited for spherical clusters with relatively equal radii. Figure 2.3 shows two examples where with the 'natural' centers, the implicit assignment of points to centers does not lead to the natural clustering. Notice however that both examples show clusters which are very close together compared to their radii. If the clusters are well-spread, an optimal k-means clustering is more likely to coincide with the intuitively expected result.

Additionally, consider Figure 2.2. While the four clusters in the picture on the left side may be found with an algorithm for the k-means problem, the seven clusters in the picture on the right are impossible to be the result of a k-means optimization. The reason is that

they are not linearly separable – there is no way to define centers that partition the two-dimensional space into rings. In Section 6.2, we will see an extension of k-means designed for this type of data.

Finally, notice that for optimizing the k-means clustering function, the value of k must be known in advance, or, if we want to find it (for example by binary search), an additional quality measure of the resulting clustering is needed. The k-means objective is monotonically decreasing in the number of clusters. Many approaches exist to determine a good number of clusters. Section 5 in the paper by Tibshirani, Walther, and Hastie [TWH01] and the summary by Gordon [Gor96] can serve as a starting point. One idea is to look for a significant drop in the cost followed by a relatively stable cost (the *elbow method*). So if there exists a number of centers k where the optimal k-means cost drops unproportionally compared to the optimal $(k-1)$-means cost, but does not continue to decrease significantly when considering $k+1$ optimal centers, the 'right' number of clusters might have been found.

Lloyd's algorithm. The most popular algorithm for the k-means problem is Lloyd's algorithm. It is a local improvement strategy. Starting with an arbitrary solution consisting of k centers from $\mathbb{R}^d$, two steps are iterated until convergence is reached or until a stopping criterion is satisfied. First, every input point is assigned to its closest center, ties broken arbitrarily. This results in a partitioning of the point set into k clusters. Second, the optimal 1-means solution is computed for each subset in this partitioning. The optimal solution for $k=1$ is the centroid (which is the sum of the points divided by their number), so the 1-means solutions for the k subsets can be computed in time $\mathcal{O}(ndk)$. The centers are then replaced by the k centroids and the algorithm continues with these new centers.

When we consider the sum of the squared distances of all points to the center that they are assigned to, both steps can only decrease this sum. To see this, notice that computing the centroids while keeping the partitioning can only decrease the cost because the centroid is the best 1-means solution. Reassigning the points to their now closest center can again only decrease the cost. There are only finitely many ways how a point set of n points can be partitioned, thus the algorithm will eventually converge to a situation where the reassignment of the points results in the same partitioning as before. Then, the algorithm stops.

Due to its popularity, Lloyd's algorithm is actually often called *k-means algorithm*. We avoid this naming because the term was first used by MacQueen [Mac67] to name a different algorithm for the k-means problem, causing some potential for confusion, and because the reference to Lloyd is also common. The algorithm was, however, independently discovered by Steinhaus [Ste56] even before Lloyd described it in 1957 [Llo57]. According to Bock [Boc07], referring to Lloyd is common in 'computer science and pattern recognition communities'. Both Steinhaus and Lloyd actually considered a continuous version of the k-means problem and thus also of the algorithm. According to Bock, the first to propose the discrete algorithm was Forgy as part of a lecture which is documented in other publications (the usual reference [For65] is the abstract of this talk).

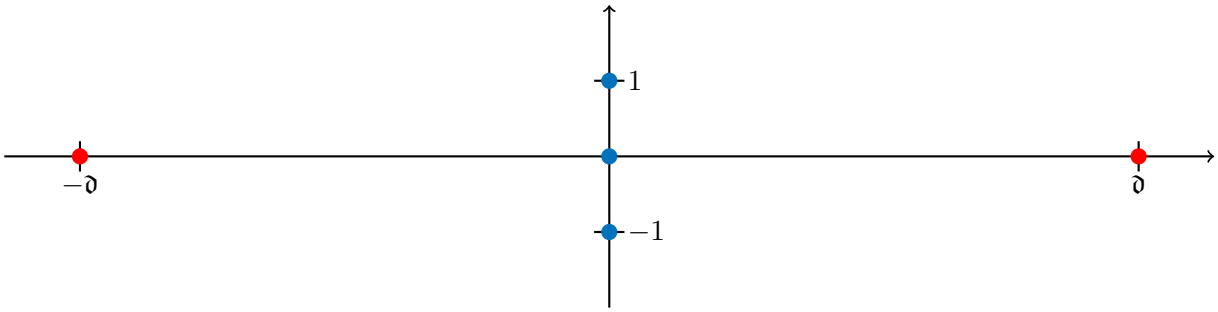


Figure 2.4: An example with five points where the blue points form a local but not global optimum for $k = 3$.

Lloyd's algorithm is over fifty years old. It is probably indeed 'the best-known' algorithm for the k-means problem as Xu and Wunsch [XW05] call it, has been titled 'the most commonly used partitional clustering strategy' [JD88] and the 'by far most popular clustering tool used in scientific and industrial applications' [Ber06]. The recent survey by Jain [Jai10] again certifies that 'it is still one of the most widely used algorithms for clustering', despite its age. Lloyd's algorithm was also named one of the ten most influential algorithms in the data mining community after a three-stage recommendation process by the organizers of the IEEE International Conference on Data Mining (ICDM), as documented in the publication by Wu et. al. [WKQ+08].

The enormous popularity of Lloyd's algorithm might be due to its simplicity (both steps are easy to understand and implement) and the experience that it is often fast (if we simply stop after running a constant number of iterations, the running time is $\mathcal{O}(nkd)$) while computing reasonable solutions, or because of the lack of an always convincing alternative, as Jain [Jai10] implies.

However, the algorithm does not come with a polynomial running time guarantee or a guarantee on the quality of computed solutions. After Dasgupta [Das03] and Har-Peled and Sadri [HPS05] raised the question on the running time of Lloyd's algorithm and gave first bounds, Arthur and Vassilvitksii [AV06] and Vattani [Vat11] showed that the number of necessary iterations can be exponential in the number of points, and this can even happen for points in $\mathbb{R}^2$.

When the algorithm converges, the result is a local optimum, but it is not guaranteed to be a global optimum or an approximation of one. Figure 2.4 depicts a five point example for this described by Mettu and Plaxton [MP04]. The coordinates of the blue points are $(0, -1)$, $(0, 0)$ and $(0, 1)$, the coordinates of the red points are $(-\mathfrak{d}, 0)$ and $(\mathfrak{d}, 0)$. If Lloyd's algorithm is initialized with the blue points, then assigning every point to its closest centers and computing the centers again yields the blue points as centers. The optimal solution would be to pick $(-\mathfrak{d}, 0)$, $(0, 0)$ and $(\mathfrak{d}, 0)$ as centers, which costs 2. The local optimum has a cost of $2 \cdot \mathfrak{d}^2$, which can be arbitrarily large.

The k-means++ algorithm. As already apparent in the above example, the initialization of Lloyd's algorithm is crucial for the quality of the solution. Among the many methods to find good initial centers, we discuss one method in more detail. In 2009, Arthur and Vassilvitskii [AV07] proposed the algorithm *k-means++*, which consists of a clever random initialization process followd by Lloyd's algorithm. The centers are chosen iteratively. The first one is drawn uniformly at random from the point set. In each of the remaining $k-1$ steps, the centers are chosen according to a probability function based on the current cost of the points. More precisely, the probability to choose a point is the squared distance to its closest center divided by the sum of the squared distances of all points to their closest center. The intuitive idea behind this procedure is to choose points which have a high cost in the current solution. If a cluster of points lies very far away from any so far chosen center, it is probably a good idea to pick one of its points. The same holds if a cluster is only medium far, but contains a lot of points. By choosing the probability distribution based on the current cost of the points, both of these scenarios are covered at the same time (in contrast to for example simply picking the currently most expensive point).

Arthur and Vassilvitskii show that the k centers chosen by this random procedure are themselves a $\mathcal{O}(\log k)$-approximative solution. Applying Lloyd's algorithm enhances the solution further, providing a local optimum of at least the same quality.

The distinctive feature of k-means++ is that it is an easy algorithm and practically fast, but it comes with an approximation guarantee, even though this guarantee is high.

Complexity of the k-means problem. As the popularity of a potentially exponential algorithm already indicates, the k-means problem is NP-hard. It has two parameters, and it is a natural approach to check whether fixing these to constants influences the hardness of the problem. The parameters are the number of centers k and the dimension d of the input points. We might ask if fixing one of them and applying some sort of enumeration strategy can yield a polynomial algorithm. Notice that we cannot easily enumerate all possible centers as there are infinitely many possibilities. In Section 2.4, we see that the 1-means problem can be solved analytically. We also see that this implies that we can solve the k-means problem by iterating through all partitions of the input point set, computing the 1-mean of every subset and keeping track of the best solution. However, this enumeration algorithm is not polynomial, not even for constant k. In fact, if the dimension of the problem is arbitrary, then the k-means problem is NP-hard even for $k = 2$. A short proof for this is due to Aloise, Deshpande, Hansen and Popat [ADHP09], a proof containing a sequence of reductions is due to Dasgupta [Das08]. The motivation for their work was an error in a previous proof due to Dasgupta, Frieze, Kannan and Vempala [DFK+04]. The k-means problem is also NP-hard for constant dimension and arbitrary k by the work of Mahajan, Nimbhorkar, and Varadarajan [MNV09], more precisely, even for $d = 2$. For constant dimension and a constant number of centers, the problem can be solved in polynomial time by the algorithm of Inaba, Katoh, and Imai [IKI94]. It is indeed an enumeration algorithm, iterating through all possible partitions which can be induced by weighted Voronoi diagrams.

Authors	Year	Guarantee	Running Time	Reference
k and d are arbitrary				
Jain, Vazirani	1999	≤ 108	$\text{poly}(n, d, k)$	[JV01]
Kanungo, Mount, Netanyahu, Piatko, Silverman, Wu	2002	$\mathcal{O}(1)$	$\text{poly}(n, d, k)$	[KMN+04]
Mettu, Plaxton	2002	$\mathcal{O}(1)$	$\mathcal{O}(ndk)$ for polynomial coordinates and $k \in [\log n, n/\log n]$	[MP04]
Arthur, Vassilvitskii	2007	exp. $\mathcal{O}(\log k)$	$\mathcal{O}(ndk)$	[AV07]
Aggarwal, Deshpande, Kannan	2009	$\mathcal{O}(1)$	$\mathcal{O}(ndk) + \text{poly}(k, \log n)$	[ADK09]
k is arbitrary, d is a constant				
Kanungo, Mount, Netanyahu, Piatko, Silverman, Wu	2002	$9 + \varepsilon$	$\omega(n\varepsilon^{-d} \ln 1/\varepsilon)$	[KMN+04]
k is a constant, d is arbitrary				
Drineas, Frieze, Kannan, Vempala, Vinay	1999	2	$\mathcal{O}(n^{k^3/2}) + \text{poly}(d, n)$	[DFK+04]
Ostrovsky, Rabani	2000	$1 + \varepsilon$	$\mathcal{O}(n^{(k+\varepsilon^{-1})^{\mathcal{O}(1)}})$	[OR02]
de la Vega, Karpinski, Kenyon, Rabani	2003	$1 + \varepsilon$	$\mathcal{O}(n(\log^k n) \cdot \varepsilon^{-8} \ln \varepsilon^{-1})$, precomputed distances	[dlVKKR03]
Kumar, Sabharwal, Sen	2004	$1 + \varepsilon$	$\mathcal{O}(nd \cdot 2^{(k/\varepsilon)^{\mathcal{O}(1)}})$	[KSS10]
Chen	2006	$1 + \varepsilon$	$\mathcal{O}(nd + 2^{(k/\varepsilon)^{\mathcal{O}(1)}} d^2 \log^{k+2} n)$	[Che09]
Feldman, Monemizadeh, Sohler	2007	$1 + \varepsilon$	$\mathcal{O}(nd) + d \cdot \text{poly}(1/\varepsilon) + 2^{\tilde{\mathcal{O}}(k/\varepsilon)}$	[FMS07]
Feldman, Langberg	2011	$1 + \varepsilon$	$\mathcal{O}(nd + 2^{\text{poly}(1/\varepsilon, k)})$	[FL11b]
Jaiswal, Kumar, Sen	2012	$1 + \varepsilon$	$\mathcal{O}(nd \cdot 2^{\tilde{\mathcal{O}}(k^2/\varepsilon)})$	[JKS14]
k is a constant, d is a constant				
Hasegawa, Imai, Inaba, Katoh	1993	2	$\mathcal{O}(n^{k+1})$	[HIIK93]
Inaba, Katoh, Imai	1994	1	$n^{\mathcal{O}(dk)}$	[IKI94]
Matoušek	2000	$1 + \varepsilon$	$\mathcal{O}(n \log^k n \varepsilon^{-2k^2 d})$	[Mat00]
Har-Peled, Mazumdar	2004	$1 + \varepsilon$	$\mathcal{O}(n + \log^9 n + \varepsilon^{-(2d+1)k} \log^{k+1} n \log^k(1/\varepsilon))$	[HPM04]
Effros, Schulman	2004	$1 + \varepsilon$	$(1/\varepsilon)^{\mathcal{O}(d)} n \log\log n + (1/\varepsilon)^{\mathcal{O}(kd)}$	[ES04]
Frahling, Sohler	2005	$1 + \varepsilon$	$\tilde{\mathcal{O}}(n \log^3 \Delta + \log^9 n + \varepsilon^{(-2d+1)k})$	[FS05]
Har-Peled, Kushal	2005	$1 + \varepsilon$	$\mathcal{O}(n + \text{poly}(\log n, 1/\varepsilon) + f(\varepsilon))$	[HPK07]

Table 2.1: A list of several approximation algorithms for the k-means problem.

The NP-hardness naturally raises the question whether approximation algorithms exist. To the best knowledge of the author of this thesis, the status of current reseach is the following. For constant k and constant d, the problem is polynomially solvable and polynomial approximation schemes serve only to speed up the running time. Matoušek was the first to propose such a PTAS in [Mat00]. Constructing a PTAS is still possible for arbitrary d and constant k, and the first PTAS in this scenario was given by Ostrovsky and Rabani [OR02]. In this case, even linear time $(1+\varepsilon)$-approximation schemes are possible, and the first such scheme was given by Kumar, Sabharwal and Sen [KSS10].

If k is arbitrary, no $(1+\varepsilon)$-approximation is known, not even for constant dimension. It is also not known whether the problem is APX-hard. Constant approximations are possible, even if k *and* d are variable. The first such approximation was given by Jain and Vazirani [JV01] who proposed a primal-dual algorithm for the k-median problem which can be extended to the k-means problem. They give 108 as a rough upper bound on the approximation guarantee, but state that a much better guarantee is achievable. Kanungo, Mount, Netanyahu, Piatko, Silverman and Wu [KMN+04] propose an approximation based on a local search algorithm proposed in [AGK+04] for the k-median problem. Their algorithm can be used in two ways. In a basic version, it achieves an approximation a little better than Jain and Vazirani, but is also applicable in the scenario of variable d and k. In a refined version, the approximation guarantee is improved to $9+\varepsilon$, but the running time is only polynomial for constant dimension d. However, a slight change to the algorithm most likely reduces the running time to be polynomial in d, too.

Table 2.1 gives an overview including many important steps in the development of faster k-means approximation algorithms. It is meant as a (probably not complete) look-up table for different approximation algorithms, giving an impression of the variety of available approximation algorithms for the k-means problem.

Remarks on Table 2.1. Notice that the algorithm by Inaba, Katoh and Imai [IKI94] is deterministic, but most of the cited algorithms are randomized. In this case, the running times are stated under the assumption that random numbers can be drawn in $\mathcal{O}(1)$. Generally, the running times do not include terms which are assumed to be constant. The algorithms developed in [Che09, FS05, FMS07, FL11a, HPM04, HPK07] involve the development of coresets and were designed for data streaming settings. We discuss coresets in Chapter 4, and data streams in Chapter 5.

The algorithm by Frahling and Sohler [FS05] was developed for *geometric data streams*, where the assumption is that the input points lie on the grid $\{1,\dots,\Delta\}^d$ for a constant Δ. Such a grid can always be found by scaling and translating the points, then Δ has to be at least the spread of the input point set, i. e., the quotient of the maximum and minimum distance of all input points.

The algorithms by Jain and Varadarajan [JV01] and Mettu and Plaxton [MP04] were originally developed for the k-median problem but were later noted to work for the k-means problem, too.

The running time of the algorithm by Kanungo et. al. [KMN+04] is cited as $n^3\varepsilon^{-d}$ by

[AV07]. It is definitely exponential in d because it uses a so-called centroid set due to Matoušek [Mat00] as a candidate set for centers instead of using $\mathbb{R}^d$. This induces only a $(1+\varepsilon)$-error. Using the algorithm without the centroid set and instead using P as the candidate set for centers induces a higher approximation ratio, but yields an algorithm with polynomial dependence on d. For a more detailed analysis of such a version of the local search algorithm, also see [SR10], where an upper bound of 50 for the approximation guarantee is given.

The approximation guarantee of the algorithm by Arthur and Vassilvitskii [AV07] is on expectation, so a single run of the algorithm may return a worse result.

The table only lists results that were explicitly stated for the k-means problem and are not bicriteria approximations. Notable additional results include the generic sampling algorithm that can be used to speed up approximation algorithms and was analyzed by Mishra, Oblinger and Pitt [MOP01] and later by Czumaj and Sohler [CS04], and which is one basis of the algorithm by [Che09]. We discuss it in more detail on page 72 in Section 4.2.3.

Organization of the remainder of this chapter. In Section 2.2, we shortly review results on the very related k-median problem. After that, we look at useful observations that help solving the k-means problem. Our choices are subjective and motivated by the topics that are considered in later chapters. Section 2.3 could also be titled 'Preliminaries' as it contains definitions and notations that are later needed. In Section 2.4, we introduce a 'folklore' finding from linear algebra. Despite its simplicity, it has had a huge impact on the study of the k-means problem and on the design of algorithms for it. We also see a few of its direct consequences.

2.2 The k-median problem

We consider the two most common k-median problems. For us, 'the' Euclidean k-median problem is the task to compute k centers from $\mathbb{R}^d$ such that the sum of the non-squared distances from each input point to its closest center is minimized. It is closely related to the k-means problem.

For the k-median problem, it is also very common to consider finite metric spaces, i. e., a set $\mathbf{X}$ with a distance function $d : \mathbf{X} \times \mathbf{X} \to \mathbb{R}^+$ that satisfies the triangle inequality, is non-negative and symmetric (and $d(x, y)$ is only zero iff $x = y$). We formally define metrics in Section A.1. For k-median problems in finite metric spaces, the input point set $\mathcal{P} \subset \mathbf{X}$ is a subset of the metric space. By 'the' metric k-median problem we mean the problem to compute k centers from $\mathcal{P}$ that minimize the sum of the distances of all points in $\mathcal{P}$ to its closest center. So, centers can only be chosen from the input point set.

Both versions are well-studied, and they are quite related to the k-means problem. The k-means counterparts of some of the results discussed below are also mentioned in Section 2.1 and in particular in Table 2.1.

Authors	Year	Guarantee	Running Time	Reference
Bartal	1996	$\mathcal{O}(k \log n \log_k n)$	$\text{poly}(n, k)$	[Bar96]
Bartal	1998	$\mathcal{O}(\log n \log \log n)$	$\text{poly}(n, k)$	[Bar98]
Charikar, Chekuri, Goel, Guha	1998	$\mathcal{O}(\log k \log \log k)$	$\text{poly}(n, k)$	[CCGG98]
Charikar, Guha, Tardos, Shmoys	1999	$6\frac{2}{3}$	$\text{poly}(n, k)$	[CGTS02]
Jain, Vazirani	1999	6	$\tilde{\mathcal{O}}(n^2)$	[JV01]
Charikar, Guha	1999	4	$\tilde{\mathcal{O}}(n^3)$	[CG05]
Arya, Garg, Khandekar, Meyerson, Munagala, Pandit	2001	$3 + \varepsilon$	$\text{poly}(n, k)$	[AGK+04]
Guha, Meyerson, Mishra, Motwani, O'Calaghan	2000	$\mathcal{O}(1)$	$\tilde{\mathcal{O}}(nk)$	[GMM+03]
Mettu, Plaxton	2002	$\mathcal{O}(1)$	$\mathcal{O}(nk)$ for $k \in [\log n, n/\log n]$	[MP04]
Charikar, O'Callaghan, Panigrahy	2003	$\mathcal{O}(1)$	one-pass streaming algorithm	[COP03]
Chen	2006	$10 + \varepsilon$	$\mathcal{O}(nk + k^7 \varepsilon^{-5} \log^5 n)$	[Che09]
Li, Svensson	2013	$1 + \sqrt{3} + \varepsilon$	$\mathcal{O}(n^{\varepsilon^{-2}})$	[LS13]

Table 2.2: Some approximation algorithms for the metric k-median problem.

Complexity. The metric k-median problem is NP-hard [KH79b]. The Euclidean k-median problem is also NP-hard, even in the plane [MS84]. The variant where the centers can only be chosen from the input points (but the metric space is the Euclidean space) is also NP-hard, as shown by Papadimitriou [Pap81]. Additionally, the Euclidean k-median problem is the Fermat-Weber problem for $k = 1$. As we mentioned before, the Fermat-Weber problem cannot be solved optimally by an algorithm that uses only arithmetic operations and the computation of roots [Baj88].

Notice that the metric k-median problem can be solved in polynomial time by iterating through all possibilities to choose k centers from n input points if k is constant.

Approximation algorithms for the metric k-median problem. For the metric k-median problem, no PTAS is known for the case of arbitrary k, and most likely, none exists. Lin and Vitter [LV92] showed that computing a $(1+\varepsilon)$-approximation to the metric k-median problem is NP-hard. Guha [Guh00] showed that it cannot be approximated within a factor less than $1 + 1/e$ unless it holds that NP $\subseteq$ DTIME$[n^{\mathcal{O}(\log \log n)}]$. The bound was improved by Jain, Mahdian and Saberi [JMS02]. They showed that the metric k-median problem cannot be approximated within a factor less than $1 + 2/e$ unless NP $\subseteq$ DTIME$[n^{\mathcal{O}(\log \log n)}]$.

Table 2.2 gives an overview on some approximation algorithms for the metric k-median problem. The approximation guarantee has been continuously improved. Two particularly noteworthy contributions are the first constant-factor approximation due to Charikar,

Guha, Tardos and Shmoys [CGTS02] and the algorithm with the currently best approximation ratio due to Li and Svensson [LS13], which was developed after a rather long period without an improvement of the best known approximation ratio. The ratio is $1 + \sqrt{3} + \varepsilon$.

We notice two details. First, the constant approximation due to Chen [Che09] uses coresets. The coreset size ist $\mathcal{O}(dk^2\epsilon^{-2} \log n \log(k/\varepsilon))$. Second, Mettu and Plaxton [MP04] developed not only an algorithm but also proved an $\Omega(nk)$ lower bound on the running time of any constant-factor approximation, even for randomized algorithms, which is nearly matched by their running time and the running time of the algorithm due to Chen.

In addition to approximation algorithms like in Table 2.2, there exist several algorithms that compute approximations under relaxed conditions, for example bicriteria approximations which relax the number of centers. For example, a randomized bicriteria approximation by Indyk [Ind99], which is of particular interest to us, computes $\mathcal{O}(k)$ centers that provide a constant factor approximation. The running time is $\tilde{O}(nk/\delta^2)$ where δ is the failure probability of the algorithm. Another example is an algorithm due to Meyerson, O'Callaghan and Plotkin [MOP04]. It is a constant factor approximation with running time $\tilde{\mathcal{O}}(k((k^2/\varepsilon) \cdot \log k)^2)$ under the assumption that there exists an optimal solution where each cluster has at least $\Omega(n\varepsilon/k)$ points for a constant $\varepsilon > 0$.

The algorithms by Guha, Meyerson, Mishra, Motwani and O'Callaghan [GMM+03] and by Charikar, O'Callaghan and Panigrahy [COP03] work in the streaming model. The algorithm in [GMM+03] needs $\mathcal{O}(n^\varepsilon)$ space and computes a $2^{\mathcal{O}(1/\varepsilon)}$-approximation. The algorithm in [COP03] needs $\mathcal{O}(k \log^2 n)$ space.

Approximation algorithms for the Euclidean k-median problem. When the set of candidates is restricted to the input point set, then the Euclidean k-median problem is a special case of the metric k-median problem. By the triangle inequality, optimally solving this discrete version yields a 2-approximation for the Euclidean k-median problem where centers can be chosen from $\mathbb{R}^d$. Consequently, all approximation algorithms in Table 2.2 induce constant approximations for the Euclidean k-median problem with twice the constant.

When either k or d is constant, then the (continuous as well as the discrete) Euclidean k-median problem can be approximated to arbitrary precision. Table 2.3 lists $(1+\varepsilon)$-approximations for the Euclidean k-median problem for either constant d or k.

The first $(1+\varepsilon)$-approximation algorithm is due to Arora, Raghavan and Rao [ARR98]. It is based on the famous result due to Arora [Aro98] on the Euclidean Traveling Salesman problem and works for instances in the Euclidean plane, so the dimension is a constant. Kolliopoulos and Rao [KR07] developed the first $(1+\varepsilon)$-approximation that works in $\mathbb{R}^d$ for constant dimensions $d \geq 2$, but for the discrete version of the problem. Nevertheless, it is the base algorithm used by the coreset based results for constant d.

For constant k, the first PTAS is due to Ostrovsky and Rabani [OR02], followed by the paper that introduced coresets to clustering and which is due to Bădoiu, Har-Peled and Indyk [BHPI02]. The base algorithm for the coreset construction due to Chen [Che09] and Feldman and Langberg [FL11a] is due to Kumar, Sabharwal and Sen [KSS10]. The algorithm by Feldman, Monemizadeh and Sohler [FMS07] is stated to be extendable to

Authors	Year	Running Time	Notes	Reference
k is a constant, d is arbitrary				
Ostrovsky, Rabani	2000	$\mathcal{O}(n^{(k+\varepsilon^{-1})^{\mathcal{O}(1)}})$		[OR02]
Bădoiu, Har-Peled and Indyk	2002	$2^{(k/\varepsilon)^{\mathcal{O}(1)}} d^{\mathcal{O}(1)} n \log^{\mathcal{O}(k)} n$		[BHPI02]
Kumar, Sabharwal, Sen	2004	$\mathcal{O}(nd \cdot 2^{(k/\varepsilon)^{\mathcal{O}(1)}})$		[KSS10]
Chen	2006	$\mathcal{O}(nd + 2^{(k/\varepsilon)^{\mathcal{O}(1)}} d^2 \log^{k+2} n)$		[Che09]
Feldman, Langberg	2011	$\mathcal{O}(nd + 2^{\text{poly}(1/\varepsilon,k)})$		[FL11b]
k is arbitrary, d is a constant				
Arora, Raghavan, Rao	1998	$\mathcal{O}(n^{\mathcal{O}(\varepsilon^{-1}+1)})$	$d = 2$	[ARR98]
Kolliopoulos and Rao	1999	$\mathcal{O}(2^{\mathcal{O}((\log(1/\varepsilon)/\varepsilon)^{d-1})} n \log^{d+6} n)$	discrete	[KR07]
Har-Peled, Mazumdar	2004	$\mathcal{O}(n + k^5 \log^9 n + k^2 \cdot \log^5 n \cdot e^{((1+\log 1/\varepsilon)/\varepsilon)^{d-1}})$		[HPM04]
Frahling, Sohler	2005	$\mathcal{O}(k^5 \log^9 n + k^2 \cdot \log^5 n \cdot e^{\mathcal{O}((1+\log(1/\varepsilon)/\varepsilon)^{d-1})})$		[FS05]
Har-Peled, Kushal	2005	$\mathcal{O}(n + \text{poly}(k, \log n, 1/\varepsilon) + f(k,\varepsilon))$		[HPK07]

Table 2.3: A list of $(1+\varepsilon)$-approximations for the Euclidean k-median problem.

the Euclidean k-median case but it is not listed since the k-median version is not explicitly stated.

2.3 Notations in Euclidean geometry

The Euclidean space $\mathbb{R}^d$ consists of all d-tuples $(x_1, \ldots, x_d)^T$ of values from $\mathbb{R}$, considered as a column vector. Whenever we talk about the k-means problem, we will always use $\mathbb{R}^d$ as the space that the input points lie in, even when not explicitly stating so. We assume that the set $\mathbb{N}$ is the set of all positive integers and does not contain 0. However, we sometimes refer to $\mathbb{N}$ as $\mathbb{N}^{>0}$ to emphasize this fact. Further, we use the abbreviation $[n] := \{1, \ldots, n\}$ for the set of the first n integers in $\mathbb{N}$.

Points and distances. We call the elements in $\mathbb{R}^d$ *vectors* or *points*. Usually, we denote points from $\mathbb{R}^d$ by lower case letters. Point sets in $\mathbb{R}^d$ are denoted by capital letters and we denote the input point set of the problem at hand by $P \subset \mathbb{R}^d$ unless otherwise stated. P is also used whenever we need an arbitrary point set from $\mathbb{R}^d$ for a definition. A vector x from $\mathbb{R}^d$ is a *unit vector* if its length $||x|| := \sqrt{\sum_{i=1,\ldots,d} x_i^2}$ is one. For a given finite point set P, the *centroid* or *mean* of P is defined by

$$\mu(P) := \frac{1}{|P|} \sum_{x \in P} x.$$

For two points $x, y \in \mathbb{R}^d$, we denote their Euclidean distance by

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

We use the notation $\text{dist}(x, y) := ||x - y||$ for the Euclidean distance and $\text{dist}^2(x, y) := (\text{dist}(x, y))^2$ for the squared Euclidean distance. For a finite set P and a point $y \in \mathbb{R}^d$, we sometimes abbreviate $\text{dist}^2(P, y) = \sum_{x \in P} \text{dist}^2(x, y)$. Given a compact point set P, the *diameter* of P is the largest distance between two points in P, and we denote it by

$$\text{diam}(P) := \max_{x,y \in P} ||x - y||.$$

For a non-empty closed (not necessary finite) set $L \subseteq \mathbb{R}^d$ and a point $x \in \mathbb{R}^d$, we abbreviate the smallest (squared) distance between x and any point from L by $\text{dist}(x, L) := \min_{y \in L} \text{dist}(x, y)$ or $\text{dist}^2(x, L) := \min_{y \in L} \text{dist}^2(x, y)$, respectively, and call it the (squared) distance between x and L. Analogously, we denote the (squared) distance between two non-empty and closed sets $L_1, L_2 \subseteq \mathbb{R}^d$ of points by $\text{dist}(L_1, L_2) := \min_{x \in L_1} \text{dist}(x, L_2)$ or $\text{dist}^2(L_1, L_2) := \min_{x \in L_1} \text{dist}^2(x, L_2)$, respectively.

For a point $x \in \mathbb{R}^d$ and a non-negative constant $r \in \mathbb{R}^{\geq 0}$ we denote the set of all points that are within distance r of x by $B(x, r)$, i. e., it is $B(x, r) = \{y \in \mathbb{R}^d \mid ||x - y|| \leq r\}$. We call this set the *sphere* or the *ball* of radius r around x.

Scalar product in $\mathbb{R}^d$. For two vectors $x, y \in \mathbb{R}^d$, the scalar product (or dot product) of x and y is defined by $\langle x, y \rangle := \sum_{i=1}^{d} x_i y_i$. The scalar product is an inner product (see Section A.2) and is also called the *standard* inner product in $\mathbb{R}^d$.

We also use the notation $x^T y := \langle x, y \rangle$. Notice that for a point $x \in \mathbb{R}^d$, the scalar product of x with itself gives the squared length of x, i. e., $\langle x, x \rangle = ||x||^2$.

If the scalar product of two points $x, y \in \mathbb{R}^d$ is zero, then we say that they are *orthogonal*. We call a set of points *orthonormal* if all points are unit vectors and they are pairwise orthogonal. Orthogonal (and in particular orthonormal) vectors satisfy the Pythagorean theorem.

Theorem 2.3.1. *Let $x_1, \dots, x_k \in \mathbb{R}^d$ be $k \leq d$ pairwise orthogonal vectors from $\mathbb{R}^d$. Then it holds that*

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

Proof. Let $k = 2$. Then, we see that

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

The general case follows by induction. □

Bases and subspaces. Let $B = \{b_1, \dots, b_k\} \subset \mathbb{R}^d$ be a finite set of vectors. The *span* of B is the subset $\operatorname{span}(B) := \{x \in \mathbb{R}^d \mid \exists \alpha_1, \dots, \alpha_k : x = \sum_{i=1}^{k} \alpha_i b_i\}$. We also say that B *spans* this subset. If B contains $\ell \geq 2$ distinct vectors $b_{i_1}, \dots, b_{i_\ell}$ such that there exists scalars $\alpha_1, \dots, \alpha_\ell$, not all zero, that satisfy $\sum_{j=1}^{\ell} \alpha_i b_{i_j} = (0, 0, \dots, 0)^T$ then B is called *linearly dependent*, otherwise B is called *linearly independent*. For a linearly independent B, we say that B is a *basis* of $\operatorname{span}(B)$. For example, if $\operatorname{span}(B) = \mathbb{R}^d$ and B is linearly independent, then we say that B forms a basis of $\mathbb{R}^d$.

A *subspace* of $\mathbb{R}^d$ is a subset $U \subseteq \mathbb{R}^d$ which is the span of a finite set of vectors from $\mathbb{R}^d$, i. e., there exists a finite set B with $U = \operatorname{span}(B)$. Notice that every basis of $\mathbb{R}^d$ contains d vectors, and that for a subspace $U \subseteq \mathbb{R}^d$, all bases have the same cardinality[5]. The number of vectors in a basis of U is called the *dimension* of U. The dimension of $\mathbb{R}^d$ is d.

Abbreviations for the optimal k-means cost. For a given center set $C \subset \mathbb{R}^d$, $|C| = k$, the k-means cost of a (finite) point set $P \in \mathbb{R}^d$ as defined in Definition 2.1.1 can be rewritten as

$$\sum_{x \in P} \operatorname{dist}^2(x, C).$$

We abbreviate this cost by $\operatorname{cost}_{\ell_2^2}(P, C)$, where the ℓ_2^2 indicates that we use the squared 2-norm, which is the Euclidean norm. Additionally, we abbreviate the optimal cost by $\operatorname{cost}^*_{\ell_2^2}(P)$, i. e.,

$$\operatorname{cost}^*_{\ell_2^2}(P) := \min_{C \subset \mathbb{R}^d, |C|=k} \operatorname{cost}_{\ell_2^2}(P, C) = \min_{C \subset \mathbb{R}^d, |C|=k} \sum_{x \in P} \operatorname{dist}^2(x, C).$$

If the point set is weighted by a weight function $w : P \to \mathbb{R}$, we use

$$\operatorname{cost}_{\ell_2^2, w}(P) := \sum_{x \in P} w(x) \operatorname{dist}^2(x, C) \text{ and } \operatorname{cost}^*_{\ell_2^2, w}(P) := \min_{C \subset \mathbb{R}^d, |C|=k} \operatorname{cost}_{\ell_2^2, w}(P, C).$$

Moving points. At this point, we add an insight that is often used in coreset constructions. We discuss it here (even though coresets are not the topic here) because it is easy to prove and adds to our understanding of the k-means problem, and because it will also help in Chapter 3.

The intuitive idea is that if we move the points in an input point set 'not too much', then the cost function will also not change 'much'. It makes sense that the dependency between the distortion of the input and the distortion of the cost function might be quadratic as the cost function includes squared distances. The following lemma makes this more precise and gives a bound on how much we can distort an input if we do not want to increase the cost by more than an ε-fraction. Statements like this lemma are frequently used in coreset theory. A more general version of the following lemma is for example contained in the long version of [FS05].

[5] For vector spaces, these statements are for example proven in Theorem 4 and Corollary 1 on page 44 in [HK72]. The Euclidean space $\mathbb{R}^d$ is a vector space (see Example 1 on page 29 plus the fact that $\mathbb{R}$ is a field, page 2), and so is every subspace of it (for which we would have to show that the above definition implies the precondition of Theorem 1 on page 35 of [HK72]).

Lemma 2.3.2. *Let P, Q be two sets of n points in $\mathbb{R}^d$, let $\varepsilon \in (0,1)$ and and let $\pi : P \to Q$ be a bijection such that $\sum_{x \in P} ||x - \pi(x)||^2 \leq \frac{\varepsilon^2}{16} \cdot \Lambda$ for $\Lambda \geq 0$. Then for any set C of k centers we have $|\text{cost}_{\ell_2^2}(Q,C) - \text{cost}_{\ell_2^2}(P,C)| \leq \varepsilon \cdot \max\{\Lambda, \text{cost}_{\ell_2^2}(P,C)\}$.*

Proof. We use the abbreviation $m_{\Lambda,c} := \max\{\Lambda, \text{cost}_{\ell_2^2}(P,C)\}$. Let C be an arbitrary center set with $|C| = k$. For each $x \in P$ let $a(x) := \text{dist}(x,C) = \min_{c \in C} ||x - c||$ denote the minimal distance from x to any center in C, and let a be the $|P|$-dimensional vector consisting of all $a(x)$ (fix an arbitrary order). Notice that the k-means cost satisfies $\text{cost}_{\ell_2^2}(P,C) = ||a||^2$.

Set $a'(x) = ||\pi(x) - x||$ and let a' be the $|P|$-dimensional vector of all $a'(x)$ (ordered in the same order as a). Notice that $\text{cost}_{\ell_2^2}(Q,C) \leq ||a + a'||^2$ by the triangle inequality. Also by the triangle inequality, $||a + a'|| \leq ||a|| + ||a'|| \leq \sqrt{\text{cost}_{\ell_2^2}(P,C)} + \sqrt{\frac{\varepsilon^2}{16}} \cdot \sqrt{\Lambda}$. The square of this term is

$$\text{cost}_{\ell_2^2}(P,C) + 2\frac{\varepsilon}{\sqrt{16}}\sqrt{\Lambda}\sqrt{\text{cost}_{\ell_2^2}(P,C)} + \frac{\varepsilon^2}{16}\Lambda \leq \text{cost}_{\ell_2^2}(P,C) + \frac{\varepsilon}{2}\Lambda + \frac{\varepsilon^2}{16}\Lambda$$

which implies that

$$\text{cost}_{\ell_2^2}(Q,C) \leq \text{cost}_{\ell_2^2}(P,C) + (\frac{\varepsilon}{2} + \frac{\varepsilon^2}{16})m_{\Lambda,c} < \text{cost}_{\ell_2^2}(P,C) + \varepsilon \cdot \max\{\Lambda, \text{cost}_{\ell_2^2}(P,C)\}.$$

To obtain a bound on $\text{cost}_{\ell_2^2}(P,C) - \text{cost}_{\ell_2^2}(Q,C)$, we distinguish two cases. Either $\text{cost}_{\ell_2^2}(Q,C) \geq \text{cost}_{\ell_2^2}(P,C)$, then the inequality holds because the difference is not positive. Otherwise, $\text{cost}_{\ell_2^2}(P,C) > \text{cost}_{\ell_2^2}(Q,C)$. Then we exchange the roles of P and Q in the above computations to obtain

$$\begin{aligned}\text{cost}_{\ell_2^2}(P,C) &\leq \text{cost}_{\ell_2^2}(Q,C) + \varepsilon \cdot \max\{\Lambda, \text{cost}_{\ell_2^2}(Q,C)\} \\ &\leq \text{cost}_{\ell_2^2}(Q,C) + \varepsilon \cdot \max\{\Lambda, \text{cost}_{\ell_2^2}(P,C)\}\end{aligned}$$

because $\text{cost}_{\ell_2^2}(P,C) > \text{cost}_{\ell_2^2}(Q,C)$. □

Notice that normally, we use a constant Λ with $\Lambda \leq \text{cost}_{\ell_2^2}(P,C)$. In this case, Lemma 2.3.2 directly implies a bound of $\varepsilon \cdot \text{cost}_{\ell_2^2}(P,C)$. On some occasions, we need Λ when the error cannot be bounded with respect to $\text{cost}_{\ell_2^2}(P,C)$ but we want to bound it with respect to a different quantity.

Matrix notation. It can be useful to represent a set of points by a matrix. For a finite set of points $P \subset \mathbb{R}^d$ with $n := |P|$ points, we define $[P] \subset \mathbb{R}^{n \times d}$ as the matrix that contains the points in P as its rows in an arbitrary but fixed order. Then, $[P]_{ij}$ denotes the jth value in row i for $i = 1, \ldots, n$ and $j = 1, \ldots, d$, and $[P]_{i*}$ denotes the complete i-th row and thus the i-th point of P in the fixed ordering.

When we want to refer to a matrix without a direct reference to the represented point set, we usually name the matrix A, its i-th row A_{i*} and the jth entry in row i of A by

A_{ij}. Occasionally, we also use A_{*j} for the jth column of A. When we use A as the input to a k-means problem, we mean that we want to cluster the rows of A. Analogously to the above defined notation we also use $\text{cost}_{\ell_2^2}(A, C) = \sum_{i=1}^{n} \min_{c \in C} ||A_{i*} - c||^2$ for the cost of clustering the points in A with the center set C, and use the abbreviation $\text{cost}^*_{\ell_2^2}(A) = \min_{C,|C|=k} \text{cost}_{\ell_2^2}(A, C)$.

We say that A is *orthogonal* if the set of its column vectors is orthonormal. The *Frobenius norm* of A is denoted by $||A||_F$, i. e.,

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

Matrix multiplication is a generalization of the scalar product for vectors. Let $A \in \mathbb{R}^{n \times d}, B \in \mathbb{R}^{d \times n}$ be matrices. Then the entries of the product $A \cdot B$ are defined by

$$(A \cdot B)_{ij} = \langle A_{i*}, B_{*j} \rangle$$

for all $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, d\}$. For two vectors $x, y \in \mathbb{R}^d$, the scalar product $x^T y$ is just the matrix product of a $(1 \times d)$- matrices with a $(d \times 1)$-matrix.

A matrix always represents a *linear transformation* or *linear map*, i. e., a function $f : \mathbb{R}^d \to \mathbb{R}^{d'}$ that satisfies $f(x + s \cdot y) = f(x) + s(f(y))$ for all $x, y \in \mathbb{R}^d$ and all $s \in \mathbb{R}$. Given a matrix $A \in \mathbb{R}^{d' \times d}$, the corresponding linear transformation is $\phi_A : \mathbb{R}^d \to \mathbb{R}^{d'}$ defined by $\phi_A(x) = Ax$ for all $x \in \mathbb{R}^d$. The matrix-vector multiplication is denoted by Ax and results in the d'-dimensional vector $(\langle A_{1*}, x \rangle, \ldots, \langle A_{d'*}, x \rangle)^T$. Even more, every linear transformation can be written in this form (see, for example, Theorem 11 on page 87 in [HK72]), which means that linear transformations can be identified with the matrices representing them. We thus do not distinguish between A and ϕ_A and for example say that 'A maps a vector' when we mean that ϕ_A maps it.

The set of all vectors that are mapped to the zero vector by A is called the *kernel* of A which we denote by

$$\ker(A) := \{x \in \mathbb{R}^d \mid Ax = 0\}.$$

Another term used for the kernel of a linear map is *null space*. The set of all vectors in $\mathbb{R}^{d'}$ that can be reached by A is the *image* of A which we denote by

$$\text{im}(A) := \{y = Ax \mid x \in \mathbb{R}^d\}.$$

The kernel of A is a subspace of $\mathbb{R}^d$, and the image of A is a subspace of $\mathbb{R}^{d'}$ (see, e. g., [HK72], page 71). Their dimensions adds up to d.

Theorem 2.3.3 (Corollary 4.1.9 on page 262 in [Wat10]). *Let $A \in \mathbb{R}^{n \times d}$. Then,*

$$\dim \ker(A) + \dim \text{im}(A) = \dim \mathbb{R}^d = d.$$

The *rank* of A is the dimension of the image, $\text{rank}(A) := \dim \text{im}(A)$. For a matrix $A \in \mathbb{R}^{n \times d}$, the *transposed* matrix is the matrix $A^T \in \mathbb{R}^{d \times n}$ which is defined by $A^T_{ij} := A_{ji}$. We note the following fact that we will need later on.

Theorem 2.3.4 (see [Wat10], Corollary 5.8.4 on page 390). *For a matrix $A \in \mathbb{R}^{n \times d}$, it holds that* $\text{rank}(A^T A) = \text{rank}(A)$.

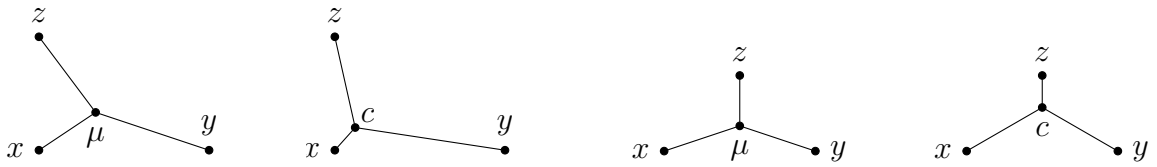


Figure 2.5: Two examples of a point set $\{x, y, z\}$ and its 1-median c and 1-mean μ. In the right example, the points are $x = (-1, 0), y = (1, 0), z = (0, 1)$, the 1-median lies at $c = (0, 1 - \frac{1}{\sqrt{3}})$ and the 1-mean lies at $\mu = (0, \frac{1}{3})$.

2.4 A basic observation and its consequences

For $k = 1$, the k-means problem can be solved analytically. More precisely, the optimal 1-mean of a point set P is its *centroid*, the sum of the points divided by their number. This the topic of this section and we see a proof for it in Lemma 2.4.1. We use the notation $\mu(P) := (\sum_{x \in P} x)/|P|$ for the centroid of a point set P.

That the centroid is the best 1-mean is the first of many consequences of the following basic observation from linear algebra. This lemma is widely used in the development and analysis of algorithms for the k-means problem. For example, it is needed for the work by Zhang, Ramakrishnan and Livny in [ZRL97a] as we discuss in Section 2.4.2. The proof included here is similar to the proof of Lemma 2.1 in [KMN+04].

The lemma allows to split the 1-means cost of a point set P with a given center c into the optimal 1-means cost of the point set plus $|P|$ times the distances between c and the optimal center.

Lemma 2.4.1. *Let $P \subset \mathbb{R}^d$ be a finite point set and let $\mu := \frac{1}{|P|} \sum_{x \in P} x$ be its centroid. Then, every point $z \in \mathbb{R}^d$ satisfies*

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

It P is weighted by $w : P \to \mathbb{R}^+$, then it holds that $\sum_{x \in P} w(x) \cdot \|x - z\|^2 = \sum_{x \in P} w(x) \cdot \|x - \mu_w\|^2 + W \cdot \|z - \mu_w\|^2$ where $W := \sum_{x \in P} w(x)$ and $\mu_w := \frac{1}{W} \sum_{x \in P} w(x)x$.

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

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

Combining both statements implies that

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

Figure 2.6: A third example of a point set $\{x, y, z\}$ and its 1-median and 1-means. When z is moved further to the right, the distance between c and μ increases.

When P is weighted with $w : P \to \mathbb{R}^+$, the calculation is similar:

$$\begin{aligned}
&\sum_{x\in P} w(x)||x-z||^2 \\
=&\sum_{x\in P} w(x)\cdot||x-\mu_w||^2 + 2(\mu_w - z)^t \sum_{x\in P} w(x)\cdot(x-\mu_w) + \sum_{x\in P} w(x)\cdot||z-\mu_w||^2 \\
=&\sum_{x\in P} w(x)\cdot||x-\mu_w||^2 + 2(\mu_w - z)^t\Big(\Big(\sum_{x\in P} w(x)\cdot x\Big) - W\cdot\mu_w\Big) + W\cdot||z-\mu_w||^2 \\
=&\sum_{x\in P} w(x)\cdot||x-\mu_w||^2 + W\cdot||z-\mu_w||^2
\end{aligned}$$

□

We have seen that the proof of Lemma 2.4.1 contains only basic linear transformations. The variety of consequences that this statement has is thus all the more surprising. The first and immediate conclusion that we draw is that the 1-means cost of a point set is always at least $\text{dist}^2(P, \mu(P))$ for its centroid $\mu(P)$, and therefore $\mu(P)$ is the optimal 1-means solution. Additionally, this optimal solution is also unique, because every deviation from μ induces additional cost.

We compare Lemma 2.4.1 with the situation for non-squared distances. There, the *triangle inequality* looks similar: For any two centers $c, c^* \in \mathbb{R}^d$, $||x-c|| \leq ||x-c^*||+||c^*-c||$ holds individually for all points $x \in \mathbb{R}^d$. However, as this only gives an *in*equality, it does not help in finding a closed formula for the 1-median of a point set. In particular, the mean of a point set is *not* an optimal 1-median, see Figure 2.5 for two easy examples where they do not coincide. Looking at Figure 2.6, we see that the centroid and 1-median can differ significantly.

Finding the 1-median or *geometric median* of an arbitrary point set in $\mathbb{R}^d$ is also known as the *Fermat-Weber problem* and has a long history. In fact, it is impossible to construct the 1-median using only a ruler and a compass (*straight-edge and compass constructions*) [Mel73], and the problem is not solvable by *radicals* [Baj88], i. e., it is not expressable in terms of $(+, -, *, /, \sqrt[k]{\cdot})$ over $\mathbb{Q}$. This highlights the importance of the explicit formula for the 1-mean given by Lemma 2.4.1. In the following, we see several other deviations that equip us with useful knowledge and background for advanced studies of the k-means problem.

2.4.1 Enumerating solutions and connections to discrete clustering

Recall that every center set induces a partitioning of the input point set by assigning each point to its closest center and breaking ties arbitrarily. Now Lemma 2.4.1 tells us that for each subset in the partitioning containing points assigned to the same center, the cost is minimized by the centroid of the subset. In particular, in the optimal solution, all centers have to be the centroids of the subset of points assigned to them. Otherwise, the solution could be improved by replacing the center by its centroid. This holds because keeping the assignment of the points to centers while exchanging the centers with the centroids decreases the sum of the squared distances, and then reassigning points to their closest center can only further improve the cost.

Now instead of looking for centers and assigning the points to their closest center, we can also directly look for the optimal partitioning. Given any partitioning, we look at the squared distances of all points to the centroid of their subset in the partitioning, and then we look for the partitioning with lowest cost. The optimal solution to this alternative formulation has the same cost as the optimal solution for the k-means problem, and we can obtain the optimal centers by computing the centroids of the subsets. We formulate this as an observation.

Observation 2.4.2. *Let $P \in \mathbb{R}^d$ be a point set. Finding an optimal solution for the k-means problem on P is equivalent to finding a partitioning of P into $P_1, \dots, P_k$ that minimizes the term*

$$\sum_{i=1}^{k} \sum_{x \in P_i} ||x - \mu(P_i)||^2$$

where $\mu(P_i) := \frac{1}{|P_i|} \sum_{x \in P_i} x$ is the centroid of partition P_i.

Observation 2.4.2 implies that there are at most k^n candidates for the optimal solution of the k-means problem. Notice, however, that this is not polynomial even for constant k. In particular, finding good approximate solutions might be easier when allowing centers that are not centroids of any partitioning.

For the discrete k-means problem, the number of possible solutions is naturally bounded by n^k because there are at most n possibilities to choose each center[6], and this *is* polynomial for constant k. This gives an interesting angle if we can establish a connection between the discrete and continuous case. The following fact states that an optimal solution to the discrete k-means problem is a 2-approximation for the continuous k-means problem, which is another quite immediate implication of Lemma 2.4.1.

Lemma 2.4.3. *Let $P \subset \mathbb{R}^d$ be a finite point set. Let C^* be an optimal center set for the continuous Euclidean k-means problem and let C_d^* be an optimal center set for the discrete Euclidean k-means problem. Then it holds that*

$$\sum_{x \in P} \text{dist}^2(x, C_d^*) \leq 2 \sum_{x \in P} \text{dist}^2(x, C^*).$$

[6]More precisely, there are $\binom{n}{k}$ reasonable possibilities because choosing a center twice cannot improve the solution.

Proof. Let $P_1, \dots, P_k$ be the partitioning of P induced by an optimal center set. Notice that this implies that $\{\mu_i = \sum_{x \in P_i} x \frac{1}{|P_i|} \mid i = 1, \dots, k\}$ is an optimal center set for P. For $i = 1, \dots, k$, let $c_i \in P$ be a point which is closest to μ_i, i. e., for all $x \in P$ it holds that $||c_i - \mu_i|| \leq ||x - \mu_i||$. Now, Lemma 2.4.1 implies that

$$\begin{aligned} \operatorname{dist}^2(P_i, c_i) &= \operatorname{dist}^2(P_i, \mu_i) + |P_i| \cdot ||c_i - \mu_i||^2 = \operatorname{dist}^2(P_i, \mu_i) + \sum_{x \in P_i} ||c_i - \mu_i||^2 \\ &\leq \operatorname{dist}^2(P_i, \mu_i) + \sum_{x \in P_i} ||x - \mu_i||^2 = 2 \operatorname{dist}^2(P_i, \mu_i) \end{aligned}$$

holds for all $i = 1, \dots, k$. The inequality holds because c_i is a point in P which is closest to μ_i. Summing up over all P_i implies the statement.

□

2.4.2 Alternative formulations for k-means

In Observation 2.4.2, we have seen a partition based formulation for the k-means problem. We conclude the section with two more examples on how Lemma 2.4.1 can be used to rewrite the cost function. In particular, we see that the optimal cost can be computed from other statistics of the point set. We start with the following formulation which is the basis for the usage of *clustering* features [ZRL97a].

Observation 2.4.4. *Let $P \subset \mathbb{R}^d$ be a finite point set with mean μ. It holds that*

$$\sum_{x \in P} ||x - \mu||^2 = \sum_{x \in P} ||x||^2 - |P| \cdot ||\mu||^2 \text{ and therefore}$$

$$\sum_{x \in P} ||x - c||^2 = \sum_{x \in P} ||x||^2 + |P|(||\mu - c||^2 - ||\mu||^2) \text{ for any } c \in \mathbb{R}^d.$$

Proof. We use Lemma 2.4.1 to derive the following equation which yields the first statement:

$$\sum_{x \in P} ||x||^2 = \sum_{x \in P} ||x - 0||^2 = \sum_{x \in P} ||x - \mu||^2 + |P| \cdot ||\mu - 0||^2.$$

The second statement directly follows by applying Lemma 2.4.1 again. □

Notice that the centroid can be computed by dividing the sum of the points by their number. With this, we see that the optimal cost and the cost for clustering P with one fixed center c can be computed by only using the number of the points, the sum of the points and the sum of the squared lengths of all points. These statistics are named clustering features in [ZRL97a]. The advantage is that they can be stored on the fly. When reading an input consisting of points, we can update their number, and the two sums after each point and thus output the centroid *and* the clustering cost for any desired center at any point during the pass over the data. We will further investigate this in Chapter 5 in the context of data stream algorithms for k-means clustering.

A second fact which will turn out to be useful is that the optimal cost can also be rewritten such that it only depends on pairwise distances of points from the data set. The benefits of this formulation are not immediately clear, but we will further discuss them in Section 3.1.1.

Lemma 2.4.5. *Let $P \subset \mathbb{R}^d$ be a finite point set with mean μ. It holds that*

$$\sum_{x \in P} ||x - \mu||^2 = \frac{1}{2|P|} \sum_{x \in P} \sum_{y \in P} ||x - y||^2.$$

Proof. We interpret $\sum_{x \in P} ||x - y||^2$ as the clustering of P with center y and can then apply Lemma 2.4.1. This yields that

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

□

3 Dimensionality reduction techniques

An input instance for the k-means problem can be large due to two factors: The number of input points, and the dimension of the input points. Dimensionality reduction targets the second issue. The idea is to map the point set to a lower dimensional point set such that a solution for the lower dimensional problem can be translated to a solution for the original problem with approximately the same cost. Dimensionality reduction can then be used to speed up algorithms that have a dimension-depending running time.

After an introduction to this area which includes two important dimensionality reduction methods, we see a new dimensionality reduction result in Section 3.2. This result and the results in Section 4.3 and Section 4.4 are joint work with Dan Feldman and Christian Sohler and have been published in [FSS13].

3.1 Two popular dimensionality reduction techniques

There are different approaches to perform a dimensionality reduction while ensuring that the solution in the lower dimensional space is somehow transferable to the space of the original points. One way is to make sure that the clustering cost of every subset of the input point set is preserved with every possible center. Then, the solution can be transferred to the original space by transferring the partitioning and getting the centers by computing the centroids of the subsets in the partitioning. A different way is to project to a *subspace* of the original space and show that the optimal centers of the projected point set form a good solution for the original point set, too. In this section, we will see examples for both techniques, resulting in two different ways to compute approximate k-means clusterings via dimensionality reductions. We start with reviewing some necessary facts about projections from linear algebra.

Projections. The easiest case of a projection is when we project along the axes of the Euclidean standard basis. If we want to project a vector $x = (x_1, \ldots, x_d)^T \in \mathbb{R}^d$ to the space spanned by the i-th standard basis vector e_i, then the length of the projection is just $|x_i|$, the ith coordinate of x, and the projection is the vector $x_i e_i$. Dimensionality reduction that selects dimensions and reduces the points to these dimensions are also called *feature selection* methods because the dimensions usually correspond to features (attributes that were measured when collecting the data). *Feature extraction* methods reduce the dimension by creating new features, for example by projecting the point set to a lower dimensional subspace of the input space. Notice that in particular feature selection is not only used to reduce the size, but also to get rid of unimportant features in order to increase the

classification quality. For a broader discussion see [JDM00, JMF99]. In the following, we mostly consider feature extraction.

An easy case of feature extraction is to (orthogonally) project onto the (one-dimensional) subspace spanned by an arbitrary unit vector $u \in \mathbb{R}^d$. We use the same principle as for the standard basis vectors, but we need an appropriate basis to understand the projection better. So, we extend u to an orthonormal basis[1] by choosing appropriate vectors $b_2, \ldots, b_d$. Then any vector $x \in \mathbb{R}^d$ has a unique representation in this basis, given by the unique choice of coefficients $\alpha, \beta_2, \ldots, \beta_d$ that satisfy $x = \alpha \cdot u + \sum_{i=2} \beta_i b_i$ (see Lemma A.5.4 in the Appendix). Now we define the projection of x onto the direction of u as $\alpha \cdot u$. Notice the similarity to the case of the standard basis, because the length of this projection is just the (absolut value of the) coefficient of u in the representation of the basis $u, b_2, \ldots, b_d$, and the projection is this coefficient multiplied by the basis vector u.

Projections and the scalar product. In this context, the following property of the scalar product is particularly useful to keep in mind. Given a unit vector $u \in \mathbb{R}^d$, it holds for every $x \in \mathbb{R}^d$ that $|u^T x|$ is just the length of the projection onto the subspace spanned by u. This is true because for the unique representation $x = \alpha \cdot u + \sum_{i=2} \beta_i b_i$ with $\alpha, \beta_2, \ldots, \beta_d \in \mathbb{R}^d$ and the orthonormal basis $u, b_2, \ldots, b_d$ we get that $u^T x = u^T(\alpha \cdot u + \sum_{i=2} \beta_i b_i) = \alpha u^T u + \sum_{i=2}^d \beta_i u^T b_i = \alpha \cdot 1 + \sum_{i=2}^d \beta_i \cdot 0 = \alpha$.

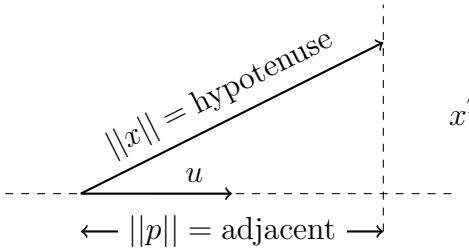
Observation 3.1.1. *Let $x \in \mathbb{R}^d$ be an arbitrary vector from the d-dimensional Euclidean space and let $u \in \mathbb{R}^d$ be a unit vector. Then, $|u^T x|$ is the length of the (orthogonal) projection of x onto the one-dimensional subspace spanned by u, and the projected vector is $(u^T x)u$.*

When we recall from trigonometry that the scalar product also satisfies $x^T y = \cos(\sphericalangle xy) \cdot ||x|| \cdot ||y||$, we get an alternative way to derive this fact[2]. The scalar product of a vector with a unit vector $u \in \mathbb{R}^d$ satisfies $x^T u = (\sphericalangle xu) \cdot ||x||$. Now consider the case that $x^T u$ is positive and recall the definition of cosine in right-angled triangles. Then Figure 3.1 visualizes why the scalar product of x and u just gives the length of the projection of x to the direction of u.

Projection matrices. We can also define a map $\pi_{\text{span}(u)}(x) = (u^T x)u$ that maps every vector in $\mathbb{R}^d$ to its projection in $\text{span}(u)$. This projection (and, in fact, every projection) is a linear transformation because for all $x \in \mathbb{R}^d$ and all $s \in \mathbb{R}$, we have that $\pi_{\text{span}(u)}(s \cdot x + y) = (u^T(s \cdot x + y))u = s \cdot \pi_{\text{span}(u)}(x) + \pi_{\text{span}(u)}(y)$ by the definition of the scalar product. As a linear transformation, it can be represented by a matrix in the form $\pi_{\text{span}(u)}(x) = Ax$. Indeed, we see that $\pi_{\text{span}(u)}(x) = (u^T x)u = u(u^T x) = uu^T x$ because multiplication with a scalar is commutative and because matrix multiplication is associative. Now, by the

[1] This is possible due to the orthogonalization process by Gram [rPG83] and Schmidt [Sch07], also see Corollary A.5.3 in the appendix.

[2] Here, $\cos(\sphericalangle xy)$ denotes the cosine of the angle between x and y.



$$x^T u = \cos(\sphericalangle xu) \cdot ||x|| = \tfrac{\text{adjacent}}{\text{hypotenuse}} \cdot ||x|| = ||p||$$

Figure 3.1: A visualization of Observation 3.1.1.

definition of the matrix product,

$$uu^T = \begin{pmatrix} u_1u_1 & \cdots & u_1u_d \\ \cdots & \ddots & \cdots \\ u_du_1 & \cdots & u_du_d \end{pmatrix}$$

is a $d \times d$-matrix, the so-called *tensor product* of u with itself. We call a linear transformation that maps a vector to a subspace *projection* (even though this creates a homonym), and we call the corresponding matrix *projection matrix*.

Projecting to subspaces of higher dimensions. To define the (orthogonal) projection onto a d'-dimensional subspace U, we choose an orthonormal basis $a_1, \ldots, a_{d'}$ of U and extend it with $d - d'$ vectors $b_{d'+1}, \ldots, b_d$ to an orthonormal basis of $\mathbb{R}^d$. For any $x \in \mathbb{R}^d$ with the unique representation $x = \sum_{i=1}^{d'} \alpha_i \cdot a_i + \sum_{i=d'+1}^{d} \beta_i b_i$ for $\alpha_1, \ldots, \alpha_{d'}, \beta_{d'+1}, \ldots, \beta_d \in \mathbb{R}$, we define the projection of x to U as $\sum_{i=1}^{d'} \alpha_i \cdot a_i$.

As we have already seen above, α_i is just $(a_i)^T x$ for all $i \in [d']$ because the a_i are unit vectors. Thus, we can rewrite the projection of x to U by

$$\sum_{i=1}^{d'} \alpha_i \cdot a_i = \sum_{i=1}^{d'} (a_i^T x) a_i = \sum_{i=1}^{d'} a_i a_i^T x.$$

Thus, the projection matrix of the linear transformation $\pi_U : \mathbb{R}^d \to U$ mapping every $x \in \mathbb{R}^d$ to its projection in U is just $\sum_{i=1}^{d'} a_i a_i^T$, the sum of the projection matrices for mapping to each of the one-dimensional subspaces spanned by the basis vectors of U. We usually denote the projection to a subspace U by π_U.

Distances and subspaces. We explicitly state two facts about distances to subspaces and to points within subspaces that are based on projections. First, using the notation from the last paragraph, notice that every point $y \in U$ is of the form $y = \sum_{i=1}^{d'} \alpha_i' a_i$ for uniquely defined $\alpha_i' \in \mathbb{R}$. This implies that the squared distance between a point

$x = \sum_{i=1}^{d'} \alpha_i \cdot a_i + \sum_{i=d'+1}^{d} \beta_i b_i \in \mathbb{R}^d$ and y is just

$$\begin{aligned}\text{dist}^2(x,y) = \left|\left|\sum_{i=1}^{d'}(\alpha_i - \alpha_i')a_i + \sum_{i=d'+1}^{d}\beta_i b_i\right|\right|^2 &= \sum_{i=1}^{d'}(\alpha_i - \alpha_i')||a_i||^2 + \sum_{i=d'+1}^{d}\beta_i||b_i||^2 \\ &= \sum_{i=1}^{d'}(\alpha_i - \alpha_i')^2 + \sum_{i=d'+1}^{d}\beta_i^2 \qquad (3.1)\end{aligned}$$

where the second equality follows by the Pythagorean theorem 2.3.1 and the fact that $\{a_1, \ldots, a_{d'}, b_{d'+1}, \ldots, b_d\}$ is an orthonormal basis. As the second term is independent of y, $\text{dist}^2(x,y)$ is minimized when $\alpha_i = \alpha_i'$ for all $i = 1, \ldots, d'$, i. e., when $y = \pi_U(x)$.

Observation 3.1.2. *Let $U \subseteq \mathbb{R}^d$ be a subspace of $\mathbb{R}^d$ and let $x \in \mathbb{R}^d$ be a point. It holds that*

$$\text{dist}^2(x,U) = \min_{y \in U} \text{dist}^2(x,y) = \text{dist}^2(x, \pi_U(x))$$

where π_U is the projection to U.

Another look at Equation 3.1 also tells us that we can split the squared distance between x and y in two parts. The first term, $\sum_{i=1}^{d'}(\alpha_i - \alpha_i')^2$, is the distance between y and $\pi_U(x)$, while the second term, $\sum_{i=d'+1}^{d}\beta_i^2$, is the distance between x and $\pi_U(x)$. This in particular holds for $y = 0$, when $\text{dist}^2(x,y)$ collapses to $||x||^2$ and $\text{dist}^2(\pi_U(x), y)$ becomes $||\pi_U(x)||^2$.

Observation 3.1.3. *Let $U \subseteq \mathbb{R}^d$ be a subspace of $\mathbb{R}^d$, let $x \in \mathbb{R}^d$ be a point and let $y \in U$. It holds that*

$$||x||^2 = \text{dist}^2(x, \pi_U(x)) + ||\pi_U(x)||^2 \text{ and } \text{dist}^2(x,y) = \text{dist}^2(x, \pi_U(x)) + \text{dist}^2(\pi_U(x), y)$$

where π_U is the projection to U.

Maps to a lower dimensional space. Instead of projecting the points to a subspace of lower dimension, we can also directly define the linear transformation to a lower dimensional space $\mathbb{R}^{d'}$. We just use the basis $\{a_1, \ldots, a_{d'}\}$ of the subspace and map every x to the d'-dimensional vector of the coordinates in this basis. This linear transformation is given by the $(d' \times d)$-matrix which contains the a_i in its rows, because then the ith entry of the resulting vector when multiplying the matrix with x is just $(a_i)^T x$.

3.1.1 Random projections

We start with a dimensionality reduction method that maps the input points to a different space of lower dimension. This means that the projection maps to points with less coordinates which do thus not lie within the original Euclidean space. Our focus is on preserving the k-means objective function, i. e., on providing a point set of smaller dimension which has a similar k-means cost as the original point set, preferably for every choice of k centers. We have seen that a linear transformation to a space of dimension d' is given by a $(d' \times d)$-matrix.

The easiest idea that might come to mind is to choose a random transformation by picking a random matrix. That this approach does actually work (if we pick according to a suitable probability distribution) follows from a seminal result by Johnson and Lindenstrauss from the eighties [JL84], the so-called *Johnson-Lindenstrauss Lemma*, sometimes also referred to as Johnson-Lindenstrauss *flattening* Lemma. We now state the Johnson-Lindenstrauss Lemma and discuss its importance. That it includes a random choice of a linear transformation will become clear in the paragraph after that. The following formulation of the Johnson-Lindenstrauss Lemma is inspired by Dasgupta and Gupta [DG03].

Theorem 3.1.4 (Johnson-Lindenstrauss Lemma [JL84])**.** *Given a constant $\varepsilon \in (0,1)$ and integers n and d' with $d' \geq d_0 \in \mathcal{O}(\varepsilon^{-2} \log n)$, there exists a linear map $f : \mathbb{R}^d \to \mathbb{R}^{d'}$ such that for all $x, y \in P$, it holds that*

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

Such a map can be found in randomized polynomial time with constant success probability.

Theorem 3.1.4 states that every point set P with $|P| = n$ can be reduced to $\mathcal{O}(\varepsilon^{-2} \log n)$ dimensions, disregarding its original dimension, while approximately keeping the pairwise distances. Distance-preserving projections to a different space are called *embeddings*. If an embedding has no error, it is called *isometric* embedding. The Johnson-Lindenstrauss Lemma gives an *approximate* embedding.

Embeddings can be seen as a form of compression. As Matoušek [Mat02] says: If we have n points in $\mathbb{R}^n$, storing them means storing $\Theta(n^2)$ numbers. It also means storing $\Theta(n^2)$ numbers if we decide to simply store their pairwise distances. However, if we fix ε to a constant and use Theorem 3.1.4, we can project the points to $\mathcal{O}(\log n)$ dimensions, meaning that we only need to store $\mathcal{O}(n \log n)$ numbers. Yet, all pairwise distances can be approximately recovered from the compressed point set.

There is a high interest in embeddings, and we are only discussing the Johnson-Lindenstrauss Lemma and its implications for the k-means problem here. For an overview on the more general topic of *metric* embeddings, see for example the book chapters by Matoušek [Mat05] or Indyk and Matoušek [IM04] on embedding finite metric spaces. For a survey on implications of embeddings for algorithms, see Indyk [Ind01].

A finite metric space is given by a finite number of points and a choice of pairwise distances that satisfies the properties of a metric. Thus, n points in the Euclidean space and their pairwise distances form a metric space, and the Johnson-Lindenstrauss Lemma is an approximate metric embedding. As such, it is also discussed in the two above references. Actually, the Johnson-Lindenstrauss Lemma holds a special position among metric embeddings because it gives a surprising and strong result. For example, consider the case of l_1 norms, i. e., the case that we define the distances for two points $x, y \in \mathbb{R}^d$ by $\sum_{i=1}^{d} |x_i - y_i|$. Then an approximate embedding like in Theorem 3.1.4 with parameter ε requires that the target space has at least $n^{\Omega(\varepsilon^{-2})}$ dimensions [BC05, LN04]. Thus, basically no analogue to the Johnson-Lindenstrauss Lemma can exist for l_1 norms.

Application to the k-means problem. The approximation guarantee in Theorem 3.1.4 is stated for non-squared distances. Some of the proofs of the Johnson-Lindenstrauss Lemma directly prove the statement for squared distances. However, notice that this is not required if we are not interested in the actual constants. If $||f(x) - f(y)|| \leq (1+\varepsilon)||x-y||$, then $||f(x) - f(y)||^2 \leq (1+\varepsilon)^2||x-y||^2 = (1+2\varepsilon+\varepsilon^2)||x-y||^2 \leq (1+3\varepsilon)||x-y||^2$, and a similar chain works for the lower bound on $||f(x) - f(y)||^2$. So, applying Theorem 3.1.4 with a smaller epsilon gives the same approximation guarantee for squared distances.

Now, reconsider Lemma 2.4.5 from Section 2.4.2 which says that the sum of the squared distances of points to their centroid can be written by solely using the pairwise distances of the points. This does not only apply to the whole input set P, but also to every subset of P. Thus, preserving all pairwise distances also preserves the 1-means clustering cost of every subset of P. Recall that Observation 2.4.2 says that we can find an optimal solution by finding an optimal partitioning. Therefore we can approximate the k-means cost of a point set by projecting it to $\mathcal{O}(\varepsilon^{-2}\log n)$ dimensions and then computing the k-means cost of the projected point set.

Lemma 3.1.5. *Let $P \in \mathbb{R}^d$ be a point set with n points and let $\varepsilon \in (0,1)$ and $d' \geq d_0 \in \mathcal{O}(\varepsilon^{-2}\log n)$ be constants. There exists a linear map $f : \mathbb{R}^d \to \mathbb{R}^{d'}$ such that*

$$(1-\varepsilon)\cdot \text{cost}^*_{\ell_2^2}(P) \leq \text{cost}^*_{\ell_2^2}(f(P)) \leq (1+\varepsilon)\cdot \text{cost}^*_{\ell_2^2}(P)$$

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

Proof. We apply Theorem 3.1.4 with precision parameter $\varepsilon/3$, gaining that there exists a map $f : \mathbb{R}^d \to \mathbb{R}^{\mathcal{O}(\varepsilon^{-2}\log n)}$ such that $(1-\varepsilon)||x-y||^2 < ||f(x)-f(y)|| < (1+\varepsilon)||x-y||^2$ holds for all $x, y \in P$. By Observation 2.4.2, optimizing the k-means cost function is equivalent to finding a partitioning of P into k subsets $P_1, \ldots, P_k$ minimizing

$$\sum_{i=1}^{k}\sum_{x\in P_i}||x-\mu(P_i)||^2 = \sum_{i=1}^{k}\frac{1}{2|P_i|}\sum_{x,y\in P_i}||x-y||^2 \tag{3.2}$$

where the equality is due to Lemma 2.4.5. Let $P_1^*, \ldots, P_k^*$ be an optimal partitioning of P. This partitioning can be applied to $f(P)$ (just partition the projections accordingly), and we can use it to bound the k-means cost of $f(P)$ by using formulation 3.2 again. This implies the second inequality of the observation because

$$\begin{aligned}\text{cost}^*_{\ell_2^2}(f(P)) &\leq \sum_{i=1}^{k}\frac{1}{2|P_i|}\sum_{x,y\in P_i}||f(x)-f(y)||^2 \\ &< \sum_{i=1}^{k}\frac{1}{2|P_i|}\sum_{x,y\in P_i}(1+\varepsilon)||x-y||^2 = (1+\varepsilon)\cdot\text{cost}^*_{\ell_2^2}(P).\end{aligned}$$

Assume that $\text{cost}^*_{\ell_2^2}(f(P)) < (1-\varepsilon) \cdot \text{cost}^*_{\ell_2^2}(P)$. Let $P'_1, \ldots, P'_k$ be a partitioning of P that is optimal for $f(P)$, which implies that

$$\begin{aligned}(1-\varepsilon) \cdot \text{cost}^*_{\ell_2^2}(P) > \text{cost}^*_{\ell_2^2}(f(P)) &= \sum_{i=1}^{k} \frac{1}{2|P'_i|} \sum_{x,y \in P'_i} ||f(x) - f(y)||^2 \\ &> \sum_{i=1}^{k} \frac{1}{2|P'_i|} (1-\varepsilon) \cdot \sum_{x,y \in P'_i} ||x-y||^2 \geq (1-\varepsilon) \cdot \text{cost}^*_{\ell_2^2}(P)\end{aligned}$$

which is a contradiction. So, $(1-\varepsilon) \cdot \text{cost}^*_{\ell_2^2}(f(P) \leq \text{cost}^*_{\ell_2^2}(P)$. □

Proof idea for the Johnson-Lindenstrauss Lemma. There exist different proofs for Theorem 3.1.4 which yield different values of d_0 and matrices with different computational properties. In the original proof by Johnson and Lindenstrauss [JL84], d_0 was only specified asymptotically. A later proof by Frankl and Maehara [FM88], which is generally said to be much simpler, achieves a value of $d_0 = \frac{9}{\varepsilon^{-2} - 2\varepsilon^3/3} \log n + 1$ for the dimension of the target space. The next alternative proof can be found in (the appendix of) the paper by Indyk and Motwani [IM98] in 1998. A notably concise proof using elementary probability theory is due to Dasgupta and Gupta [DG03]. The latter achieves a dimension of $d_0 = \frac{4}{\varepsilon^2 - \varepsilon^3/3} \ln n$. Additional alternative and short proofs are contained in the papers by Matoušek and [Mat08] and by Indyk and Naor [IN07].

The core of the proof is showing that for a suitable probability distribution, a randomly chosen linear transformation does not distort the length of an arbitrary but fixed vector $x \in \mathbb{R}^d$ too much. More precisely, a lemma like the following statement holds. A similar statement can be found in the presentation of the Johnson-Lindenstrauss Lemma by Matoušek in [Mat08].

Lemma 3.1.6. *Given a constant $\varepsilon \in (0,1)$ and integers n and d' with $d' \geq d_0 \in \mathcal{O}(\varepsilon^{-2} \log n)$, there exists a probability distribution $\mathcal{D}$ over the space of all $d' \times d$-matrices such that a random matrix A^* drawn according to $\mathcal{D}$ satisfies for every fixed vector $x \in \mathbb{R}^d$ that*

$$\text{Prob}\left((1-\varepsilon)||x|| \leq ||A^*x|| \leq (1+\varepsilon)||x||\right) \geq 1 - \frac{1}{n^2}.$$

The randomness in this statement is (only) over the random choice of A^*. The probability $1 - 1/n^2$ means that we can apply Lemma 3.1.6 to n^2/κ vectors simultaneously and achieve that the approximation bounds hold for all vectors simultaneously with the constant probability $1 - 1/\kappa$ by the union bound (for a constant number κ). Given a set of n input points P, the idea is to look at the $\binom{n}{2} = \frac{n(n-1)}{2} < n^2/2$ distance vectors between pairs of input points. A random linear transformation chosen according to $\mathcal{D}$ will then satisfy $(1-\varepsilon)||x-y|| \leq ||A^*(x-y)|| \leq (1+\varepsilon)||x-y||$ for all $x, y \in P$ at the same time with a probability of at least $1/2$. As $A^*(x-y) = A^*x - A^*y$, this implies the Johnson-Lindenstrauss Lemma.

Lemma 3.1.6 is an illustration of the type of result that is needed to obtain Theorem 3.1.4, but different proofs of the Johnson-Lindenstrauss Lemma use different (but similar) statements. In particular, the proof in [DG03] that we now shortly review, contains a different concentration lemma. In fact, Dasgupta and Gupta also proof a slightly stronger version of the Johnson-Lindenstrauss Lemma where the distances are squared.

As in earlier proofs, Dasgupta and Gupta chose A^* as a projection to a random k-dimensional subspace, appropriately scaled. Notice that the proof can be restricted to unit vectors because A^*x is a linear map. So the proof requires to show that the length of the scaled projection of a given unit vector to a random d_0-dimensional subspace is close to one with the given probability. This is essentially the same as showing that the scaled projection of a random unit vector on its first d_0 coordinates satisfies this property. Imagine that we want to project a vector to a random subspace. Then we draw a random unit vector, rotate our unit vector into this vector, project it to its first d_0 coordinates and reverse the rotation.

A random unit vector is drawn as follows. Define d independent random variables $X_1, \ldots, X_d$ which are $\mathcal{N}(0,1)$ distributed and write $X = (X_1, \ldots, X_d)^T$. Then the random unit vector is just $X/||X||$, and the result of the projection is $Y = (X_1, \ldots, X_{d'})/||X||$. Now the core of the proof in [DG03] is to show that the squared length of Y deviates from k/d with sufficiently small probability. This suffices to show a sufficient concentration bound similar to Lemma 3.1.6 and to achieve a Johnson-Lindenstrauss Lemma with squared distances for projections to $d_0 = \frac{4}{\varepsilon^2 - \varepsilon^3/3} \ln n$ dimensions.

Computationally preferable versions of the Johnson-Lindenstrauss Lemma. As the proof idea above suggested, the linear map A^* will consist of real values, chosen from a scaled Gaussian distribution. This means that constructing A^* and calculating the matrix-vector product A^*x is computationally expensive.

Achlioptas [Ach03] suggested two computationally preferable versions of the Johnson-Lindenstrauss Lemma. The first version uses only $\{-1, +1\}$-based random variables. Every entry is chosen uniformly at random from $\{-1, +1\}$, then A^* is appropriately scaled. In his second version, each entry of A^* is either $+1$ with probability $1/6$, -1 with probability $1/6$ and 0 with probability $2/3$, then A^* is again appropriately scaled. His work shows the two main ideas that help to find embeddings with better computational properties. First, the usage of randomness is expensive, and drawing Gaussian distributed numbers is more expensive than drawing from a finite set of constants. Second, sparse matrices reduce the computation cost because the zeros do not contribute to the matrix-vector product.

A lot of research was subsequently devoted to the field of computationally efficient Johnson-Lindenstrauss embeddings. Ailon and Chazelle [AC09] overcame the challenge that significantly sparse random projections do no longer satisfy sufficient concentration of the length of projected vectors around their expected value. They showed that the concentration holds if the coordinates of the input vector are in some sense 'well spread' meaning that the maximum entry of the vector x is around $1/\sqrt{d}$. They then defined A^* as the product of different matrices, first spreading the coordinates of an input vector

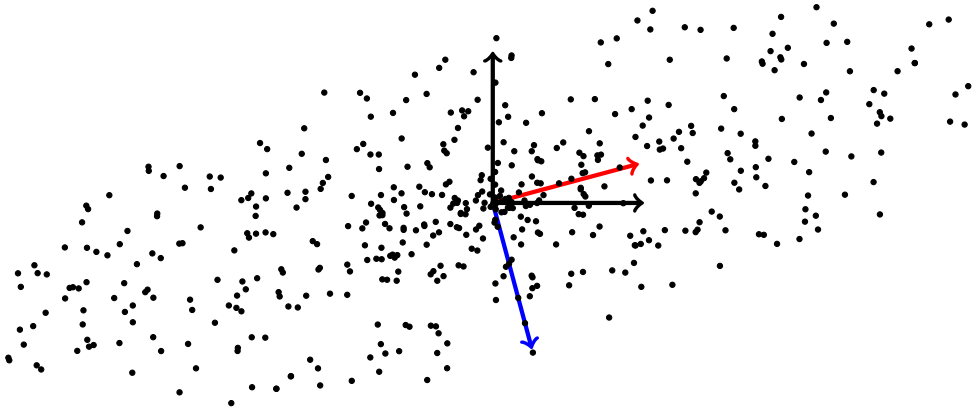


Figure 3.2: An example of a point set and its principal components. The black arrows indicate the standard basis, the red and blue arrow indicate a basis based on the principal components.

with high probability before applying the actual projection. The sparsest known Johnson-Lindenstrauss embedding is due to Kane and Nelson [KN12]. Their construction yields embeddings where the matrices contains at most $s = \mathcal{O}(\varepsilon^{-1} \log(n))$ nonzero entries in each column, while still embedding into only $\mathcal{O}(\varepsilon^{-2} \log(n))$ dimensions.

Several other variants and improvements were proposed between the work Achlioptas and the result by Kane and Nelson, and the field is still developing. For pointers to several papers from this field, including derandomization results, see for example the introduction of [KN10].

Lower bound. The Johnson-Lindenstrauss Lemma is tight (up to constant factors). Alon [Alo03] shows (in Section 9, Remarks) that in order to embed $n + 1$ points with pairwise distance 1 into $\mathbb{R}^{d'}$ such that the pairwise distances of the projected points lie within $(1, 1 + \varepsilon)$, it must hold that $d' \in \Omega(\frac{1}{\varepsilon^2 \log 1/\varepsilon} \log n)$. By scaling, this lower bound also applies to the above version of the Johnson-Lindenstrauss Lemma. The $(\log 1/\varepsilon)$ gap was closed for random linear maps by Jayram and Woodruff [JW13] and Kane, Meka and Nelson [KMN11], and for all linear maps by Larsen and Nelson [LN14].

The tight lower bound in particular means that the logarithmic dependence on the number of points is necessary. Notice however that this does *not* mean that embeddings into smaller dimensions cannot preserve the k-means cost function. While Lemma 2.4.5 gives a convenient way to use embeddings, it is not necessarily the only way. What the lower bound does tell us is that any dimensionality reduction to $o(\log n)$ dimensions that wants to preserve the k-means cost function by an arbitrarily small factor cannot preserve the pairwise distances.

3.1.2 The best fit subspace and the singular value decomposition

Now we turn to a equally popular tool for dimensionality reduction. Consider Figure 3.2. It shows a point set in $\mathbb{R}^2$ and the standard two-dimensional Euclidean basis. The mean of the point set lies in the origin. Assume we want to reduce the point set to one dimension, but we want to keep as much of the lengths of the vectors as possible. Then projecting them to the direction of the red vector is the best thing to do. It points into the direction where the point set has the largest variance, also called *principal component*. Alternatively, we can also say that it spans the one dimensional *best fit subspace*, which is the subspace minimizing the (squared) distance of the points to the subspace.

In particular in higher dimensions, more than just the first principal component are of interest. To find the next principal component, we look at the subspace orthogonal to the first component and again pick the direction where the points have the highest variance. This is iteratively done until all principal components are found. In Figure 3.2, the second direction is basically determined because there is only one subspace orthogonal to the red vector.

We now review the mathematical background of this idea. As it turns out, given a point set P, the principal components are the eigenvectors of the matrix $[P]^T[P]$. Most of the time, we look at them from a different angle, based on the *singular value decomposition* (SVD). The SVD allows us to rewrite a matrix as a product of matrices which are composed of *singular* vectors and values of $[P]$. We will discuss the definition and interpretation of these terms below. The singular value decomposition is older than the Johnson-Lindenstrauss Lemma and was developed by different mathematicians around the beginning of the 19th century. For an historical overview on the origins of the decomposition, see the survey by Stewart [Ste93].

Definition of the SVD. The singular value decomposition (SVD) of a matrix $A \in \mathbb{R}^{n\times d}$ allows us to write A as $A = UDV^T$ where $D \in \mathbb{R}^{n\times d}$ is a diagonal matrix and $U \in \mathbb{R}^{n\times n}$, $V \in \mathbb{R}^{d\times d}$ are orthogonal matrices, i. e.,

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

where $u_1, \ldots, u_n \in \mathbb{R}^n$ and $v_1, \ldots, v_d \in \mathbb{R}^d$ are orthogonal unit vectors and $\sigma_1 \geq \sigma_2 \geq \ldots \sigma_{\min\{n,d\}} \geq 0$ are nonincreasing and nonnegative values. Such a decomposition always

exists (see, e. g., [Wat10], page 260, Theorem 4.1.1). Notice that

$$
\begin{aligned}
UDV^T &= \left(\begin{array}{cccc|c} \sigma_1 u_{11} & \sigma_2 u_{21} & \cdots & \sigma_{\min\{n,d\}} u_{\min\{n,d\}1} & \\ \sigma_1 u_{12} & \sigma_2 u_{22} & & & \\ \vdots & \vdots & \ddots & \vdots & 0 \\ \sigma_1 u_{1n} & \sigma_2 u_{2n} & \cdots & \sigma_{\min\{n,d\}} u_{\min\{n,d\}n} & \end{array}\right) \cdot \begin{pmatrix} — & v_1 & — \\ & \vdots & \\ — & v_d & — \end{pmatrix} \\
&= \begin{pmatrix} \sum_{i=1}^{\min\{n,d\}} \sigma_i u_{i1} v_{i1} & \cdots & \sum_{i=1}^{\min\{n,d\}} \sigma_i u_{i1} v_{id} \\ \vdots & \ddots & \vdots \\ \sum_{i=1}^{\min\{n,d\}} \sigma_i u_{in} v_{i1} & \cdots & \sum_{i=1}^{\min\{n,d\}} \sigma_i u_{in} v_{id} \end{pmatrix} \\
&= \begin{pmatrix} — & \sum_{i=1}^{\min\{n,d\}} \sigma_i u_{i1} v_i & — \\ & \vdots & \\ — & \sum_{i=1}^{\min\{n,d\}} \sigma_i u_{in} v_i & — \end{pmatrix} \qquad (3.3) \\
&= \sum_{i=1}^{\min\{n,d\}} \sigma_i u_i v_i^T \qquad (3.4)
\end{aligned}
$$

where u_{ij} and v_{ij} are the jth coordinate of u and v, respectively, and $u_i v_i^T$ is the tensor product of u_i and v_i. When we restrict the sum in Equation 3.4 to the first $k \leq \min\{n, d\}$ terms, we get the matrix $A^{(k)} := \sum_{i=1}^{k} \sigma_i u_i v_i^T =: UD^{(k)}V^T$ where $D^{(k)}$ is the matrix resulting from replacing $\sigma_{k+1}, \ldots, \sigma_{\min\{n,d\}}$ by zeros. We discuss $A^{(k)}$ in more detail on page 45.

This decomposition is called *singular value* decomposition because it contains the values $\sigma_1, \ldots, \sigma_{\min\{n,d\}}$ which are the singular values of A, and because it contains singular vectors, the u_i and v_i. In the following, we review the definitions of eigenvalues, eigenvectors, singular values and singular vectors and see why they can be used as building blocks of this decomposition.

Eigenvalues and eigenvectors. For a square matrix $A \in \mathbb{R}^{d \times d}$, a real value $\lambda \in \mathbb{R}$ is an *eigenvalue* if there exists a nonzero vector $v \in \mathbb{R}^d$ called *eigenvector* such that

$$Av = \lambda v. \qquad (3.5)$$

Notice that while the eigenvectors are nonzero by definition, the eigenvalues *can* be zero. Then, the corresponding eigenvectors are mapped to the zero vector and thus belong to the kernel of A. The other way around, every nonzero vector in the kernel of A is an eigenvector corresponding to the eigenvalue zero, and an orthonormal basis for the kernel forms a set of orthonormal eigenvectors belonging to the eigenvalue zero. By Theorem 2.3.3 on page 24, this set contains $r := d - \operatorname{rank}(A)$ vectors $v_1, \ldots, v_r$. If we can extend this set with $\operatorname{rank}(A)$ eigenvectors which belong to nonzero eigenvalues, are orthogonal to each other, and are orthogonal to $v_1, \ldots, v_r$, then we have a basis for $\mathbb{R}^n$ consisting solely of eigenvectors. Such a basis is called an *eigenbasis*. It always exists in the case of symmetric matrices.

Theorem 3.1.7 (Theorem 8.1.1 on page 393 in [GL96], Lemma 1.21 on page 21 in [MS06])**.** *Let $A \in \mathbb{R}^{d\times d}$ be symmetric. Then there exist d distinct eigenvectors of A that are pairwise orthogonal with d corresponding eigenvalues, and the number of eigenvectors belonging to nonzero eigenvalues is equal to the rank of A.*

Singular values and vectors. If $A \in \mathbb{R}^{n\times d}$ is not square, singular values and a pair of singular vectors replace eigenvalues and eigenvectors, because Equation (3.5) is not well-defined in this case. A real value $\sigma \in \mathbb{R}$ is a *singular value* if there exist a *left singular vector* $u \in \mathbb{R}^n$ and a *right singular vector* $v \in \mathbb{R}^d$ such that

$$Av = \sigma u \tag{3.6}$$
$$u^T A = \sigma v^T. \tag{3.7}$$

Consider the matrix $A^T A \in \mathbb{R}^{d\times d}$ and notice that this matrix is symmetric. Thus, there exist d pairwise orthogonal eigenvectors of $A^T A$ with corresponding eigenvalues. By the following fact, the non-zero eigenvalues of $A^T A$ are also singular values of A.

Lemma 3.1.8. *For every non-zero eigenvalue λ of $A^T A$ with corresponding eigenvector $v \in \mathbb{R}^d$, λ is positive and $\sigma := \sqrt{\lambda}$ is a singular value of A with corresponding left singular vector v and corresponding right singular vector $u := Av/\sigma \in \mathbb{R}^n$.*

Proof. We follow the corresponding part of the proof of Proposition 1.2 on pages 162 and 163 of [KV09]. Notice that $u^T A = \sigma^{-1} v^T A^T A = \sigma^{-1}\lambda v^T = \sigma v^T$ and $Av = \sigma Av/\sigma = \sigma u$ by definition of σ and u, so all that is to show is that σ and u are actually well-defined. For this, notice that

$$\begin{aligned} & A^T Av = \lambda v \\ \Rightarrow \quad & v^T A^T Av = \lambda v^T v \\ \Rightarrow \quad & ||Av||^2 = \lambda ||v||^2 \\ \Rightarrow \quad & \lambda = ||Av||^2/||v||^2 \geq 0 \end{aligned}$$

and thus λ is non-negative. By our assumption that $\lambda \neq 0$, this means that $\lambda > 0$. Thus, $\sigma := \sqrt{\lambda}$ and $u := Av/\sigma$ are well-defined. □

Notice that if v_i and v_j for $i \neq j$ are orthogonal eigenvectors of A with corresponding eigenvalues σ_i and σ_j, then $(Av_i/\sigma_i)^T Av_j\sigma_j = \sigma_i\sigma_j v_i^T v_j = 0$ and thus, $u_i := Av_i/\sigma_i$ and $u_j := Av_j/\sigma_j$ are also orthogonal.

By Lemma 2.3.4, $\text{rank}(A^T A) = \text{rank}(A)$, so Lemma 3.1.8 implies the existence of $r := \text{rank}(A)$ nonzero singular values $\sigma_1, \ldots, \sigma_r$ of A, together with r corresponding left singular vectors $u_1, \ldots, u_r$ and right singular vectors $v_1, \ldots, v_r$ which satisfy that all left singular vectors are pairwise orthogonal, and all right singular vectors are pairwise orthogonal. Assume that we extend $\{v_1, \ldots, v_r\}$ to an orthonormal basis of $\mathbb{R}^d$ by adding an orthonormal basis $v_{r+1}, \ldots, v_d$ of the kernel of A. Let $x \in \mathbb{R}^d$ be an arbitrary point from $\mathbb{R}^d$ and define α_i by $x = \sum_{i=1}^d \alpha_i v_i$ (such α_i must exist and are unique because the v_i form

a basis). Notice that $|\alpha_i|$ is the length of the projection of x into direction v_i because $v_i^T x = v_i^T \sum_{j=1}^d \alpha_j v_j = \alpha_i$. Then it holds that

$$Ax = A\sum_{i=1}^{d} \alpha_i v_i = \sum_{i=1}^{d} \alpha_i A v_i = \sum_{i=1}^{r} \alpha_i \sigma_i u_i = \sum_{i=1}^{r} \sigma_i u_i \alpha_i = \sum_{i=1}^{r} \sigma_i u_i v_i^T x = \left(\sum_{i=1}^{r} \sigma_i u_i v_i^T\right) x. \tag{3.8}$$

In other words, the matrix $\sum_{i=1}^r \sigma_i u_i v_i^T$ maps an arbitrary point x to Ax. This implies that $A = \sum_{i=1}^r \sigma_i u_i v_i^T$ and we have found our decomposition except for the fact that the sum only goes up to r instead of $\min\{d, n\}$. However, we can extend the u_i to a basis of $\mathbb{R}^n$ by adding a basis $\{u_{r+1}, \ldots, u_n\}$ of the kernel of A^T. Additionally, we set $\sigma_{r+1}, \ldots, \sigma_{\min\{n,d\}} := 0$. Then we achieve that $\{u_i\}_{i=1,\ldots,n}$ and $\{v_i\}_{i=1,\ldots,d}$ are orthonormal sets that span $\mathbb{R}^n$ and $\mathbb{R}^d$, $\sigma_i \geq 0$ are non-negative numbers and $A := \sum_{i=1}^{\min\{n,d\}} \sigma_i u_i v_i^T$.

Theorem 3.1.9 ([Wat10], Theorem 5.8.11 on page 391, and [KV09], Theorem 1.3 on pages 163-165)**.** *Let $A \in \mathbb{R}^{n\times d}$ be a nonzero matrix with rank r. Then there exists an orthonormal basis $v_1, \ldots, v_d$ of $\mathbb{R}^d$ and an orthonormal basis $u_1, \ldots, u_n$ of $\mathbb{R}^n$ and positive values $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0$ such that*

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

It holds that $A = \sum_{i=1}^{\min\{n,d\}} \sigma_i u_i v_i^T$.

Geometric interpretation of singular values and vectors. We have a short look at the interpretation of singular values. For this purpose, we loosely follow the introduction of Section 1.1 on pages 161-162 in [KV09]. Recall that the singular values are ordered such that $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0$. Also recall that we defined a basis $v_1, \ldots, v_d$ consisting of orthogonal unit vectors, where $v_1, \ldots, v_r$ are right singular vectors ordered according to the corresponding singular values, and the remaining vectors are a basis of the kernel of A.

Our first observation is that v_1, the right singular vector belonging to σ_1, maximizes the expression $||Av||$ among all unit vectors from $\mathbb{R}^d$. To see this, let x be an arbitrary unit vector. We can express x by $x = \sum_{i=1}^d \alpha_i v_i$. First notice that $\sum_{i=1}^d \alpha_i^2 = 1$ because x is a unit vector and $1 = ||x||^2 = \left|\left|\sum_{i=1}^d \alpha_i v_i\right|\right|^2 = \sum_{i=1}^d \alpha_i^2 ||v_i||^2 = \sum_{i=1}^d \alpha_i^2$ by the Pythagorean theorem 2.3.1. Now we observe that

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

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

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

$i \neq 2$ is the maximal choice. This argument works inductively. Furthermore, if we are given any v_i that maximize $||Av||$ in this fashion, then this is enough to ensure that the vectors are singular vectors ordered according to their singular values.

Theorem 3.1.10 (KV09, Theorem 1.3 on pages 163+164)**.** *For $A \in \mathbb{R}^{n \times d}$, find vectors such that*

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

Then all $v_i \in \mathbb{R}^d$ are right singular vectors of A with corresponding singular values $\sigma_1, \ldots, \sigma_d$, and it holds that $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_d$.

We have a look at the geometric interpretation of $||Av||$. For a point A_{i*} in A and a unit vector v, we notice that $||A_{i*}v||$ is the length of the vector that arises from projecting A_{i*} to the direction of v by Observation 3.1.1. Thus, Av contains these projection lengths (plus a sign), and $||Av||^2$ is the sum of the squared projection lengths.

We have seen that for a singular vector v_i, it holds that $||Av_i|| = \sigma_i$. This means that one interpretation of the singular value σ_i is that it is the square root of the sum of the squared lengths of all points in A when projected onto the direction of the right singular vector v_i. Singular values are thus a measure of the extent of the point set into the direction of the corresponding right singular vectors.

Recall that $\text{span}\{v\} := \{\alpha v \mid \alpha \in \mathbb{R}\}$ is set of all points spanned by a vector v. By Observation 3.1.3, the length of a vector A_{i*} satisfies that $||A_{i*}||^2 = \text{dist}^2(A_{i*}, \text{span}\{v\}) + ||A_{i*}v||^2$, which implies that

$$||Av||^2 = \sum_{i=1}^{n} ||A_{i*}v||^2 = \sum_{i=1}^{n} ||A_{i*}||^2 - \text{dist}^2(A_{i*}, \text{span}\{v\}).$$

As the length of the vectors is fixed, this means that maximizing $||Av||^2$ is equivalent to minimizing $\sum_{i=1}^{n} \text{dist}^2(A_{i*}, \text{span}\{v\})$, the sum of the squared distances of all points to the line spanned by v. Thus, v_1 is the vector minimizing the sum of the squared distances of all points in A to the line spanned by v_1. Again, this argument also works inductively.

Theorem 3.1.11 (KV09, Theorem 1.3 on pages 163+164)**.** *Find right singular vectors $v_i \in \mathbb{R}^d$ as in Theorem 3.1.10. For $k \leq d$, define $V_k := \text{span}\{v_1, \ldots, v_k\}$ as the span of the first k of these vectors. Then, V_k is a* best fit subspace *of dimension k, i. e.,*

$$V_k = \arg \min_{V \subsetneq \mathbb{R}^n, \dim(V) = k} \sum_{i=1}^{n} \text{dist}^2(A_{i*}, V).$$

Finally, we have a second look at Equation 3.3, which tells us that A_{j*} can be written as $\sum_{i=1}^{\min\{n,d\}} \sigma_i u_{ij} v_i$. Notice that $A_{j*}v_i = \sigma_i u_{ij}$ because $Av_i = \sigma_i u_i$. Thus, $\sigma_i u_{ij}$ is the length of the projection of A_{j*} to the line spanned by v_i. We get the projected point by multiplying this with v_i, i. e., the projection of A_{j*} to span$\{v_i\}$ is $\sigma_i u_{ij} v_i$. Thus, $\sum_{i=1}^{\min\{n,d\}} \sigma_i u_{ij} v_i$ is just the sum of the projections of A_{j*} to all vectors of the basis $v_1, \ldots, v_d$ which do not belong to the kernel of A, which makes sense. If we ignore upper terms of this sum and get $\sum_{i=1}^{k} \sigma_i u_{ij} v_i$, the resulting point is just the projection to the best fit subspace V_k.

Observation 3.1.12. *The matrix $A^{(k)} = \sum_{i=1}^{k} \sigma_i u_i v_i = UD^{(k)}V^T$ contains the projections of the row vectors of A to V_k as its rows.*

SVD and k-means. The best fit subspace from Theorem 3.1.11 gives us a computable lower bound on the k-means cost of a point set because the squared distances to k centers cannot be larger than the squared distances to a subspace of dimension k. This follows basically from the Pythagorean theorem 2.3.1 or the observations on the squared distances to subspaces that we showed earlier, based on the Pythagorean theorem.

Observation 3.1.13. *Let $A \in \mathbb{R}^{n \times d}$ be a matrix, let k be an integer and let C be a set of k centers. Then it holds that*

$$\text{dist}^2(A, V_k) \leq \text{cost}_{\ell_2^2}(A, C).$$

Proof. Let $\tilde{V}_k$ be the subspace spanned by the points in C. As V_k is the best fit subspace, we have that $\text{dist}^2(A, V_k) \leq \text{dist}^2(A, \tilde{V}_k)$. Furthermore, by Observation 3.1.3, we know that $\text{dist}^2(A_{i*}, c) = \text{dist}^2(A_{i*}, \pi_{\tilde{V}_k}(A_{i*})) + \text{dist}^2(\pi_{\tilde{V}_k}(A_{i*}), c)$ for all $c \in C \subset \tilde{V}_k$ and for all $i = 1, \ldots, n$. As $\sum_{i=1}^{n} \text{dist}^2(A_{i*}, \pi_{\tilde{V}_k}(A_{i*})) = \text{dist}^2(A, \tilde{V}_k)$ by Observation 3.1.2 and $\text{dist}^2(\pi_{\tilde{V}_k}(A_{i*}), c) \geq 0$, this implies $\text{dist}^2(A, \tilde{V}_k) \leq \text{cost}_{\ell_2^2}(A, C)$. □

Drineas, Frieze, Kannan, Vempala and Vinay [DFK+04] show that the best fit subspace V_k can also be used to reduce the dimensionality of the input data at the cost of a constant factor in the cost function. Recall that $A^{(k)}$ is the matrix containing the projections of all points to V_k, and $A^{(k)}_{j*}$ is the projection of the jth point to V_k. Let $C^*_{(k)} \subset V_k$ be an optimal center set for $A^{(k)}$, i. e., an optimal solution for the projected point set, and let C^* be an optimal center set for A.

Let A_{j*} be an input point and let c be its closest center in C^*, while c' is its closest center in $C^*_{(k)}$. Observation 3.1.3 says that by the Pythagorean theorem, $||A_{j*} - c'||^2 = ||A_{j*} - A^{(k)}_{j*}||^2 + ||A^{(k)}_{j*} - c'||^2$. By the above argumentation, we already know that $\sum_{j=1}^{n} ||A_{j*} - A^{(k)}_{j*}||^2 = \text{dist}^2(A, V_k) \leq \text{cost}_{\ell_2^2}(A, C)$, so we only have to worry about the second term.

Define $D_{j*} := A_{j*} - c$ for $j = 1, \ldots, n$. We can express D_{j*} by its representation in the singular vector basis $v_1, \ldots, v_d$, let the coefficients in this representation be $\alpha_{1j}, \ldots, \alpha_{dj} \in \mathbb{R}$, i. e., $D_{j*} = \sum_{i=1}^{d} \alpha_{ij} v_i$. Notice that $D^{(k)}_{j*} := \sum_{i=1}^{k} \alpha_{ij} v_i$ is the projection of D_{j*} to V_k, and that $||D^{(k)}_{j*}||$ is just the distance between A_{j*} and its closest center after both are projected down to V_k.

Furthermore, notice that $||D_{j*}^{(k)}||^2 \leq ||D_{j*}||^2$ by the Pythagorean theorem because the vectors $v_1, \ldots, v_d$ form an orthonormal basis. This implies that the (squared) distance between $A_{j*}^{(k)}$ and $\pi_{V_k}(c)$ is at most the distance between A_{j*} and c. As this holds for all points, we have $\text{dist}^2(A^{(k)}, \pi_{V_k}(C^*)) \leq \text{dist}^2(A, C^*)$, if we define $\pi_{V_k}(C^*) := \{\pi_{V_k}(c) \mid c \in C^*\}$ as the projection of the optimal center set for A to V_k.

Switching from the projections of C^* to the optimal center set for $A^{(k)}$ can only decrease the cost. Thus, $\text{dist}^2(A^{(k)}, C_{(k)}^*) \leq \text{dist}^2(A^{(k)}, \pi_{V_k}(C^*)) \leq \text{cost}_{\ell_2^2}(A, C)$, and consequently $\text{dist}^2(A, C_{(k)}^*) \leq 2 \cdot \text{cost}_{\ell_2^2}(A, C)$.

Theorem 3.1.14 (Drineas, Frieze, Kannan, Vempala, Vinay [DFK+04])**.** *Let $A \in \mathbb{R}^{n \times d}$ be a matrix, let k be an integer, let C be a set of k centers from $\mathbb{R}^d$. Denote the optimal solution of the k-means problem on $A^{(k)}$ by $C_{(k)}^*$. It holds that*

$$\text{cost}_{\ell_2^2}(A, C_{(k)}^*) \leq 2 \cdot \text{cost}_{\ell_2^2}(A, C).$$

Drineas et al. combine this dimensionality reduction with an algorithm that computes an optimal solution for the k-means problem in $\mathbb{R}^d$ with runtime $\mathcal{O}(n^{dk^2/2}) = \mathcal{O}(n^{k^3/2})$. Their output is then a 2-approximation. The next section shows that this dimensionality reduction can be improved.

3.2 The singular value decomposition revisited

In this chapter, we develop a dimensionality reduction based on the singular value decomposition. In contrast to the technique in Section 3.1.2, the dimensionality reduction presented here gives a $(1+\varepsilon)$-approximation guarantee. To achieve this, we change two things about the process presented in the last section. First, we project to a higher dimensional best fit subspace, namely V_m for $m = \lceil 17k/\varepsilon^2 \rceil$. Notice that this is independent of n in contrast to the Johnson-Lindenstrauss based approach described in Section 3.1.1. Second, in addition to the projected point set we store a constant value which is the length that the points 'lose' when they are projected down. In other words, if the input is given as a matrix A, we replace it by $A^{(m)}$ and store the constant $\Delta := ||A||_F^2 - ||A^{(m)}||^2 = \sum_{i=m+1}^{d} \sigma_i^2$.

Then we can prove that for every set of k centers C, adding the cost of clustering $A^{(m)}$ with C and the constant Δ gives a $(1+\varepsilon)$-approximation of the cost of clustering A with C. As this holds for every center set, finding a solution which is optimal for $A^{(m)}$ gives a $(1+\varepsilon)$-approximation for the optimal k-means cost of A.

Notice that the approximation statements do not hold on the level of single points or pair of points. They only hold for the cost of the whole point set, i. e., for the sum over all squared distances. This makes sense as we know from the lower bound in Section 3.1.1 that an approximation guarantee for all pairwise distances would imply a logarithmic dependency on the number of points n, and the above stated m does not depend on n (or $\log n$).

Table 3.1 lists dimensionality reduction results for the k-means problem. In addition to the dimensionality reduction based on the Johnson-Lindenstrauss Lemma and the SVD-based 2-approximation, it contains two results that appeared before the content of this

Authors	Year	Guarantee	Dimensions	Reference
Johnson, Lindenstrauss	1984	$1+\varepsilon$	$\Theta(\log n/\varepsilon^2)$	[JL84]
Drineas, Frieze, Kannan, Vempala, Vinay	1999	2	k	[DFK+04]
Boutsidis, Mahoney, Drineas	2009	$2+\varepsilon$	$\Theta(k\log(k/\varepsilon)/\varepsilon^2)$	[BMD09]
Boutsidis, Zouzias, Drineas	2010	$2+\varepsilon$	$\Theta(k/\varepsilon^2)$	[BZD10]
Feldman, Schmidt, Sohler	2013	$1+\varepsilon$	$\Theta(k/\varepsilon^2)$	[FSS13]
Cohen, Elder, Musco, Musco, Persu	2014	$1+\varepsilon$	$\lceil k/\varepsilon \rceil$	[CEM+14]

Table 3.1: Dimensionality reduction results for k-means clustering.

section. One is by Boutsidis, Mahoney and Drineas [BMD09], the other by Boutsidis, Zouzias, Drineas [BZD10]. The latter is based on random projections and projects to asymptotically the same number of dimension as we do. However, the result gives an approximation guarantee of $2+\varepsilon$ instead of $1+\varepsilon$. The preceding result [BMD09] is based on the singular value decomposition, but combines it with a sampling process that samples dimensions from the original dimensions. It has the advantage that the chosen dimensions are features of the original point set, yielding a better interpretability of the lower dimensional point set. The approximation guarantee is $2+\varepsilon$.

The dimensionality reduction by Cohen, Elder, Musco, Musco and Persu [CEM+14] was published in 2014 and is a further development of the reduction presented in this section. It achieves a an approximation guarantee of $1+\varepsilon$ while projecting to only $\lceil k/\varepsilon \rceil$ dimensions.

The results in this section and the results in Section 4.3 and Section 4.4 are joint work with Dan Feldman and Christian Sohler and have been published in [FSS13].

Projecting to V_m. Recall that $A^{(m)}$ is the projection of A to V_m. The plan of this section is the following. First, we study how the projections of A and $A^{(m)}$ to an arbitrary k-dimensional subspace V relate to each other. Second, we look at the squared distances between A or $A^{(m)}$ and V. Third, we consider the actual clustering cost of A and $A^{(m)}$ and show how to bound the error for a sufficiently large m.

More specifically, the first step is to show that $\pi_V(A_{i*})$ and $\pi_V(A^{(m)}_{i*})$ have similar average squared length, and that the average squared distance between $\pi_V(A_{i*})$ and $\pi_V(A^{(m)}_{i*})$ is small enough for our purposes.

Intuitively, projecting to a subspace means that we only keep the part of the points that goes into the direction of the subspace. By first projecting onto V_m, we lose all extent of the points which is orthogonal to V_m. If we then later want to project onto a different subspace V, we cannot recover the contribution of these orthogonal dimensions. However, if V is k-dimensional, then the worst that can happen is that we would have needed k of these $d-m+1$ lost dimensions, and that these are the dimensions where A had the highest extent among the lost dimensions. As we kept all contributions into the direction of the

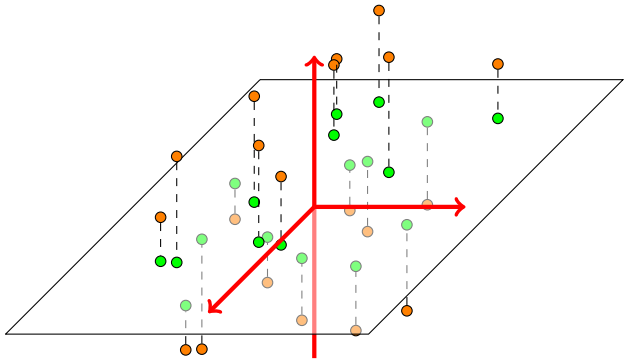


Figure 3.3: A point set is projected to a two-dimensional subspace.

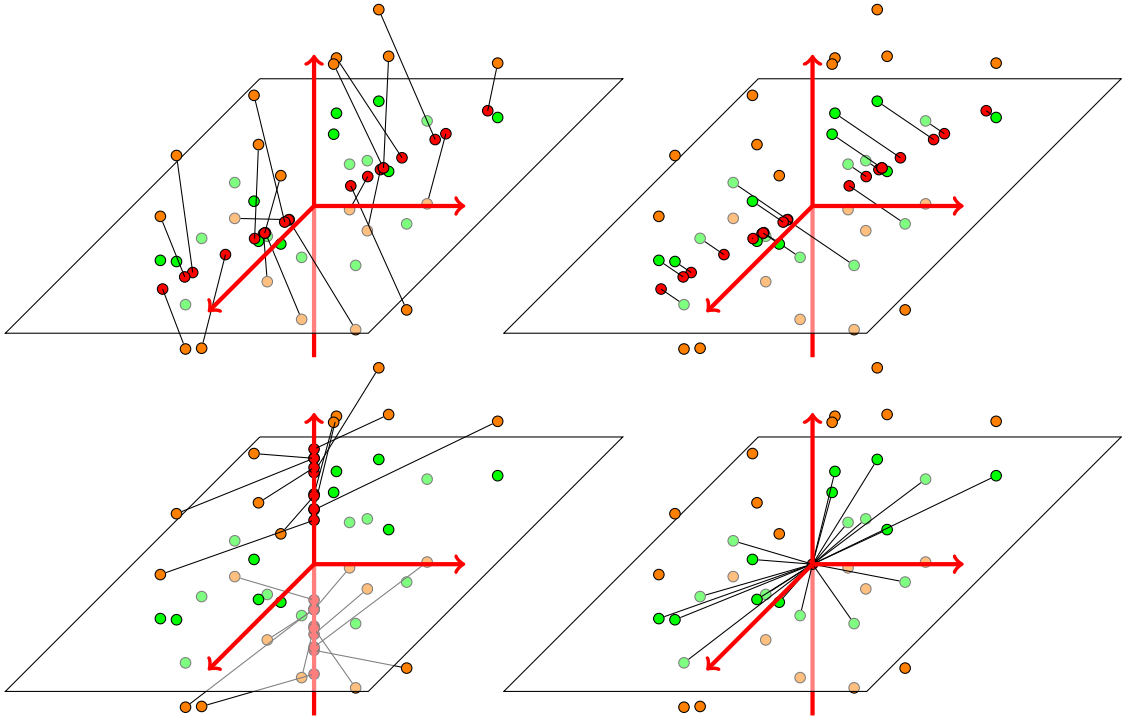


Figure 3.4: Both the original and the two-dimensional point set are projected onto a one-dimensional subspace. In the first case, the one-dimensional subspace lies within the two-dimensional subspace, so the projections actually coincide. The second case shows a worst-case example where the one-dimensional subspace is orthogonal to the two-dimensional subspace, and all information about the projection of the points to the one-dimensional subspace is lost.

first m most important singular vectors, the the sum of squared lengths of what we lose is bounded by $\sum_{i=m+1}^{m+k} \sigma_i^2$. Figure 3.3 shows an example of a three-dimensional point set that is projected to a two-dimensional subspace. Notice that the projection is not to the best fit subspace but simply to an example subspace. Figure 3.4 visualizes the best and worst case that can happen. In the case that we project to an orthogonal subspace, no information about the projection can be recovered, and the error equals the extent of the points in the direction of the query subspace.

If V is spanned by singular vectors, this statement is easy to show formally. Otherwise, we have to deal with subspaces that are spanned by vectors which are not orthogonal to our basis. This does not make the proof much harder, but a bit more technical. In the following lemma, we prove that we do not lose too much of the lengths of the projected points, and also that the projections are actually in some sense close together.

This lemma is the main technical step before we proceed to the dimensionality reduction for the k-means problem which is the goal of this section. Notice however that we are not using many facts about the k-means problem. The main properties that we need is that the solutions for the k-means problem live in k-dimensional subspaces of $\mathbb{R}^d$, and that the objective is based on the squared Euclidean distance. This is also true for projective clustering problems which we discuss in Chapter 8. In order to reuse the following lemma and the subsequent corollary and theorem later, we define it independent of the k-means problem and in fact even use a different letter, namely j, for the dimension of our solutions. For the application to the k-means problem, j is equal to k.

Lemma 3.2.1. *Let $A \in \mathbb{R}^{n \times d}$ and let V be a j-dimensional subspace and define $\pi_V : \mathbb{R}^d \to \mathbb{R}^d$ to be the mapping that projects every point to its projection in V. Let $\varepsilon \in (0,1)$ and $m \in \mathbb{N}$ with $m \geq \lceil j/\varepsilon \rceil$ and $n, d \geq m + j$. Then*

$$0 < \sum_{i=1}^{n} ||\pi_V(A_{i*})||^2 - \sum_{i=1}^{n} ||\pi_V(A_{i*}^{(m)})||^2 \leq \varepsilon \cdot \sum_{i=j+1}^{m+j} \sigma_i^2$$

and

$$\sum_{i=1}^{n} \left|\left|\pi_V(A_{i*}) - \pi_V(A_{i*}^{(m)})\right|\right|^2 \leq \varepsilon \cdot \sum_{i=j+1}^{m+j} \sigma_i^2.$$

Proof. Intuitively, the worst case that can happen is that V is the span of the singular vectors $v_{m+1}, \ldots, v_{m+j}$ because among those directions that we lose, these have the highest contribution to A. The loss is then exactly $\sum_{i=m+1}^{m+j} \sigma_i^2$. The main part of the proof is to show that this is indeed the worst case. Before that, we prove the that it holds that

$$\sum_{i=m+1}^{m+j} \sigma_i^2 \leq \varepsilon \sum_{i=j+1}^{m+j} \sigma_i^2.$$

There are $m + j - j - 1 + 1 = m$ terms in the second sum, and the first sum contains the smallest j of them. In the worst case, all summands are equal, and then $j \leq \varepsilon \cdot m$ implies

the inequality. To see this formally for not necessarily equal summands, notice that we have

$$\sum_{i=j+1}^{m} \sigma_i^2 \geq (m-j) \cdot \sigma_m^2 \geq \left(\frac{j}{\varepsilon} - j\right) \cdot \sigma_m^2 = \frac{1-\varepsilon}{\varepsilon} \cdot j \cdot \sigma_m^2 \geq \frac{1-\varepsilon}{\varepsilon} \sum_{i=m+1}^{m+j} \sigma_i^2$$

and thus

$$\frac{\sum_{i=m+1}^{m+j} \sigma_i^2}{\sum_{i=j+1}^{m+j} \sigma_i^2} \leq \frac{\sum_{i=m+1}^{m+j} \sigma_i^2}{\sum_{i=m+1}^{m+j} \sigma_i^2 + \frac{1-\varepsilon}{\varepsilon} \sum_{i=m+1}^{m+j} \sigma_i^2} = \frac{1}{1 + \frac{1-\varepsilon}{\varepsilon}} = \frac{1}{\frac{\varepsilon+1-\varepsilon}{\varepsilon}} = \varepsilon.$$

Recall that the right singular vectors of A can be extended to a orthonormal basis $\{v_1, \ldots, v_d\}$ of $\mathbb{R}^d$. Now let $\{a_1, \ldots, a_j\}$ be an orthonormal basis of V which can be extended by $\{a_{j+1}, \ldots, a_d\}$ to an orthonormal basis of $\mathbb{R}^d$. We start with the second statement of the lemma because the proof techniques are very similar and the second statement is slightly more difficult. Here, we are interested in the distance between the projections of A and $A^{(m)}$. As projecting is a linear transformation, we can equivalently look at the difference vectors between the points in A and $A^{(m)}$ and project those to V. Then we need to bound the sum of the squared lengths of these vectors.

Recall that we find the length of the projection of a vector to $\text{span}\{a_\ell\}$ by computing the scalar product of the vector with a_ℓ, and that the length of the projection to V is the sum of these scalar products for all a_ℓ. We now have that

$$\begin{aligned}
&\sum_{i=1}^{n} \left|\left|\pi_V(A_{i*}) - \pi_V(A_{i*}^{(m)})\right|\right|^2 = \sum_{i=1}^{n} \left|\left|\pi_V(A_{i*} - A_{i*}^{(m)})\right|\right|^2 = \sum_{i=1}^{n} \sum_{\ell=1}^{j} (\langle A_{i*} - A_{i*}^{(m)}, a_\ell \rangle)^2 \\
&= \sum_{\ell=1}^{j} \sum_{i=1}^{n} (\langle A_{i*} - A_{i*}^{(m)}, a_\ell \rangle)^2 = \sum_{\ell=1}^{j} ||(A - A^{(m)}) a_\ell||^2. \qquad (3.9)
\end{aligned}$$

Now define coefficients $\alpha_{\ell,s}$ as the unique values satisfying $a_\ell = \sum_{s=1}^{d} \alpha_{\ell,s} v_s$ for $\ell = 1, \ldots, d$ and $s = 1, \ldots, d$. Recall from the paragraph about the SVD definition and in particular from the explanations on page 41 that we have $A = \sum_{r=1}^{\min\{n,d\}} \sigma_r u_r v_r^T$ and $A^{(m)} = \sum_{r=1}^{m} \sigma_r u_r v_r^T$. Together, we get

$$\begin{aligned}
(A - A^{(m)}) a_\ell = \left(\sum_{r=m+1}^{\min\{n,d\}} \sigma_r u_r v_r^T \right) \sum_{s=1}^{d} \alpha_{\ell,s} v_s &= \sum_{r=m+1}^{\min\{n,d\}} \sum_{s=1}^{d} \sigma_r \alpha_{\ell,s} u_r v_r^T v_s \\
&= \sum_{r=m+1}^{\min\{n,d\}} \sigma_r \alpha_{\ell,r} u_r. \qquad (3.10)
\end{aligned}$$

We recall that if we project all points in A onto direction v_i and compute the sum of the squared lengths of the projections, we get σ_i^2. So, intuitively it makes sense that the

lengths that remains can be computed based on the singular values, and that it depends on how V lies compared to the singular vectors. We have now shown that

$$\sum_{i=1}^{n} \left|\left|\pi_V(A_{i*}) - \pi_V(A_{i*}^{(m)})\right|\right|^2 = \sum_{\ell=1}^{j} ||(A - A^{(m)})a_\ell||^2 = \sum_{\ell=1}^{j} || \sum_{r=m+1}^{\min\{n,d\}} \sigma_r \alpha_{\ell,r} u_r ||^2$$
$$= \sum_{\ell=1}^{j} \sum_{r=m+1}^{\min\{n,d\}} \sigma_r^2 \alpha_{\ell,r}^2.$$

We can only set j of the coefficients to one, and the expression gets largest when we set those $\alpha_{\ell,r}^2$ to one where we get the highest available σ_r^2 terms, i. e., $\alpha_{\ell,r} = 1$ for $r = m+1, \ldots, m+j$. To see that this is indeed true, consider the matrix $B \in \mathbb{R}^{d \times d}$ that contains $\alpha_{\ell,r}$ as the value in row ℓ and column r. Notice that the fact that all a_ℓ are orthonormal vectors implies that the rows of B are orthonormal vectors, too. Thus, B is an orthogonal matrix, and in particular, B has full rank and is invertible. The inverse of B is just B^T because $BB^T = I$ is just an alternative way of saying that the rows of B are orthonormal vectors. But the fact that B^T is the inverse of B also implies that $B^T B = I$, and this means that the columns of B are orthonormal, too. Thus, the d vectors $(\alpha_{1r}, \ldots, \alpha_{dr})$ are orthonormal. In particular, $\sum_{\ell=1}^{d} \alpha_{\ell r}^2 = 1$ for all $r = 1, \ldots, d$. It follows

$$\sum_{\ell=1}^{j} \sum_{r=m+1}^{\min\{n,d\}} \sigma_r^2 \alpha_{\ell,r}^2 = \sum_{r=m+1}^{\min\{n,d\}} \sigma_r^2 \sum_{\ell=1}^{j} \alpha_{\ell,r}^2 \leq \sum_{r=m+1}^{m+j} \sigma_r^2 \tag{3.11}$$

because the σ_r^2 are ordered decreasingly and because in total, $\sum_{\ell=1}^{j} \sum_{r=m+1}^{\min\{n,d\}} \alpha_{\ell,r}^2 = j$ as the a_ℓ are orthonormal.

For the second statement, notice that $\sum_{i=1}^{n} ||\pi_V(A_{i*})||^2 = \sum_{\ell=1}^{j} \left|\left|\sum_{r=1}^{d} \sum_{s=1}^{d} \sigma_r \alpha_{\ell,s} u_r v_r^T v_s\right|\right|^2$ and $\sum_{i=1}^{n} ||\pi_V(A_{i*}^{(m)})||^2 = \sum_{\ell=1}^{j} \left|\left|\sum_{r=1}^{m} \sum_{s=1}^{d} \sigma_r \alpha_{\ell,s} u_r v_r^T v_s\right|\right|^2$ by the same transformation steps as in (3.9) and (3.10). Thus, we directly get $\sum_{i=1}^{n} ||\pi_V(A_{i*})||^2 > \sum_{i=1}^{n} ||\pi_V(A_{i*}^{(m)})||^2$, and it also follows that

$$\sum_{i=1}^{n} ||\pi_V(A_{i*})||^2 - \sum_{i=1}^{n} ||\pi_V(A_{i*}^{(m)})||^2 = \sum_{\ell=1}^{j} \sum_{r=1}^{\min\{n,d\}} \sigma_r^2 \alpha_{\ell,r}^2 - \sum_{\ell=1}^{j} \sum_{r=1}^{m} \sigma_r^2 \alpha_{\ell,r}^2 = \sum_{\ell=1}^{j} \sum_{r=m+1}^{\min\{n,d\}} \sigma_r^2 \alpha_{\ell,r}^2.$$

□

The second step is now merely a corollary of Lemma 3.2.1 because the Pythagorean theorem allows us to reduce the error in the squared distances to V to the error in the projection lengths. Notice that we approximate the squared distances by computing the squared distances of $A^{(m)}$ to V and then adding the length that we have lost by projecting to V_m. In the worst case, the largest j of the added squared singular values are superfluous and become the error.

Corollary 3.2.2. *Let $A \in \mathbb{R}^{n\times d}$ and let V be a j-dimensional subspace. Let $\varepsilon \in (0,1)$ and $m \in \mathbb{N}$ with $m \geq \lceil j/\varepsilon \rceil$ and $n, d \geq m + j$. Let $A^{(m)} \in \mathbb{R}^{n\times m}$ be the projection of A to V_m, the best fit subspace of dimension m. Then it holds that*

$$\operatorname{dist}^2(A,V) \leq \operatorname{dist}^2(A^{(m)},V) + \sum_{i=m+1}^{\min\{n,d\}} \sigma_i^2 \leq \operatorname{dist}^2(A,V) + \varepsilon \cdot \sum_{i=j+1}^{m+j} \sigma_i^2 \leq (1+\varepsilon)\operatorname{dist}^2(A,V).$$

Proof. As Observation 3.1.3 states, we can split the length of a point into the squared length of its projection to a subspace plus the squared distance of the point to the subspace by using the Pythagorean theorem. For the points in A and $A^{(m)}$, this means that we get the following two equalities:

$$||A_{i*}||^2 = ||\pi_V(A_{i*})||^2 + \operatorname{dist}^2(A_{i*},V)$$
$$||A^{(m)}_{i*}||^2 = ||\pi_V(A^{(m)}_{i*})||^2 + \operatorname{dist}^2(A^{(m)}_{i*},V)$$

This still holds if we sum both inequalities over all n points in A and $A^{(m)}$, respectively. Recall that we can actually compute the loss in the squared length of the points because

$$\sum_{i=1}^{n} ||A_{i*}||^2 = ||A||_F^2 = \sum_{i=1}^{n} \sigma_i^2 \text{ and } \sum_{i=1}^{n} ||A^{(m)}_{i*}||^2 = ||A^{(m)}||_F^2 = \sum_{i=1}^{m} \sigma_i^2.$$

We gain that we can express the difference between the sum of the squared distances of the points in A and $A^{(m)}$ in the following convenient way.

$$\begin{aligned}\operatorname{dist}^2(A,V) - \operatorname{dist}^2(A^{(m)},V) &= ||A||_F^2 - ||A^{(m)}||_F^2 - \sum_{i=1}^{n} ||\pi_V(A_{i*})||^2 + \sum_{i=1}^{n} ||\pi_V(A^{(m)}_{i*})||^2 \\ &= \sum_{i=m+1}^{d} \sigma_i^2 - \sum_{i=1}^{n} ||\pi_V(A_{i*})||^2 + \sum_{i=1}^{n} ||\pi_V(A^{(m)}_{i*})||^2\end{aligned}$$

Lemma 3.2.1 tells us that $0 \leq \sum_{i=1}^{n} ||\pi_V(A_{i*})||^2 - \sum_{i=1}^{n} ||\pi_V(A^{(m)}_{i*})||^2 \leq \varepsilon \cdot \sum_{i=j+1}^{m+j} \sigma_i^2$, and so we can conclude that

$$\begin{aligned}\operatorname{dist}^2(A,V) - \operatorname{dist}^2(A^{(m)},V) &\leq \sum_{i=m+1}^{d} \sigma_i^2 \text{ and} \\ \operatorname{dist}^2(A,V) - \operatorname{dist}^2(A^{(m)},V) &\geq \sum_{i=m+1}^{d} \sigma_i^2 - \varepsilon \cdot \sum_{i=j+1}^{m+j} \sigma_i^2\end{aligned}$$

which gives the desired bounds. □

Now for the third step notice that the squared distances to a center set within V can be split into the squared distances to V and within V. We have bounded the first part in Corollary 3.2.2, and we can bound the second part by combining Lemma 3.2.1 with

a statement we saw in the last chapter. That will imply the dimensionality result. We formulate it for the distance to a non-empty and closed set C that is contained in a j-dimensional subspace. A solution for the k-means problem is a set of k points, so it is in particular a non-empty and closed set. Additionally, it is contained in a k-dimensional subspace, so the theorem applies to solutions for the k-means problem for $j = k$.

Theorem 3.2.3. *Let $A \in \mathbb{R}^{n\times d}$ be a matrix. Let $j \geq 1$ be an integer, let $\varepsilon \in (0,1)$ and let $m \in \mathbb{N}$ with $m \geq \lceil 18j/\varepsilon^2 \rceil$ and $n, d \geq m + j$. Then for any non-empty closed set C which is contained in a j-dimensional subspace, we have*

$$\left| \left(\mathrm{cost}_{\ell_2^2}(A^{(m)}, C) + \sum_{i=m+1}^{d} \sigma_i^2 \right) - \mathrm{cost}_{\ell_2^2}(A, C) \right| \leq \varepsilon \cdot \mathrm{cost}_{\ell_2^2}(A, C).$$

Proof. We use the fact that C lies in a j-dimensional subspace V. As before, let $\pi_V : \mathbb{R}^d \to V$ be the linear map that maps every point to its projection in V. The Pythagorean theorem allows us to split squared distances between a point A_{i*} and points in V. More precisely, Observation 3.1.3 says that it holds for all points $x \in V$ that

$$\mathrm{dist}^2(A_{i*}, x) = \mathrm{dist}^2(A_{i*}, V) + \mathrm{dist}^2(\pi_V(A_{i*}), x). \tag{3.12}$$

Thus, we have $\mathrm{dist}^2(A_{i*}, C) = \mathrm{dist}^2(A_{i*}, V) + \mathrm{dist}^2(\pi_V(A_{i*}), C)$, and the same statement holds for all $A^{(m)}_{i*}$. Now the remainder of the proof is to collect and combine the different statements we have already proven above. Recall that $\mathrm{cost}_{\ell_2^2}(A, C) = \sum_{i=1}^n \mathrm{dist}^2(A_{i*}, C)$ and $\mathrm{cost}_{\ell_2^2}(A^{(m)}, C) = \sum_{i=1}^n \mathrm{dist}^2(A^{(m)}_{i*}, C)$, respectively. By Corollary 3.2.2, we deduce that

$$\begin{aligned}
&\left| \mathrm{cost}_{\ell_2^2}(A^{(m)}, C) + \sum_{i=m+1}^{d} \sigma_i^2 - \mathrm{cost}_{\ell_2^2}(A, C) \right| \\
&= \Bigg| \sum_{i=1}^{n} \mathrm{dist}^2(A^{(m)}_{i*}, V) + \mathrm{dist}^2(\pi_V(A^{(m)}_{i*}), C) + \sum_{i=m+1}^{d} \sigma_i^2 \\
&\qquad\qquad - \sum_{i=1}^{n} \mathrm{dist}^2(A_{i*}, V) + \mathrm{dist}^2(\pi_V(A_{i*}), C) \Bigg| \\
&\leq \left| \sum_{i=1}^{n} \mathrm{dist}^2(\pi_V(A^{(m)}_{i*}), C) - \sum_{i=1}^{n} \mathrm{dist}^2(\pi_V(A_{i*}), C) \right| + \frac{\varepsilon^2}{18} \cdot \sum_{i=j+1}^{m+j} \sigma_i^2.
\end{aligned} \tag{3.13}$$

By Observation 3.1.13 and Observation 3.1.2, we know that $\mathrm{cost}_{\ell_2^2}(A, C) \geq \mathrm{dist}^2(A, V) \geq ||A - A^{(m)}||^2 = \left|\left|\sum_{r=1}^{d} \sigma_r u_r v_r^T - \sum_{r=1}^{j} \sigma_r u_r v_r^T\right|\right|^2 = \sum_{r=j+1}^{d} \sigma_r^2$. Lemma 3.2.1 says that for our choice of j, we have

$$\sum_{i=1}^{n} \left|\left| \pi_V(A_{i*}) - \pi_V(A^{(m)}_{i*}) \right|\right|^2 \leq \frac{\varepsilon^2}{18} \cdot \sum_{i=j+1}^{m+j} \sigma_i^2 \leq \frac{((17/18)\varepsilon)^2}{16} \cdot \frac{16 \cdot 18}{17 \cdot 17} \cdot \Lambda \leq \frac{((17/18)\varepsilon)^2}{16} \Lambda$$

for $\Lambda = \mathrm{cost}_{\ell_2^2}(A, C) \geq \sum_{i=j+1}^{d} \sigma_i^2 \geq \sum_{i=j+1}^{m+j} \sigma_i^2$. This means we can apply Lemma 2.3.2 with precision parameter $(17/18)\varepsilon$ which yields that

$$\begin{aligned}|\mathrm{cost}_{\ell_2^2}(\pi_V(A), C) - \mathrm{cost}_{\ell_2^2}(\pi_V(A^{(m)}), C)| &\leq (17/18)\varepsilon \cdot \max\{\Lambda, \mathrm{cost}_{\ell_2^2}(\pi_V(A), C)\} \\ &\leq (17/18)\varepsilon \cdot \mathrm{cost}_{\ell_2^2}(A, C).\end{aligned}$$

We can therefore complete the chain of inequalities started in (3.13) by

$$\begin{aligned}\left|\mathrm{cost}_{\ell_2^2}(A^{(m)}, C) + \sum_{i=m+1}^{d} \sigma_i^2 - \mathrm{cost}_{\ell_2^2}(A, C)\right| &\leq (17/18)\varepsilon \cdot \mathrm{cost}_{\ell_2^2}(A, C) + \frac{\varepsilon^2}{18} \cdot \mathrm{cost}_{\ell_2^2}(A, C) \\ &\leq \varepsilon \cdot \mathrm{cost}_{\ell_2^2}(A, C).\end{aligned}$$

□

We conclude the section by applying the theorem to the k-means problem and stating the dimensionality result explicitly.

Algorithm 3.1: Dimensionality reduction for k-means.

Algorithm k-`means approximation`(A,k,ε)
  Set $\varepsilon' = \varepsilon/3$ and $m = \lceil 18k/\varepsilon'^2 \rceil = \lceil 162k/\varepsilon^2 \rceil$;
  Compute the singular value decomposition $A = U\Sigma V^T$ and $A^{(m)} = U\Sigma^{(m)}V^T$;
  Compute an α-approximative solution C to the optimal solution on $A^{(m)}$;
  return C;

Lemma 3.2.4. *Let $A \in \mathbb{R}^{n\times d}$ be a matrix, let $k \in \mathbb{N}_{\geq 1}$, $\varepsilon \in (0,1)$ and $\alpha \in \mathbb{R}$ with $\alpha \geq 1$. Let $n, d \geq \lceil 162k/\varepsilon^2 \rceil + k$. Algorithm 3.1 computes an α'-approximation to the optimal solution for the k-means problem on A, where $1 \leq \alpha' \leq \alpha \cdot (1+\varepsilon/3)/(1-\varepsilon/3) \leq \alpha \cdot (1+\varepsilon)$. In particular, if $\alpha \leq 1 + \varepsilon/10$ and $\varepsilon < 1/7$, then it computes a $(1+\varepsilon)$-approximation.*

Proof. Let C^* be an optimal center set for A and let $C'_{(m)}$ the solution computed by Algorithm 3.1. By Theorem 3.2.3, we know that

$$\left|\left(\mathrm{cost}_{\ell_2^2}(A^{(m)}, C) + \sum_{i=m+1}^{d} \sigma_i^2\right) - \mathrm{cost}_{\ell_2^2}(A, C)\right| \leq (\varepsilon/3) \cdot \mathrm{cost}_{\ell_2^2}(A, C)$$

holds for both $C = C^*$ and $C = C'_{(m)}$ because the dimensionality reduction is performed with $\varepsilon' = \varepsilon/3$ and m is set to $\lceil 18/\varepsilon'^2 \rceil$, respectively. In particular, we have that $\mathrm{cost}_{\ell_2^2}(A, C) \leq 1/(1-\varepsilon/3)\left(\mathrm{cost}_{\ell_2^2}(A^{(m)}, C) + \sum_{i=m+1}^{d} \sigma_i^2\right)$. Notice that $\varepsilon < 1$ and this implies $1/(1-\varepsilon/3) = 1 + \varepsilon/(3-\varepsilon) \leq 1 + \varepsilon/2$. Now we bound the cost of $C'_{(m)}$. Let $C^*_{(m)}$

be an optimal solution for $A^{(m)}$. Then we have that

$$\begin{aligned}
&\mathrm{cost}_{\ell_2^2}(A, C'_{(m)}) \leq \Big(\frac{1}{1-\varepsilon/3}\Big)\Big(\mathrm{cost}_{\ell_2^2}(A^{(m)}, C'_{(m)}) + \sum_{i=m+1}^{d} \sigma_i^2\Big) \\
&\leq \Big(\frac{1}{1-\varepsilon/3}\Big)\alpha\Big(\mathrm{cost}_{\ell_2^2}(A^{(m)}, C^*_{(m)}) + \sum_{i=m+1}^{d} \sigma_i^2\Big) \\
&\leq \Big(\frac{1}{1-\varepsilon/3}\Big)\alpha\Big(\mathrm{cost}_{\ell_2^2}(A^{(m)}, C^*) + \sum_{i=m+1}^{d} \sigma_i^2\Big) \\
&\leq \Big(\frac{1}{1-\varepsilon/3}\Big)\alpha\Big(1+\frac{\varepsilon}{3}\Big) \cdot \mathrm{cost}_{\ell_2^2}(A, C^*) \leq \Big(1+\frac{\varepsilon}{2}\Big)\alpha\Big(1+\frac{\varepsilon}{3}\Big) \cdot \mathrm{cost}_{\ell_2^2}(A, C^*) \\
&\leq \alpha \cdot (1+\varepsilon) \cdot \mathrm{cost}_{\ell_2^2}(A, C^*).
\end{aligned}$$

Additionally, if $\varepsilon < 1/7$, then we can bound $1/(1-\varepsilon/3) = 1+\varepsilon/(3-\varepsilon) \leq 1+\varepsilon/(3-1/7) = 1 + (7/20)\varepsilon$, and for $\alpha = 1 + \varepsilon/10$, the we get

$$\begin{aligned}
\mathrm{cost}_{\ell_2^2}(A, C'_{(m)}) &\leq \Big(\frac{1}{1-\varepsilon/3}\Big)\alpha\Big(1+\frac{\varepsilon}{3}\Big) \cdot \mathrm{cost}_{\ell_2^2}(A, C^*) \\
&\leq \Big(1+\frac{7}{20}\varepsilon\Big)\Big(1+\frac{\varepsilon}{10}\Big)\Big(1+\frac{\varepsilon}{3}\Big) \cdot \mathrm{cost}_{\ell_2^2}(A, C^*) \\
&\leq \Big(1+\frac{7}{20}\varepsilon+\frac{1}{3}\varepsilon+\frac{7}{20\cdot 3}\varepsilon^2\Big)\Big(1+\frac{\varepsilon}{10}\Big) \cdot \mathrm{cost}_{\ell_2^2}(A, C^*) \\
&\leq \Big(1+\frac{4}{5}\varepsilon\Big)\Big(1+\frac{\varepsilon}{10}\Big) \cdot \mathrm{cost}_{\ell_2^2}(A, C^*) \\
&\leq \Big(1+\frac{5}{50}\varepsilon+\frac{40}{50}\varepsilon+\frac{4}{50}\varepsilon^2\Big) \cdot \mathrm{cost}_{\ell_2^2}(A, C^*) \\
&\leq (1+\varepsilon) \cdot \mathrm{cost}_{\ell_2^2}(A, C^*).
\end{aligned}$$

□

3.2.1 Weighted input sets

Assume that the input contains a weight function $\{1, \ldots, n\} \to \mathbb{N}^+$ that assigns a positive integer weight to each input point in addition to the input matrix $A \in \mathbb{R}^{n \times d}$. We can still use the dimensionality reduction, but we have to work around the fact that the singular value decomposition as defined here does not apply to weighted point sets. Let $W := \sum_{i=1}^{n} w(i)$ be the sum of all weights. We define $E \in \mathbb{R}^{W \times d}$ as an expanded version of A in the following way. For each input point A_{i*}, E contains $w(i)$ copies in consecutive rows. The matrix E thus has W rows. Notice that E and the weighted points in A have the same clustering behaviour, i. e.,

$$\sum_{i=1}^{n} w(i)||A_{i*} - z||^2 = \sum_{i=1}^{n}\sum_{\ell=1}^{w(i)} ||A_{i*} - z||^2 = \sum_{i=1}^{W} ||E_{i*} - z||^2$$

for any point $z \in \mathbb{R}^d$. Thus, it is possible to replace A by E and then apply the dimensionality reduction. However, E has a pseudopolynomial size and applying the singular value decomposition to it is potentially inefficient. Instead, we imitate computing the singular value decomposition by computing it for a different matrix F. In the ith row of F, we store $\sqrt{w(i)} \cdot A_{i*}$. Notice that F does *not* have the same clustering behaviour as A and E. For example, if a center coincides with a point A_{i*} with $w(i) \geq 2$, then it induces no cost, but the point $\sqrt{w(i)} \cdot A_{i*}$ induces cost. So, we cannot replace E by F for the whole process, but we can use F to compute the best fit subspace for E. This is true because E and F have the same left singular vectors, and the same singular values.

Let $F_1 \in \mathbb{R}^{(W-w(1)+1)\times d}$ be the matrix where only the first $w(1)$ rows are replaced and the remaining rows are identical to E. Let $u \in \mathbb{R}^W$ and $v \in \mathbb{R}^d$ be a pair of left and right singular vectors for E with singular value σ, i. e., it holds that $Ev = \sigma u$ and $u^T E = \sigma v^T$ for $u = (u_1, \ldots, u_W)^T$ and $v = (v_1, \ldots, v_d)^T$. By definition, we have that

$$Ev = \begin{pmatrix} — & A_{1*} & — \\ — & A_{1*} & — \\ & \vdots & \\ — & A_{1*} & — \\ — & E_{(w(1)+1)*} & — \\ & \vdots & \\ — & E_{W*} & — \end{pmatrix} v = \begin{pmatrix} A_{1*}^T v \\ A_{1*}^T v \\ \vdots \\ A_{1*}^T v \\ E_{(w(1)+1)*}^T v \\ \vdots \\ E_{W*}^T v \end{pmatrix} = \begin{pmatrix} \sigma u_1 \\ \sigma u_2 \\ \vdots \\ \sigma u_{w(1)} \\ \sigma u_{w(1)+1} \\ \vdots \\ \sigma u_W \end{pmatrix}.$$

Notice that $u_1 = u_2 = \ldots = u_{w(1)}$ because $\sigma u_i = A_{1*}^T v$ for all $i \in \{1, \ldots, w(1)\}$. Now

$$u^T \begin{pmatrix} | & & | \\ E_{*1} & \cdots & E_{*W} \\ | & & | \end{pmatrix} = \left(\sum_{i=1}^{W} u_i E_{i1}, \sum_{i=1}^{W} u_i E_{i2}, \cdots, \sum_{i=1}^{W} u_i E_{id} \right) = (\sigma v_1, \sigma v_2, \cdots, \sigma v_d)$$

in particular implies that

$$\sum_{i=1}^{W} u_i E_{ij} = w(1) u_1 A_{1j} + \sum_{i=w(1)+1}^{W} u_i E_{ij} = \sigma v_j. \tag{3.14}$$

We define a different left singular vector u'. It is apparent that u cannot be a left singular vector of F_1 since it has the wrong dimension. We define the vector $u' := (\sqrt{w(1)} u_1, u_{w(1)+1}, \cdots, u_W) \in \mathbb{R}^{W-w(1)+1}$. Now we check that v and u' form a pair of left and right singular vectors for F_1, and that the corresponding singular value is still σ. We get that $F_1 v = \sigma u'$ by

$$F_1 v = \begin{pmatrix} — & \sqrt{w(1)} A_{1*} & — \\ — & E_{(w(1)+1)*} & — \\ & \vdots & \\ — & E_{W*} & — \end{pmatrix} v = \begin{pmatrix} \sqrt{w(1)} A_{1*}^T v \\ E_{(w(1)+1)*}^T v \\ \vdots \\ E_{W*}^T v \end{pmatrix} = \begin{pmatrix} \sigma \sqrt{w(1)} u_1 \\ \sigma u_{w(i)+1} \\ \vdots \\ \sigma u_W \end{pmatrix} = \sigma u'.$$

Algorithm 3.2: Dimensionality reduction for k-means with weighted input points.

Algorithm k-means approximation(A,w,k,ε)
  Set $\varepsilon' = \varepsilon/3$ and $m = \lceil 18k/\varepsilon'^2 \rceil = \lceil 162k/\varepsilon^2 \rceil$;
  Obtain F by multiplying each row i in A with $\sqrt{w(i)}$;
  Compute the singular value decomposition of F;
  Let $V^{(m)}$ contain the first m right singular values as its columns;
  Compute $A_w^{(m)} := AV^{(m)}V^{(m)}$;
  Compute an α-approximative solution C for $A_w^{(m)}$ with w;
  return C;

By Equality 3.14, we see that $u'^T F_1 = \sigma v^T$ since

$$\begin{aligned}
u'^T F_1 =& u'^T \begin{pmatrix} — & \sqrt{w(1)}A_{1*} & — \\ — & E_{(w(1)+1)*} & — \\ & \vdots & \\ — & E_{W*} & — \end{pmatrix} \\
=& \Big(u'_1\sqrt{w(1)}A_{11} + \sum_{i=w(1)+1}^{W} u_i E_{i1}, \cdots, u'_1\sqrt{w(1)}A_{1d} + \sum_{i=w(1)+1}^{W} u_i E_{id} \Big) \\
=& \Big(u_1 w(1) A_{11} + \sum_{i=w(1)+1}^{W} u_i E_{i1}, \cdots, u_1 w(1) A_{1d} + \sum_{i=w(1)+1}^{W} u_i E_{id} \Big) \\
=& (\sigma v_1, \ldots, \sigma v_d) = \sigma v^T.
\end{aligned}$$

Thus, for every left singular vector $v \in \mathbb{R}^n$ of E with singular value σ there exists a $u' \in \mathbb{R}^{W-w(i)+1}$ such that v and u' form a pair of singular vectors for F_1, and the singular value is σ. Notice that E and F_1 have the same rank, so they have the same number of positive singular values by Theorem 3.1.9. This means that the positive singular values (and right singular vectors) are identical. We can iteratively repeat this argument for every $i \in [n]$, replacing the $w(i)$ rows with copies of A_{i*} by one row with $\sqrt{w(i)}A_{i*}$ until we reach F. As all steps preserve the singular values and right singular vectors, we get that E and F have the same singular values and right singular vectors.

Since the best fit subspace is spanned by right singular vectors, this implies that E and F have the same best fit subspace. In order to compute the best fit subspace of dimension $m \in [d]$ for E we can thus use a singular value decomposition of F, say $F = UDV^T$, and return V_m. Let $V^{(m)}$ be the matrix containing the first m right singular values as the columns. Then $EV^{(m)}$ contains the projection lengths of all points in E to the best fit subspace, and $EV^{(m)}V^{(m)}$ contains the actual projected points. Instead of projecting E we can project A directly and weight the output, so the resulting m-dimensional point set is given by $AV^{(m)}V^{(m)}$. This yields Algorithm 3.2 for applying the dimensionality reduction to weighted input points for the k-means problem.

4 Small coresets for the k-means problem

Coresets are a fundamental concept to deal with large amounts of data. Their idea is to reduce the size of the input dramatically while keeping the main characteristics of the input set, at least with regard to the objective function at hand. More precisely, a (weighted) set S is a coreset for an input set P if for all possible solutions of an optimization problem, the objective function is approximately the same for S and P.

The important virtue of a coreset is its size. If the coreset is (dramatically) smaller than P, then computing it can be used as a preprocessing step before performing the actual optimization. We have a more detailed look at this and at applications of coresets in distributed and data stream settings in Section 4.1.1. Then, we look at the history of coresets for the k-means problem in Section 4.2.3.

In Section 4.3 and Section 4.4 we see two new methods to construct coresets. The main feature of these constructions is that they produce coresets that contain a number of coreset points which is independent on both the number and the dimension of the input points. The results in these sections as well as the results in Section 3.2 are joint work with Dan Feldman and Christian Sohler and have been published in [FSS13].

4.1 Coresets for the k-means problem

The concept of coresets is not restricted to clustering problems or even geometric problems, but as we are mostly concerned with the k-means problem in this chapter, we specifically define coresets for k-means for the time being.

Coresets have been proposed and used in different versions. The now commonly used definition was proposed by Har-Peled and Mazumdar [HPM04]. They specified coresets for the k-means and k-median problem and we state the definition for the k-means problem here. Notice that the definition allows that the input is weighted, for unweighted inputs we can just set all weights to one. We define the term $(1+\varepsilon)$-coresets so that we can specify coresets with different error parameters. However, if we do not need this flexibility, we usually abbreviate to just coreset.

Definition 4.1.1 (Coresets, Har-Peled, Mazumdar, [HPM04])**.** *Let $P \subsetneq \mathbb{R}^d$ be a finite set of points and let $w_1 : P \to \mathbb{R}$ be a function assigning a weight to each point in P. Let $\varepsilon \in (0,1)$. A finite set $S \subsetneq \mathbb{R}^d$ and a weight function $w_2 : S \to \mathbb{R}$ form a (strong)* $(1+\varepsilon)$-coreset *for P if for all sets of k centers $C \subsetneq \mathbb{R}^d$, it holds that*

$$(1-\varepsilon)\sum_{x\in P} w_1(x)\min_{c\in C}||x-c||^2 \leq \sum_{y\in S} w_2(y)\min_{c\in C}||y-c||^2 \leq (1+\varepsilon)\sum_{x\in P} w_1(x)\min_{c\in C}||x-c||^2.$$

Typically, the aim of a coreset construction algorithm is to compute a coreset of sublinear size, preferably of a size that is polylogarithmic in the input size. Some problems, including the k-means problem, admit coreset sizes that are independent of the number of input points. If the coreset size is constant or depends only on parameters that we assume to be constant, then the k-means problem can be solved on the coreset in constant time, even optimally. In this way, every polynomial coreset construction gives us a fixed-parameter approximation algorithm for the k-means problem.

Parameters that can influence the coreset size are the dimension of the points, the precision parameter ε, and the failure probability δ in case that the coreset construction is randomized. In this chapter, we develop coresets for the k-means problem that have a noticably small dependence on the dimension of the input point set. More precisely, while the coresets still depend on the dimension because they consist of d-dimensional points, the *number* of points will not depend on the dimension. This type of coreset is thus particularly well suited in the context of large and high-dimensional point sets.

Coresets with offset. As an easy example, reconsider the k-means problem for $k = 1$. Assume we are given a data set P and want to compute a summary that allows to (approximately) compute the 1-means cost of P. We can represent P by two points which have the centroid $\mu(P)$ of P as their centroid, and which together have the same sum of squared distances to $\mu(P)$ as all the points in P. More precisely, we represent P by two points $x = \mu(P) - \mathfrak{d} \cdot u$ and $y = \mu(P) + \mathfrak{d} \cdot u$ where u is an arbitrary unit vector, and $\mathfrak{d} := \sqrt{\operatorname{dist}^2(P, \mu(P))/2}$, and weight both points by $|P|/2$. Notice that the 1-means cost of $\{x, y\}$ is just $2\mathfrak{d}^2 = \operatorname{dist}^2(P, \mu(P))$, and that clustering x and y with an arbitrary center z costs

$$2\mathfrak{d}^2 + 2 \cdot (|P|/2) \cdot \operatorname{dist}^2(\mu(P), z) = \operatorname{dist}^2(P, \mu(P)) + |P| \operatorname{dist}^2(\mu(P), z) = \operatorname{dist}^2(P, z).$$

So, $\{x, y\}$ is a coreset for the 1-means problem on P, and it is even exact. However, it seems a bit unnatural how we force the constant cost into the coreset. Instead, we could simply store the centroid $\mu(P)$ as the (only) coreset point, weighted by $|P|$, and keep track of the constant cost $\operatorname{dist}^2(P, \mu(P))$ separately. Strictly speaking, we then no longer have a coreset according to the above definition, because we have to store a constant in addition to the coreset point. In Section 4.3 and Section 4.4, we study the following slight generalization of the coreset definition which allows us to store this type of extra information.

Definition 4.1.2 (Coresets with offsets, [FSS13])**.** *Let $P \subsetneq \mathbb{R}^d$ be a finite set of points and let $w_1 : P \to \mathbb{R}$ be function assigning a weight to each point in P. Let $\varepsilon \in (0, 1)$. A finite set $S \subsetneq \mathbb{R}^d$, a weight function $w_2 : S \to \mathbb{R}$ and a constant $\mathfrak{C} \in \mathbb{R}$ form a* coreset with offset *for P if for all sets of k centers $C \subsetneq \mathbb{R}^d$, it holds that*

$$(1-\varepsilon) \sum_{x \in P} w_1(x) \min_{c \in C} ||x-c||^2 \leq \sum_{y \in S} w_2(y) \min_{c \in C} ||y-c||^2 + \mathfrak{C} \leq (1+\varepsilon) \sum_{x \in P} w_1(x) \min_{c \in C} ||x-c||^2.$$

Now we notice that given a point set P, the centroid $\mu(P)$ with weight $|P|$ and the constant $\operatorname{dist}^2(P, \mu(P))$ form a coreset with offset for the 1-means problem on P.

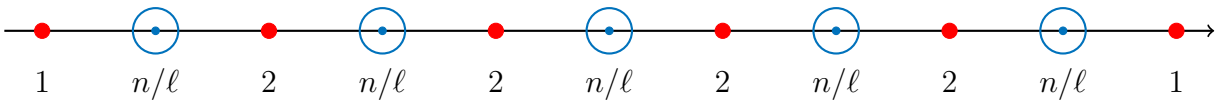


Figure 4.1: An example with the sole purpose to illustrate the *strong* coreset property.

This idea works because we can use Lemma 2.4.1 whenever we want to cluster a point set with one center. For $k > 1$, we do not know in advance which points will choose which cluster center, so we cannot simply summarize with centroids. In Section 4.3, we will see a way how to generalize the idea to k-means with $k > 1$.

Centroid sets and weak coresets. Coresets as defined in Definition 4.1.1 and 4.1.2 are also referred to as *strong* coresets because they demand that the approximation guarantee holds for *every* set of centers. For designing an approximation algorithm, this is not always necessary, and other forms of summaries of point sets have been developed. These are sometimes referred to as *weak* coresets, but there is no commonly agreed upon definition of this term. The crucial idea behind weak coresets is the possibility to compute $(1+\varepsilon)$-approximations for the original point set based on the weak coreset, and the important distinction to strong coresets is that weak coresets usually do not guarantee approximations for all sets of centers. Examples for weak coresets can be found in [BHPI02, HPV02, FMS07, FL11a], and we discuss at least the first of them in a bit more detail in the second paragraph of Section 4.2.3. In the context of k-means clustering, the joint use with an (approximate) *centroid set* makes sense. An (approximate) centroid set is a set of points in $\mathbb{R}^d$ that can supersede the $\mathbb{R}^d$ as the domain for choosing centers without introducing too much error, e. g., it contains a $(1+\varepsilon)$-approximative solution. It may satisfy additional conditions. Matoušek [Mat00] proposed a centroid set that was used several times in other works, and [ES04] and [FMS07] also propose centroid sets, but we do not go into more detail on this topic.

4.1.1 Applications of strong coresets

Coresets are point sets that behave like a much larger set of points, at least with regard to a cost function. In the case of k-means, strong coresets provide an approximation guarantee for every choice of centers from $\mathbb{R}^d$. We review the three main applications of this.

Approximation algorithms. The ability to reduce the size of a point set while keeping its main characteristics directly leads to a speed-up of algorithms which can then be run on the reduced set instead of the large original point set. Notice that it is important that we have the strong coreset property which gives us a guarantee for every choice of centers. In particular, it is not enough to demand that the cost of the *optimal solution* is preserved. Figure 4.1 illustrates this with a toy example. Assume that we have ℓ blue 'clusters' and that the optimal solution consists of the blue points. Assume that each cluster has a cost

of $\mathfrak{D}$ (the points are suitably distributed within the blue circle), so the total optimum cost is $\ell \cdot \mathfrak{D}$. Additionally assume that the distance between the blue and red centers is just $\sqrt{\mathfrak{D}/2}$. Then assigning the two red points to the left and right of a blue center exactly costs $2 \cdot \mathfrak{D}/2$. In other words, replacing the points by the $2 \cdot \ell$ red points exactly preserves the cost of the optimal centers. However, the red points can be clustered much cheaper by placing ℓ centers on top of red points and leaving one single red point alone, which then costs $(2\mathfrak{D}/2)^2 = \mathfrak{D}^2$ to assign to the nearest red point. This optimal solution for the red points does not yield a good approximation to the optimal solution for the blue points.

The strength of the strong coreset property is that it does not only preserve the cost of the optimal solution, but it also prevents other solutions from becoming drastically cheaper. That is why we can just run an approximation algorithm on the coreset instead of the original point set. In order to obtain a $(1+\varepsilon)$-approximation, we have to appropriately adjust the coreset error and the approximation error of the approximation algorithm. The following lemma shows one way to do this in the case of a $(1+\varepsilon)$-approximation algorithm. It is quite similar to Lemma 3.2.4 on combining dimensionality reduction with an approximation algorithm.

Lemma 4.1.3. *Let $P \subsetneq \mathbb{R}^d$ be a set of n points, let $k \in \mathbb{N}$ and $\varepsilon \in (0,1)$, and let $S \subsetneq \mathbb{R}^d$ with weight function $w : S \to \mathbb{R}$ be a $(1+\varepsilon/3)$-coreset for P. Let C^* be an optimal solution of the k-means problem on P, and let C_S^* be an optimal solution for the weighted k-means problem on S.*

Then it holds that $|\mathrm{cost}_{\ell_2^2}(P, C_S^*) - \mathrm{cost}^*_{\ell_2^2}(P)| \leq \varepsilon \cdot \mathrm{cost}^*_{\ell_2^2}(P)$. If $\varepsilon < 1/7$ and C'_S is a $(1+\varepsilon/10)$-approximate solution for S, then $\mathrm{cost}_{\ell_2^2}(P, C'_S) \leq (1+\varepsilon) \cdot \mathrm{cost}_{\ell_2^2}(P, C^*)$.

Proof. It holds $\mathrm{cost}_{\ell_2^2}(P, C^*) \leq \mathrm{cost}_{\ell_2^2}(P, C_S^*)$ because C^* is an optimal solution. Furthermore, by the definition of coresets, $(1-\varepsilon/3) \cdot \mathrm{cost}_{\ell_2^2}(P, C_S^*) \leq \mathrm{cost}_{\ell_2^2,w}(S, C_S^*)$. By $\varepsilon < 1$ and basic transformations, it holds that $1/(1-\varepsilon/3) = 1 + \varepsilon/(3-\varepsilon) \leq 1 + \varepsilon/2$. We thus have

$$\begin{aligned}\mathrm{cost}_{\ell_2^2}(P, C_S^*) &\leq (1+\varepsilon/2) \cdot \mathrm{cost}_{\ell_2^2,w}(S, C_S^*) \leq (1+\varepsilon/2) \cdot \mathrm{cost}_{\ell_2^2,w}(S, C^*) \\ &\leq (1+\varepsilon/2) \cdot ((1+\varepsilon/3) \cdot \mathrm{cost}_{\ell_2^2}(P, C^*)) \leq (1+\varepsilon) \cdot \mathrm{cost}_{\ell_2^2}(P, C^*).\end{aligned}$$

Additionally, if we have a $(1+\varepsilon/10)$-approximate solution for S, we get that

$$\begin{aligned}\mathrm{cost}_{\ell_2^2}(P, C'_S) &\leq \left(1 + \frac{\varepsilon}{3-\varepsilon}\right)\mathrm{cost}_{\ell_2^2,w}(S, C'_S) \leq \left(1 + \frac{\varepsilon}{3-1/7}\right)\mathrm{cost}_{\ell_2^2,w}(S, C'_S) \\ &= (1+7\varepsilon/20)\mathrm{cost}_{\ell_2^2,w}(S, C'_S) \leq (1+7\varepsilon/20)[(1+\varepsilon/10) \cdot \mathrm{cost}_{\ell_2^2,w}(S, C_S^*)] \\ &\leq (1+[7/20+1/10+7/200]\varepsilon) \cdot \mathrm{cost}_{\ell_2^2,w}(S, C_S^*) \\ &\leq (1+(97/200)\varepsilon) \cdot \mathrm{cost}_{\ell_2^2,w}(S, C^*) \\ &< (1+(1/2)\varepsilon) \cdot [(1+\varepsilon/3) \cdot \mathrm{cost}_{\ell_2^2}(P, C^*)] \\ &\leq (1+(1/2+1/3+1/6)\varepsilon) \cdot \mathrm{cost}_{\ell_2^2}(P, C^*) = (1+\varepsilon) \cdot \mathrm{cost}^*_{\ell_2^2}(P).\end{aligned}$$

□

Using a coreset instead of the original point set can speed up approximation algorithms significantly. For example, if we compute a coreset S and then use the algorithm by Kumar, Sabharwal and Sen [KSS10] with its running time $\mathcal{O}(nd2^{(k/\varepsilon)^{\mathcal{O}(1)}})$, computing the approximation on S takes $\mathcal{O}(|S|d2^{(k/\varepsilon)^{\mathcal{O}(1)}})$ time, which is polynomial in the dimension and independent of n. Of course, the total running time includes the time to compute the coreset itself. Depending on the running time of the coreset construction and depending on the desired approximation guarantee, it can be interesting to choose different approximation algorithms from Table 2.1 on page 15. Actually, the list contains the approximation algorithms in [Che09, FS05, FMS07, FL11a, HPM04, HPK07] which use coresets to compute the approximation. This also shows in the running times which typically consist of the running time for the coreset construction plus the running time on the coreset. For example, Har-Peled and Mazumdar compute a coreset and then use the algorithm by Matoušek [Mat00], and Chen computes a coreset and uses the algorithm by Kumar, Sabharwal and Sen [KSS10].

Even in the context of heuristics coresets are interesting because we would usually expect the heuristic to perform similar steps on the coreset as on the original point set, so running it on a coreset makes sense. However, for the latter the coreset construction has to be sufficiently practical.

Streaming algorithms. The streaming setting is a new algorithmic paradigm developed in the emerging field of big data. A streaming algorithm is only allowed to read the data once, and it has a limited storage capacity, usually assumed to be polylogarithmic in the input size, i. e., in the length of the stream.

Notice that a coreset computation alone does not suffice to solve a problem in the streaming setting, because the computation of the coreset might need random access to the data. However, when we know a coreset computation for a problem, we can read chunks of the data, compute coresets of the chunks and only keep these coresets. If the storage for keeping the coresets gets to large, we can reduce it again by computing a coreset of the union of several stored coresets (also see Observation 4.1.4 below).

As each reduction step means an additional error, we have to do these computations in a reasonable way, which is what the *Merge-and-Reduce* technique is about. Roughly speaking, a coreset computation that computes strong coresets of size $s = \varepsilon^{-\ell} \cdot S(k, n, d)$ for a function that depends on n, k and d but not on ε can be turned into a streaming algorithm which needs storage $\mathcal{O}(s \cdot \log^{\ell+1} n)$.

Notice that for weak coresets, this would require much more care because a weak coreset only guarantees the approximation for certain center sets. When merging sets, different subsets satisfy guarantees for different center sets, and we do not automatically get a situation where repeated merging and reducing yields overall approximation guarantees.

Streaming algorithms are the topic of Chapter 5 where we also discuss the Merge-and-Reduce technique in more detail. We keep in mind that coreset computations directly imply streaming algorithms, and that the strong coreset property turns out to be very convenient here, too.

Distributed algorithms. Another quite immediate consequence of the existence of a strong coreset computation for a problem is that the problem can be solved in a distributed setting. The reason is that coresets can be computed for subsets of a point set and then be merged into a coreset for the whole point set. So if the data points are spread over different computers, we can first compute a coreset and then send the coresets to a central computer or around in the network, gathering a coreset for the distributed data set. Similarly, if we want to do a parallel computation, we can first split a point set, compute coresets for the subsets and then join the coresets. The following observation states the necessary property for the unions of coresets. Notice that we define the weight functions on $\mathbb{R}^d$ to shorten the formulation. We assume they are zero everywhere except for the points in their respective point set.

Observation 4.1.4. *Let $\varepsilon \in (0,1)$. Let $S_1, S_2 \in \mathbb{R}^d$ with weight functions $w_1, w_2 : \mathbb{R}^d \to \mathbb{R}$ be $(1+\varepsilon)$-coresets for two disjoint point sets $P_1, P_2 \subset \mathbb{R}^d$ of the same dimension. Then $S_1 \cup S_2$ with $w_1 + w_2 : \mathbb{R}^d \to \mathbb{R}$ is a $(1+\varepsilon)$-coreset for $P_1 \cup P_2$.*

Proof. Let C be a set of k centers from $\mathbb{R}^d$. As S_1 and S_2 are coresets for P_1 and P_2, respectively, it holds that $|\text{cost}_{\ell_2^2,w_1}(S_1,C) - \text{cost}_{\ell_2^2}(P_1,C)| \leq \varepsilon \text{cost}^*_{\ell_2^2}(P_1)$ and $|\text{cost}_{\ell_2^2,w_2}(S_2,C) - \text{cost}_{\ell_2^2}(P_2,C)| \leq \varepsilon \text{cost}^*_{\ell_2^2}(P_2)$. Thus, we also have that

$$\begin{aligned}
&|\text{cost}_{\ell_2^2,w_1+w_2}(S_1 \cup S_2, C) - \text{cost}_{\ell_2^2}(P_1 \cup P_2, C)| \\
&= |\text{cost}_{\ell_2^2,w_1}(S_1,C) + \text{cost}_{\ell_2^2,w_2}(S_2,C) - \text{cost}_{\ell_2^2}(P_1,C) - \text{cost}_{\ell_2^2}(P_2,C)| \\
&\leq \varepsilon \cdot \text{cost}^*_{\ell_2^2}(P_1) + \varepsilon \cdot \text{cost}^*_{\ell_2^2}(P_2) = \varepsilon \cdot \text{cost}^*_{\ell_2^2}(P)
\end{aligned}$$

by the triangle inequality and because the cost of a point set can be split into the sum of the costs of subsets. □

Notice that computing $(1+\varepsilon')$-coresets of size s for ℓ subsets and merging them results in a $(1+\varepsilon')$-coreset of size $\ell \cdot s$. The size can be reduced by another coreset computation, which increases the error to $(1+\varepsilon')^2$. Using an error parameter of $\varepsilon' = \varepsilon/3$ gives a $(1+\varepsilon')^2 = 1 + 2\varepsilon/3 + \varepsilon^2/9 < (1+\varepsilon)$-coreset of size s.

4.2 Techniques used in coreset constructions

In this section, we shortly look at two main techniques used in the construction of coresets. There are two reasons for doing this. First, we need both techniques later on (sampling is used in Chapter 7). Second, it is convenient for the review of different results in Section 4.2.3. Our focus is not on obtaining small coresets, but on playing with the techniques to get an idea how they work.

Notice that we talk about unweighted input point sets for the sake of a clearer exposition. Most coreset constructions can be naturally extended to work for inputs with integer weights. If this is not the case, we now see how to compute a coreset of a weighted point

set based on a coreset construction for unweighted point sets. However, if an explicitly worked out extension to weighted inputs is known, it should be prefered.

The following lemma uses a similar technique as used by Chen [Che09] who attributes it to Mettu and Plaxton [MP04].

Lemma 4.2.1. *Let P be a point set with n points and let $w : P \to \mathbb{N}$ be integer weights and total weight W. Computing a coreset with integer weighted points for the k-means problem on P with w can be reduced to $\mathcal{O}(\varepsilon^{-1} \log W)$ unweighted coreset computations on point sets with at most $\mathcal{O}(n)$ points.*

Proof. We compute the coreset in three steps. First, we partition the input into groups of points with similar weight. Second, we round the weights such that they are the same integer for all points in the same group. Third, we compute a $(1+\varepsilon)$-coreset for each group. The coreset points are then weighted by the integer corresponding to the group.

In order to prevent the rounding in the second step from introducing to much error, we spend one group for each integer weight from 1 to $\lceil 2/\varepsilon \rceil$. These are $\mathcal{O}(\varepsilon^{-1})$ groups. We set $\tilde{w}(x) := w(x)$ for all points in these groups.

To partition the remaining points, we assign an ℓ to each point $x \in P$ which is defined as the largest integer (including zero) that satisfies $(1+\varepsilon/2)^\ell \leq w(x)$. All points with the same exponent ℓ form a group, and the number of these groups is bounded by $\mathcal{O}(\log_{1+\varepsilon/2} W) = \mathcal{O}(\varepsilon^{-1} \log W)$. If a point x is in the group for exponent ℓ, then we set $\tilde{w}(x) := \lfloor (1+\varepsilon/2)^\ell \rfloor$. By definition, it holds $\tilde{w}(x) \leq w(x)$. Additionally, it holds that

$$\begin{aligned}\tilde{w}(x) = \lfloor (1+\varepsilon/2)^\ell \rfloor \geq (1+\varepsilon/2)^\ell - 1 \geq w(x)/(1+\varepsilon/2) - 1 \geq (1-\varepsilon/2)w(x) - 1 \\ \geq (1-\varepsilon)w(x)\end{aligned}$$

where the third inequality holds since $0 \leq \varepsilon \leq 1$ and the last inequality holds since $w(x) \geq 2/\varepsilon$, which means that $1 \leq (\varepsilon/2)w(x)$.

If all points in a set have the same weight, then computing a coreset for it is equivalent to computing an unweighted coreset. Thus, we compute a coreset for each of the $\mathcal{O}(\varepsilon^{-1}) + \mathcal{O}(\varepsilon^{-1} \log W)$ groups and multiply the weight of each point x in it by $\tilde{w}(x)$. The union of the resulting weighted sets is the coreset. □

4.2.1 Moving points

The idea to move points without changing the k-means cost function too much is so central that it appears at various places in this thesis. We proved the necessary Lemma 2.3.2 in Section 2.3, and first used it in Section 3.2 for obtaining the SVD-based dimensionality reduction. It also plays an important role in this chapter for constructing coresets for the k-means problem, and in the streaming algorithm in Section 5.4. A generalized version is used in Chapter 6.

Lemma 2.3.2 says the following about the error induced by moving points of a point set $P \in \mathbb{R}^d$ to obtain a point set $Q \in \mathbb{R}^d$. Denote the movement distance of a point $x \in P$ by $d(x)$. If the sum of $d(x)^2$ for all $x \in P$ is bounded by $(\varepsilon^2/16) \cdot \text{cost}^*_{\ell_2^2}(P)$, then the k-means

cost of the moved point set, $\text{cost}_{\ell_2^2}(Q, C)$, is a $(1+\varepsilon)$-approximation of $\text{cost}_{\ell_2^2}(P, C)$ for every set of centers C.

For a coreset construction, the idea is to move multiple points to the same location without distorting the cost function too much, and then to replace the set of points on the same location by one weighted coreset point. As an example, we consider a way to use grids that cover P. The point set is contained in k boxes of width $2\sqrt{\text{cost}^*_{\ell_2^2}(P)}$: Any optimal solution induces a partitioning into k subsets, and in each subset, the distance between any point and the center can be at most $\sqrt{\text{cost}^*_{\ell_2^2}(P)}$. Each box is now discretized by a grid where the cells have width $\varepsilon \cdot (4\sqrt{nd})^{-1} \cdot \sqrt{\text{cost}^*_{\ell_2^2}(P)}$. Every point is moved to an arbitrary corner of the cell it falls into, points that are already in a corner stay there. The movement of a point x is then bounded by $d(x) = \sqrt{d} \cdot \varepsilon \cdot (4\sqrt{nd})^{-1} \cdot \sqrt{\text{cost}^*_{\ell_2^2}(P)}$, so we have that

$$\sum_{x \in P} d(x)^2 \leq \sum_{x \in P} \varepsilon^2 \cdot (16 \cdot n)^{-1} \cdot \text{cost}^*_{\ell_2^2}(P) = \frac{\varepsilon^2}{16} \cdot \text{cost}^*_{\ell_2^2}(P).$$

We can thus apply Lemma 2.3.2. Let S contain all corners of the grid that are occupied by at least one moved point, and define the weight of a point $y \in S$ as the number of points $x \in P$ that are moved to y. Then Lemma 2.3.2 guarantees that $|\text{cost}_{\ell_2^2,w}(S, C) - \text{cost}_{\ell_2^2}(P, C)| \leq \varepsilon \cdot \text{cost}_{\ell_2^2}(P, C)$ for all center sets C with $|C| = k$. In other words, S is a coreset for P.

However, the size of S is unconvincing. The grid has $\mathcal{O}(k \cdot (4\sqrt{n}\varepsilon^{-1})^d)$ corners, so the size of S is only guaranteed to be sublinear for $d = 1$ (and n large enough compared to $4/\varepsilon$, but this is a common assumption), and it is nowhere near the polylogarithmic dependence on n that we desire[1]. We move the points around too blindly, not using any structure of the points. In Section 4.2.3, we hear about different approaches to employ the movement lemma in a more clever way in order to obtain true coresets with at most polylogarithmic dependence on n. What they do have in common with the toy construction above, however, is the exponential dependence on the dimension. Purely geometric arguments do not suffice do get rid of this, but a sampling based technique helps. We review its basic idea in the next section.

4.2.2 Uniform sampling

The core of this idea to construct a coreset is simple: Choose a set of points uniformly at random. The difficulty lies in bounding the error of the so constructed set, and then in employing the sampling in a reasonable way to reduce this error.

Let C be a solution candidate, i. e., a set of k centers. When choosing one point x uniformly at random, its expected cost given the solution C is

$$\text{E}[\text{dist}^2(x, C)] = \sum_{y \in P} \frac{1}{|P|} \text{dist}^2(y, C) = \frac{1}{|P|} \text{dist}^2(P, C).$$

[1]Notice that S contains at most n points, so at least it is not larger than P.

Figure 4.2: An example for $k = 2$ where the two blue circles represent nearly $n/2$ points each, but some outliers in red dominate the optimal k-median cost.

Similarly, if we choose a set S of points independently and uniformly at random, the expected cost is $\mathrm{E}[\mathrm{cost}_{\ell_2^2}(S, C)] = \frac{|S|}{|P|}\mathrm{cost}_{\ell_2^2}(P, C)$. Thus, we get an unbiased estimator for the cost of P (with one center set C) by weighting all points in S by $|P|/|S|$.

This approach faces two difficulties. First, the number of different center sets can be large depending on the specific problem, and the estimation has to be good for all of them. Second, the estimation can have a high variance, such that with high probability, the sample set is not a coreset.

A usual way to attack the first problem is to discretize the space of possible centers in a clever way in order to bound the number of different center sets while not losing too much quality. We do not want to deal with this issue here and just look at a problem which is discrete in itself, the discrete k-median problem. Notice that the restriction that centers have to be chosen from the input point set implies that there are at most n^k different center sets. We choose k-median here to simplify the exposition and add a few comments on k-means below.

The second issue is equivalent to the question how many samples are needed to achieve that the cost of the sample set is close enough to the cost of the original points. Sufficient bounds on the sample size can only be found if the input has a convenient structure.

Consider Figure 4.2. Most of the points are concentrated on the positions of the optimal two centers, and in an optimal solution, these points cost nothing (or very little, if we pertubate them a bit to avoid points with exactly the same position). So in order to approximate the k-median cost of the optimal solution, a coreset needs to contain some of the red points. In order to sample even one red point with high probability during a uniform sampling, we need to sample a linear number of points. Thus, uniform sampling does not yield a sublinear coreset for this point set.

As we consider the discrete k-median problem, all distances to centers are pairwise distances between input points. Thus, the cost directly relates to the pairwise distances between input points. Imagine that all pairwise distances are the same, for example that all distances are one. Then the diameter of P is also one, and the cost of any fixed center set consisting of points from P is always $n - k$ (all points except the k center points pay one). A subset S of s points that are weighted by n/s costs between $(s - k) \cdot n/s$ and $s \cdot n/s = n$, depending on how many of the center points are contained in S. If we assume that n is large enough (for example, $n \geq 2k/\varepsilon$), then $n \leq (1 + \varepsilon)(n - k)$. If n is smaller, we can just store all points. We can ensure that $(s - k) \cdot n/s = n - (k \cdot n)/s$ is never too small by choosing an $s \geq k/\varepsilon$. Then, $n - (k \cdot n)/s \geq n - \varepsilon n = (1 - \varepsilon)n$. Thus, in this

simplified case, we can find a coreset with $\Theta(k/\varepsilon)$ points.

In general, we cannot achieve that pairwise distances are equal. We can, however, seek to achieve that the diameter of the point set is similar to the average cost, which is a relaxation of the above situation. This can be done by partitioning a point set into subsets where the diameter can be related to the average cost. In Section 4.2.3 below and in Chapter 7, we discuss how Chen [Che09] finds such a partitioning in more detail.

We conclude this section with demonstrating that relating the diameter to the average cost does indeed help in bounding the sample size needed when constructing coresets by uniform sampling. In the process, we review necessary concentration inequalities that are of independent interest to us. The following chain of ideas is similar to Chen's work [Che09]. Earlier work by Mishra, Oblinger and Pitt [MOP01] also contains a simpler form of the sampling strategy described here. We start with Hoeffding's inequality from probability theory.

Theorem 4.2.2 (Hoeffding's Bound, Theorem 2 in [Hoe63])**.** *Let $X_1 \dots, X_t$ be real-valued independent random variables that satisfy $a_i \leq X_i \leq b_i$ for $i = 1, \dots, t$ and let $\hat{X} := \sum_{i=1}^{t} X_i/t$. Then it holds for all $\varepsilon > 0$ that*

$$\text{Prob}\left(\hat{X} - \text{E}(\hat{X}) > \varepsilon\right) \leq e^{-2 \cdot t^2 \cdot \varepsilon^2 / \sum_{i=1}^{t} (b_i - a_i)^2}.$$

Notice that while Theorem 4.2.2 only bounds $\text{Prob}(\hat{X} - \text{E}(\hat{X}))$, we can still use it to bound the probability $\text{Prob}\left(\text{E}(\hat{X}) - \hat{X} > \varepsilon\right)$ by applying the theorem to the random variable $-X$ which is the sum of the random variables $-X_i$ for $i := 1, \dots, t$. By the union bound, this implies that

$$\text{Prob}\left(|\hat{X} - E(\hat{X})| > \varepsilon\right) \leq 2e^{-2 \cdot t^2 \cdot \varepsilon^2 / \sum_{i=1}^{t} (b_i - a_i)^2}.$$

Hoeffding's bound tells us how many samples one needs to bring a real-valued variable close to its expected value. The following theorem by Haussler gives us a convenient way to apply Hoeffding's bound. It was originally posted in the context of PAC learning.

Theorem 4.2.3 ([Hau90, Hau92])**.** *Let $M > 0$ be a fixed constant and T a finite set, and let $\mathcal{F}$ be a finite set of functions $f : T \to \mathbb{R}$ with $0 \leq f(x) \leq M$ for all functions $f \in \mathcal{F}$ and all $x \in T$. Let S be a finite set of samples drawn independently and uniformly at random from T, and let $\delta, \varepsilon > 0$ be parameters. If $|S| \geq \frac{M^2}{2\varepsilon^2} \cdot \left(\ln|\mathcal{F}| + \ln\left(\frac{2}{\delta}\right)\right)$, then it holds for an arbitrary but fixed $f \in \mathcal{F}$ that*

$$\text{Prob}\left(|E_T(f) - E_S(f)| \geq \varepsilon\right) \leq \delta/|\mathcal{F}|,$$

where $E_T(f) = \sum_{x \in T} f(x)/|T|$ and $E_S(f) = \sum_{y \in S} f(y)/|S|$.

Proof. Let $f \in \mathcal{F}$ be an arbitrary fixed function from $\mathcal{F}$. Let $x_1, \dots, x_s$ be the samples in S. Then, $f(x_i)$ is a real-valued random variable with $0 \leq f(x_i) \leq M$, and the expected value of $f(x_i)$ when choosing x_i uniformly at random from T is $E_T(f(x_i)) = \sum_{x \in T} f(x)/|T| =$

$E_T(f)$ for all $i = 1, \ldots, s$. Set $\hat{X} := \sum_{i=1}^{|S|} f(x_i)/|S| = E_S(f)$ and notice that $E(\hat{X}) = \sum_{i=1}^{|S|} E(f(x_i))/|S| = |S| \cdot E_T(f)/|S| = E_T(f)$ by the linearity of the expected value. By Hoeffding's Bound (Theorem 4.2.2) and the argument below Theorem 4.2.2 this implies that

$$\begin{aligned}\text{Prob}\left(|E_S(f) - E_T(f)| > \varepsilon\right) &< 2 \cdot e^{-2\cdot|S|^2\cdot\varepsilon^2/\sum_{i=1}^{|S|} M^2} \\ &= 2 \cdot e^{-2\cdot|S|\cdot\varepsilon^2/M^2} \leq 2 \cdot e^{-2\frac{M^2}{2\varepsilon^2}(\ln\mathcal{F}+\ln(2/\delta))\cdot\varepsilon^2/M^2} = \delta/|\mathcal{F}|.\end{aligned}$$

□

Let $T = P$ be an input point set for the discrete k-median problem. For every choice of k centers C, we define one cost function f_C which sets $f_C(x) = \text{dist}(x, C)$ for each point $x \in P$. Thus, the set $\mathcal{F}$ has cardinality n^k, and so it holds that $\ln \mathcal{F} = k \ln n$.

For any set of k centers $C \subset P$ and the corresponding $f_C \in \mathcal{F}$, Theorem 4.2.3 then yields that a random sample S of sufficient size satisfies with probability $1 - \delta/|\mathcal{F}|$ that

$$\begin{aligned} & |E_T(f_C) - E_S(f_C)| \leq \varepsilon \\ \Leftrightarrow \quad & \left|\frac{\sum_{x\in T} f_C(x)}{|T|} - \frac{\sum_{x\in S} f_C(x)}{|S|}\right| \leq \varepsilon \\ \Leftrightarrow \quad & \frac{1}{|T|} \cdot \left|\sum_{x\in T} \text{dist}^2(x, C) - \frac{|T|}{|S|} \cdot \sum_{x\in S} \text{dist}^2(x, C)\right| \leq \varepsilon \\ \Leftrightarrow \quad & \left|\sum_{x\in T} \text{dist}^2(x, C) - \frac{|P|}{|S|} \cdot \sum_{x\in S} \text{dist}^2(x, C)\right| \leq \varepsilon \cdot |P|. \end{aligned} \tag{4.1}$$

By applying the union bound, we get that Inequality (4.1) holds for all $f_C(x)$ simultaneously with probability $1 - \delta$. The inequality yields a coreset if the error is connected to the cost of P. Thus, we set $\varepsilon := \varepsilon' \cdot \text{cost}^*_{\ell_2}(P)/|P|$, implying that the error is at most $\varepsilon \cdot |P| = \varepsilon' \cdot \text{cost}^*_{\ell_2}(P)$.

The theorem requires $|S| \geq \frac{M^2}{2\varepsilon^2} \cdot (\ln|\mathcal{F}| + \ln(\frac{2}{\delta}))$. The factor $\ln|\mathcal{F}| + \ln(\frac{2}{\delta}) = k \ln n + \ln(2/\delta)$ is fine, but $M^2/(2\varepsilon^2)$ might be large.

This is where the diameter of P comes into play. We observe that $f_C(x)$ is bounded by the diameter of P for any C and for any $x \in P$. We thus set $M := \text{diam}(P)$, which yields a sample size of

$$\frac{M^2}{2\varepsilon^2} \cdot (\ln|\mathcal{F}| + \ln(\frac{2}{\delta})) = \frac{(\text{diam}(P))^2}{2(\varepsilon' \cdot \text{cost}^*_{\ell_2}(P)/|P|)^2} \cdot (k \ln n + \ln(2/\delta)).$$

As suggested above, assume that the diameter is related to the average optimal cost of P, e. g., $\text{diam}(P) \leq a \cdot \text{cost}^*_{\ell_2}(P)/|P| \leq a \cdot \text{cost}_{\ell_2}(P, C)$ for a constant $a \in \mathbb{R}$. Then by setting $\varepsilon' := \varepsilon''/a$, we get a $(1 + \varepsilon'')$-coreset for the discrete k-median problem of size $|S| = \mathcal{O}(\varepsilon''^{-2} \cdot (k \ln n + \ln \delta^{-1}))$.

Concluding remark on k-means. For the discrete k-means problem, $\text{diam}(P)$ is not an upper bound on the cost of a point, but $(\text{diam}(P))^2$ is, implying that we need a relationship between the square of the diameter and the average cost. When we go to the continuous version, the upper bound changes even more because centers can have arbitrary positions within the Euclidean space. Chen [Che09] describes how to apply Theorem 4.2.3 for various cases including the k-means problem. We consider his work in Section 4.2.3 and apply it to a probabilistic clustering problem in Chapter 7.

4.2.3 History of coresets in view of the k-means problem

The present definition of coresets and in particular the distinction between strong and weak coresets have not always been used during the development of coreset theory. Earlier papers use the term coreset for a smaller point set that somehow represents or summarizes the point set, or they do not even use the word coreset at all. We have a short look at different important steps during the progress of coreset using algorithms.

Besides k-means and k-median, we will also repeatedly touch results on other geometric problems. Given a set of points $P \subseteq \mathbb{R}^d$, *the k-center problem* asks for k centers such that the *maximum* distance of a point to its nearest center is minimized. For $k = 1$, the optimal solution to the 1-center problem is a *minimum enclosing ball*, i. e., a ball consisting of a center c and a radius r such that all points in P lie within distance r of c, where r is as small as possible among all choices of c and r.

The *k-cylinder problem* or *k-line center problem* asks for k lines (subspaces of dimension one) such that the maximum distance of any point to its nearest line is minimized. The k-center and the k-line center problem can be generalized to a projective clustering problem that asks for k subspaces of dimension j that minimize the maximal distance of all points to their closest subspace.

An *(α, β)-bicriteria approximation* for the k-means problem is a set of $\alpha \cdot k$ centers which gives a β-approximation to the optimal cost.

Coreset constructions for geometric problems. Since the idea of coresets is a very natural concept it is hard and probably impossible to determine where it was first used. A seminal paper is due to Agarwal, Har-Peled and Varadarajan [AHPV04], who published a joined journal publication of two preliminary conference publications [AHP01, HPV01]. Agarwal et al. work on extent measures of points. For a $u \in \mathbb{R}^d$, they define the measure $\bar{w}(u, P) := \max_{p \in P} \langle p, u \rangle - \min_{p \in P} \langle p, u \rangle$. Notice that for a unit vector[2] u, $\langle p, u \rangle$ is the projection of p onto the line spanned by u, so $\bar{w}(u, P)$ is the *directional width*, the largest distance between two points when projected onto a direction u.

Agarwal et al. use the term *ε-approximation* for a subset $Q \subseteq P$ which satisfies $(1 - \varepsilon)\bar{w}(u, P) \leq \bar{w}(u, Q)$ for all possible directions u. Thus, an ε-approximation can be seen

[2]Notice that Agarwal et al. use a different representation of directions suitable for their write-up, while we use unit vectors here.

as a coreset with the additional property that the point set is an unweighted subset of the input point set which therefore can only underestimate the true value.

To show that an ε-approximation exists, Agarwal et al. first reduce the problem to *fat* point sets by showing that affine transformations do not change the problem and by proving that for every point set P, there exists a affine transformation mapping P to a fat point set P'. They call a point set P' α-*fat* if there exists a point $p \in \mathbb{R}^d$ and a hypercube H such that $p + H \subseteq P' \subseteq p + \alpha H$, i. e., P lies in the 'shell' between the two cubes. The transformed point set is then discretized by a grid, and points from carefully chosen cells form the ε-approximation. The size of the ε-approximation is then further reduced by combining it with a technique by Bronshteyn and Ivanov [BI75], yielding an ε-approximation of size $\mathcal{O}(\varepsilon^{-(d-1)/2})$. Agarwal et al. use their ε-approximation for approximation algorithms for computing various extent measures like the diameter or the size of the smallest enclosing ball of a point set. They also propose kinetic and streaming variants of their algorithms.

Coresets in clustering. Bădoiu, Har-Peled and Indyk [BHPI02] find a coreset $S \subset P$ for the 1-center problem with an iterative process. The process starts with $S = \emptyset$ and first adds an arbitrary point of P to S. Then, in each round, it computes the center c and radius r of the minimum enclosing ball of the current subset S. If all points in P lie within distance $(1 + \varepsilon)r$ of c, the process stops. Otherwise, the point in P which is farthest away from c is added to S. When the process stops, S has approximately the same minimum enclosing ball as P (but this does not imply that it is a *strong* coreset because it might not approximate *all* enclosing balls). By a lower bound on the radius after the insertion of the first two points and the observation that each round increases the radius by a large enough amount, Bădoiu et al. show that the process stops after $\mathcal{O}(\varepsilon^{-2})$ rounds.

Bădoiu et al. use their coreset to develop an approximation algorithm for k-center and combine it with other techniques to obtain an approximation algorithm for the 1-cylinder problem. They also develop a summary for the k-median problem and use it for an approximation algorithm. The size of a coreset for 1-center of the above type was subsequently improved [BC03, BC08, KMY03a, KMY03b] to $\lceil \varepsilon^{-1} \rceil$ which is optimal [BC08].

The so far mentioned coreset constructions are deterministic, and they share the property that the coreset is a subset of the input point set and can only underestimate the objective value. A different line of development uses random sampling to obtain an input reduction.

Indyk [Ind99] uses sampling to speed up bicriteria approximation algorithms for the metric k-median problem. First, a sample S of $\tilde{O}(\sqrt{nk})$ points is drawn uniformly at random from the input set. Next, a bicriteria approximation is used on S to obtain $\mathcal{O}(k)$ centers. S is not shown to be a coreset, but the center set computed by the bicriteria approximation is then refined by adding additional centers to reduce the cost of the most expensive points under the computed bicriteria approximation. Together with these additional centers, the solution is an β-approximation with αk centers with constant probability for constants $\alpha, \beta > 0$ which are higher than for the invoked bicriteria approximation. The gain is that the algorithm runs in time $\tilde{\mathcal{O}}(n)$ if the invoked bicriteria approximation has a running time of $\tilde{\mathcal{O}}(n^2)$. Indyk also proposes similar algorithms for other geometric problems including

the Maximum Traveling Salesman and Maximum Spanning Tree Problem.

Mishra, Oblinger and Pitt [MOP01] connect clustering to developments in statistics [Hau92, Pol84]. Their work includes the observation that Haussler's Lemma that we stated as Theorem 4.2.3 is beneficial in the context of sampling for clustering problems. They perform a similar chain of ideas as described in Section 4.2.2 in the context of Haussler's Lemma. In particular, they define a set of $\mathcal{O}(n^k)$ functions corresponding to the possible solutions of a metric k-median problem and show that a uniform sample of size $\mathcal{O}(\frac{(\operatorname{diam}(P))^2}{\varepsilon^2}(k \ln n + \ln \delta^{-1}))$ yields a set with an absolute error in the clustering cost of ε. However, their work concentrates on drawing connections to statistics and developing PAC-learning inspired algorithms and it was before coresets in the present form were even defined. Thus, they do not extend their result to obtain a strong coreset.

Mishra et al. use their sample set for a constant factor approximation for the metric k-median problem and develop a similar algorithm for the Euclidean k-median problem.

Czumaj and Sohler [CS04] develop an improved analysis of the sampling proposed by Mishra et al. They also analyze the quality of solutions that are obtained by computing an α-approximation on the sample and obtained better approximation algorithms for several clustering problems, including the metric and Euclidean versions of the k-means problem and the k-median problem.

List of coreset results for the k-means problem. Table 4.1 lists coreset results for the k-means problem. It only contains strong coresets, i. e., point sets that yield a $(1+\varepsilon)$-approximation for *all* possible choices of k centers. In the following two paragraphs, we discuss the results in more detail. Notice that we mostly concentrate on the k-means results even though the papers often contain results on other problems, too. We also do not discuss the implications of the results for the data stream setting because this is the topic of Chapter 5. In particular we do not go into detail on the type of streaming algorithm that the coreset allows (which is a distinctive feature of a coreset) but concentrate on the techniques used and the sizes obtained by the constructions.

Coresets for k-means clustering in constant dimension. Har-Peled and Mazumdar define (strong) coresets for the k-median and the k-means problem in the way that we use the term today [HPM04]. They also propose coreset constructions for both problems. The basic underlying observation is that when input points are moved, it is possible to bound the change in the cost function if the movement distances are related to the optimum cost. We have seen this relation in Lemma 2.3.2.

In Section 4.2.1 we used a simple grid to partition the Euclidean space into cells such that all points in a cell can be replaced by one weighted point, or, in other words, all points can be moved to one location which then serves as a representative of the cell. However, the number of cells was much too high.

To relate the movement distances to the optimum cost while keeping the number of cells small, Har-Peled and Mazumdar base the partitioning on a (α, β)-bicriteria approximation. An exponential grid is placed around every of the αk centers. The exponential grids are

Authors	Year	Size	Reference
k and d are arbitrary			
Feldman, Langberg	2011	$\tilde{\mathcal{O}}(dk\varepsilon^{-6})$ / $\tilde{\mathcal{O}}(dk\varepsilon^{-4})$	[FL11b]
Langberg, Schulman	2010	$\tilde{\mathcal{O}}(d^2k^3\varepsilon^{-2})$	[LS10]
Feldman, Monemizadeh, Sohler	2007	$\text{poly}(k, \varepsilon^{-1})$, weak coreset	[FMS07]
Chen	2006	$\tilde{\mathcal{O}}(k\varepsilon^{-2}\log n(dk+\log\delta^{-1}))$	[Che09]
k is arbitrary, d is a constant			
Ackermann, Lammersen, Märtens, Raupach, Sohler, Swierkot	2010	$\tilde{\mathcal{O}}((\varepsilon^{-1}\delta^{-1})^{\mathcal{O}(d)} \cdot k\log n)$	[AMR+12]
Har-Peled, Kushal	2005	$\mathcal{O}(k^3\varepsilon^{-(d+1)})$	[HPK07]
Frahling, Sohler	2005	$\mathcal{O}(k\log n\varepsilon^{-d})$	[FS05]
Har-Peled, Mazumdar	2004	$\mathcal{O}(k\log n\varepsilon^{-d})$	[HPM04]

Table 4.1: A list of several coreset results for the k-means problem.

based on exponentially growing boxes, and the idea is to partition the 'ring' between one box and the next into a constant number of cells. Thus, the cells are also exponentially growing. This is okay because the larger boxes are further away from any center and thus there is a higher lower bound on the contribution to the optimum cost that these points have.

Har-Peled and Mazumdar use a bicriteria approximation with $\mathcal{O}(k\log^3 n)$ centers, construct $\mathcal{O}(\log n)$ rings around every center, each partitioned into $\mathcal{O}(\varepsilon^{-d})$ cells (for constant d). Thus, the coreset contains $\mathcal{O}(k\log^4 n \cdot \varepsilon^{-d})$ points. Notice that constants depending on d or α are omitted. We still see the exponential dependence on the dimension, which stems from the need to cover d-dimensional cubes with cubes of smaller width.

Frahling and Sohler [FS05] remove the need to compute a bicritera approximation. Their algorithm only uses the *cost* of an approximate solution, which can be found by starting with an upper bound and halving it until the coreset size meets the proven worst-case size.

Their idea is to construct a partitioning in such a way that all cells contribute a similar amount to the overall sum of all (squared) movement distances and thus to the error. Therefore they have to control the *number* of points in each cell. If a cell is large, fewer points are allowed to achieve the same worst-case bound on the (squared) movement distances. Thus, the cells are constructed by using nested grids, quite similar to a (multidimensional version of a) quadtree. The starting point is a grid with cell diagonal $\text{cost}^*_{\ell_2^2}(P)$. Cells with comparatively few points are called *light*, cells with relatively many points are called *heavy*, and the heavy cells are split into 2^d cells with a smaller diagonal. The bound that distinguishes light and heavy cells depends on the level, i. e., on the cell size. The coreset then consists of weighted representatives of the leaf cells of this construction.

For the analysis, the points are distinguished into near and far points. Near points are so close to their closest center in a (fixed but arbitrary) solution that, for constant d, the

number of cells containing near points can be bounded by $\mathcal{O}(k \cdot \varepsilon^{-d})$ by a volume argument. Then, the (carefully chosen) definition of light cells bounds the overall error in these cells. Far points have a lower bound on the distance to their respective center, implying that the movement distance of every far point is small enough in comparison to its contribution to the optimum cost.

The size of the coreset can be bounded by a similar distinction of cells into near and far for heavy cells as for light cells. Frahling and Sohler show that for a suitably refined version of their algorithm, the coreset size is bounded by $\mathcal{O}(k \log n/\varepsilon^d)$.

Frahling and Sohler also implemented their coreset construction which is documented in [FS08] (conference version published in 2006).

Notice that the two coreset constructions described above depend exponentially on the dimension and logarithmically on the number of points. Har-Peled and Kushal [HPK07] discovered that the dependence on the number of points can be avoided. Their construction is again based on centers of an approximative solution, but they replace the exponential grid around the centers. Instead, they place a unit sphere around each center and compute a set of $\mathcal{O}(\varepsilon^{-(d-1)})$ points for each sphere. Such a set satisfies that every point on the corresponding unit sphere is close to one of the points in the set. Then, each center is connected to each of the points in the respective set, which results in $\mathcal{O}(k\varepsilon^{-(d-1)})$ rays. The points are then snapped onto the rays. Har-Peled and Kushal show that the resulting movement distances are small enough.

The advantage of the thus reduced point set is that the points on every line lie on a one-dimensional subspace, allowing a reduction to a 1-dimensional coreset computation. The only difference is that the coreset must guarantee its approximation factor not only for centers which are themselves on the line, but for all centers from $\mathbb{R}^d$. Har-Peled and Kushal give a construction that satisfies this and that contains $\mathcal{O}(k^2\varepsilon^{-2})$ points, implying a coreset of size $\mathcal{O}(k^3\varepsilon^{-(d+1)})$.

Coresets for k-means clustering in high dimension. The idea behind coresets is to dramatically reduce the size of an input instance in order to compress the data, speed up approximation algorithms or reduce communication in distributed settings. If the dimension of the input is high, it is intuitive to ask for a coreset where the number of points does not exponentially depend on the dimension. Notice that as long as we stay in the input space, there is a natural dependence of $\Omega(d)$ in the actual coreset *size* because the coreset points are d-dimensional. However, the aim of coreset constructions in high dimension is to reduce the additional dependency on d as much as possible.

The work by Chen [Che09] is the first to achieve this for the k-median and k-means problem (and metric versions of both). As stated above, Mishra, Oblinger and Pitt [MOP01] already observed the usefulness of Theorem 4.2.3 in the context of clustering. Chen combines the uniform sampling approach with the idea to use a bicriteria approximation.

Chen divides the input according to the bicriteria approximation, assigning every point to its closest center and placing exponentially growing spheres around the centers. All points contained in the same shell (the ring between one sphere and the next) form one

subset. Then, sufficiently many points are sampled from every such ring set to form the coreset. The inner most sphere has a radius proportional to the average optimal cost. In this way, the cost of every point in the sphere can be bounded above by the (a multiple) of the average optimal cost, and as we observed in Section 4.2.2, this limits the number of samples needed from this subset. The point sets in the shells have a lower and upper bound on the cost of each point which are within a constant factor of each other, and this cost is related to the diameter of the subset of points in the shell. Again, this limits the number of necessary samples. By applying Theorem 4.2.3 appropriately, Chen gives an approximation guarantee of the sample for a set of k centers.

So far, this guarantee only holds for one choice of centers. In order to make it work in general, the construction can be called with a smaller failure probabilty parameter, such that the union bound yields that the approximation guarantee holds for more choices of centers with constant probability. However, the Euclidean space offers an infinite number of different choices of centes. Chen solves this problem by appropriately discretizing the space, making sure that the discretized choice of centers do not change the cost too much, and that the number of such choices is sufficiently small. Chen proves a coreset size of $\tilde{\mathcal{O}}(k\varepsilon^{-2}(\log n) \cdot (dk + \log \delta^{-1}))$, where δ is the overall failure probabilty.

The next step is to remove *both* the exponential dependency on d and the logarithmic dependency on the number of points at the same time. A suitable sampling process is a good candidate for this as it does not have an inherit dependence on the dimension as the different grid based constructions. When sampling uniformly at random, the expected cost of a point $x \in P$ with a center set C is just the average cost of P with C, $\text{cost}_{\ell_2^2}(P, C)/|P|$, so repeated sampling could potentially lead to a coreset. However, sampling uniformly at random usually has a too high variance. In Chen's work, this problem is solved by partitioning the points beforehand to reduce the variance. A different approach is to use non-uniform sampling to increase the probability of 'important' points. However, if the probability of a point x is $p(x) \neq 1/|P|$, then $\text{cost}_{\ell_2^2}(x, C)$ is biased. The estimator can be made unbiased again by weighting the sampled point by $1/p(x)$.

Feldman, Monemizadeh and Sohler [FMS07] are the first to use non-uniform sampling in the context of coresets for k-means clustering. Their sampling probabilities are based on the distances of the input points to the centers of a bicriteria approximation. Additionally, for points that are very close to their respective centers, a uniform sample is added to the sample. The sample is not shown to be a strong coreset, but it is a building block to compute a weak coreset. The result does not appear in Table 4.1 which only lists strong coreset results. It is noticable though that the size of the weak coreset in [FMS07] is $\text{poly}(k, \varepsilon^{-1})$, so the number of points is not only polynomial in d but even independent of d, while it is also independent of n. Finding a strong coreset with this properties is the main topic of this chapter.

Langberg and Schulman [LS10] propose a sampling scheme that yields coresets of size $\tilde{\mathcal{O}}(d^2k^3\varepsilon^{-2})$. Notice that they work on the more general topic of shape fitting and that they use a quite different notation. Nevertheless, the results on *ε-approximators* proposed in [LS10] in particular apply to the k-means problem.

The work in [LS10] is based upon the *sensitivity* of points. Given a point set P,

the sensitivity $\mathfrak{S}(x)$ of a point x (in the context of k-means clustering) is defined as $\sup_{C\subset\mathbb{R}^d,|C|=k} \operatorname{dist}^2(x,C)/\sum_{y\in P}\operatorname{dist}^2(y,C)$. So, the sensitivity is the maximal fraction that x can contribute to the total cost for any set of centers, and as such it is a value between zero and one. For every point x and any center set C, the cost of x with C is bounded by $\mathfrak{S}(x)\cdot\operatorname{cost}_{\ell_2^2}(P,C)$. The *total sensitivity* $\mathfrak{S}(P)$ is just the sum of the sensitivities of all points.

Langberg and Schulman propose to sample according to the sensitivity, or, more precisely, according to pointwise upper bounds $s(x)\geq\mathfrak{S}(x)$ which sum up to $S(P)\geq\mathfrak{S}(P)$. The probability to sample a point x is then defined as $p(x):=s(x)/S(P)$. To make the estimation unbiased, the sampled points are weighted by $1/p(x)$. Langberg and Schulman show that the variance of drawing one point with this sampling probability is bounded by $(S(P)-1)(\operatorname{cost}_{\ell_2^2}(P,C))^2$. Thus, finding a good bound on the sensitivity reduces the variance of the estimator. Two things remain. First, Langberg and Schulman also have to get around the problem that there are infinitely many choices of centers. Second, they have to compute the bounds $s(x)$ in order to perform the sampling. The latter is done based on a bicriteria approximation, so this construction is among those that build upon bicriteria approximations.

The idea to sample according to the sensitivity was further developed by Feldman and Langberg [FL11a]. They develop a more general framework using connections to PAC Learning developing coresets for different problems, including the k-means problem. First, they present a general bicriteria approximation algorithm, then they state coreset constructions based on the bicriteria approximation. The k-means coreset construction uses the bicriteria approximation to obtain probabilities for the then following sample step. The new ingredient is that the centers of the bicriteria approximation are actually added to the sample and the union then is the coreset. The size of the coreset is bounded by $\tilde{\mathcal{O}}(dk\varepsilon^{-6})$, which follows from a more general theorem in the current version of the paper [FL11b]. A analysis tailored for the k-means problem leads to a smaller bound on the size. Balcan, Ehrlich and Liang [BEL13] state $\tilde{\mathcal{O}}(dk\varepsilon^{-4})$ as the size of the coreset.

Coresets for the Euclidean k-median problem. The corset constructions [Che09, HPM04, HPK07, FL11a, FS05, LS10] also work in the Euclidean k-median case and produce coresets of similar size as stated in Table 4.1. The main difference is that the result by Har-Peled and Kushal implies a smaller coreset in the k-median case. It is then of size $\mathcal{O}(k^2\varepsilon^{-d})$. Recall that Chen computes coresets for the k-means and k-median problem of size $\tilde{\mathcal{O}}(k\varepsilon^{-2}\log n(dk+\log\delta^{-1}))$. His construction will be particularly important in Chapter 7.

As for the k-means problem, the sampling approach analyzed by Mishra, Oblinger and Pitt [MOP01], also studied by Czumaj and Sohler [CS04], provides insights on how to apply an approximation algorithm for the metric or Euclidean k-median problem directly on a random sample from the input point set. We discuss it in Section 4.2.3.

4.3 Computing small coresets for k-means

This section and the next section are devoted to the construction of coresets whose size does not depend on the dimension of the input points or on the number of input points. We start with an algorithm with an intuitive idea that illustrates the strength of the concept of coresets with offsets very well. The algorithm can be used in the context of kernel k-means as we discuss in Chapter 6.2. In Section 4.4, we see another coreset construction whose output size in addition to being independent of d and n has a notably small polynomial dependence on k and ε^{-1}. It is also quite different from the first construction considering the used techniques. This illustrates that the concept of coresets with offsets is not tailor-made for one specific coreset construction.

We already refered to this Section's construction when defining coresets with offsets. Recall that the basic observation in Lemma 2.4.1 that we discussed in some detail in Section 2.4 allows to split the 1-means cost of a point set P with a given center c into the optimal 1-means cost and the distance between c and the centroid of P, weighted by $|P|$. The optimal 1-means cost is a constant which appears in every clustering of P. We can thus seperately store it and concentrate on how to compute the second part of the cost. For this, we only need the centroid, yielding an exact coreset for the 1-means problem which consists of one constant and one weighted point.

The challenge of this section is to generalize this idea to the k-means problem. Notice that for a given center set C, a point set P decomposes into k subsets of points which have the same closest center. To compute the cost for C, it would be sufficient to know the centroid and the 1-means cost of every subset. However, the partitioning depends on C, and we do not know C in advance. This poses the challenge that we have to overcome in order to compute a coreset with offset for arbitrary k.

The idea to do this is to subdivide P into subsets such that for every subset P', representing it by a 1-means coreset is enough to approximate its k-means cost as well. Then, we can replace all subsets by their centroids and keep track of the constants. We distinguish two types of suitable subsets. The first type are *k-unstructured* subsets by which we mean that the 1-means cost and the k-means cost are similar for each of these sets, more precisely, the 1-means cost is at most $(1 + \varepsilon')$ times the k-means cost for a suitable ε'. Then, using a 1-means coreset instead of a k-means coreset does not introduce too much error. If a set is not k-unstructured we say that it is *k-structured*. The second type are *cheap* subsets, which have a very small 1-means cost. In this case, even if the subset is k-structured, clustering with 1 or k means does not make a difference compared to the *overall* cost.

We partition P into a set of k-unstructured and cheap subsets recursively. Given a point set, we check if it is k-unstructured. If so, we are done and store the subset, and the recursion stops. If not, we partition it into subsets according to an optimal k-means solution and recursively subdivide each of the subsets into k-unstructured or cheap subsets. The recursion also stops when we reach recursion depth ν for a parameter ν which is chosen such that all subsets on level ν of the recursion are cheap. Finally, we replace all subsets by their centroids and store the sum of the squared distances of all points to the centroid

Algorithm 4.1: Computing a coreset with offset, first algorithm.

```
1  Function Partition(P, k, t, ν,ε′)
2      Compute an optimal center set C* = {μ_1, ..., μ_k} for P;
3      Let P_1, ..., P_k be the partitioning of P induced by C*;
4      Let μ be the centroid of P;
5      if t = ν or dist^2(P, μ) ≤ (1 + ε′) Σ_{i=1}^{k} dist^2(P_i, μ_i) then
6          M := M ∪ P;
7      else
8          for i=1,...,k do
9              M := M ∪ Partition(P_i,k,t + 1,ν,ε′);
10         end
11     end
12     return M;
13 Algorithm Coreset(P,k,ε)
14     Set ε′ := ε^2/50, S := ∅ and 𝔇 := 0;
15     M := Partition(P, k, 0, ⌈log_{1+ε′} 1/ε′⌉, ε′);
16     for every set P′ ∈ M do
17         S := S ∪ μ(P′);
18         w(μ(P′)) := |P′|;
19         𝔇 = 𝔇 + dist^2(P′, μ(P′));
20     end
21     return S, w and 𝔇;
```

of their respective subset.

Algorithm 4.1 shows the algorithm in pseudo code. We use $\mathcal{M}$ to denote the set of subsets in our partitioning. We generate the partitioning in the function `Partition`. Lines 2-4 compute a partitioning of the input set P based on an optimal solution, and the centroid of P. In Line 5 we check whether P is k-unstructured or cheap. If one of these is the case, we store P in $\mathcal{M}$, otherwise we make a recursive call for all k subsets in the partitioning of P. The main algorithm `Coreset` calls the partition method and replaces the subsets by their centroids, keeping track of the constants. Also, in Line 18 the centroid is weighted by the number of points that it represents.

Notice that for a given center set C, the error of the coreset S with offset $\mathfrak{D}$ computed by Algorithm 4.1 stems from points that are in the same subset in the partitioning $\mathcal{M}$, but belong to different subsets of the partitioning induced by C. The cost of the coreset is

$$\mathrm{cost}_{\ell_2^2,w}(S,C)+\mathfrak{D}=\sum_{P'\in\mathcal{M}}|P'|\min_{c\in C}\mathrm{dist}^2(\mu(P'),c)+\sum_{P'\in\mathcal{M}}\mathrm{dist}^2(P',\mu(P')).$$

We recall from Section 2.4 that Lemma 2.4.1 allows to split the cost of the point set with one center c into the weighted distance between the centroid and c plus the squared distances of the points to the centroid. Reversely applied, this means that $\mathrm{cost}_{\ell_2^2,w}(S,C)+\mathfrak{D}=$

$\sum_{P' \in \mathcal{M}}(\text{dist}^2(\mu(P'), c_{P'}) + \text{dist}^2(P', \mu(P'))) = \sum_{P' \in \mathcal{M}} \text{dist}^2(P', c_{P'})$ where $c_{P'}$ is the center closest to $\mu(P')$ in C. In the optimal solution, however, the points in P' might not be closest to the same center and this induces the error.

For the analysis, we need some additional notation. For $i = 1, \ldots, \nu$, we name the set of subsets that are added to $\mathcal{M}$ on the ith level of the recursion $\mathcal{M}^i$. In particular, $\mathcal{M}^0$ contains the partitioning of the original P into k subsets according to an optimal solution. All subsets in $\mathcal{M}^i$ for $i < \nu$ are k-unstructured. We denote the union of all $\mathcal{M}^i$ for $i < \nu$ by $\mathcal{M}^{<\nu}$. By $\mathcal{M}^\nu$ we denote the set of subsets that are added to $\mathcal{M}$ on level ν. These subsets are cheap subsets (which may or may not be k-unstructured in addition). We start with showing that the subsets in $\mathcal{M}^\nu$ are indeed cheap, also making precise what we mean by cheap. Notice that being cheap is actually a property of a *set* of subsets that ensures that the sum of all the 1-means costs of the subsets in the set is small.

Lemma 4.3.1. *Let $P \in \mathbb{R}^d$ be a point set. Let $\mathcal{M}$ be the partitioning computed by Algorithm 4.1 and let $\mathcal{M}^\nu$ be the sets added to $\mathcal{M}$ on level ν of the recursion. Then it holds that*

$$\sum_{P' \in \mathcal{M}^\nu} \text{dist}^2(P', \mu(P')) \leq \varepsilon' \cdot \text{cost}_{\ell_2^2}(P).$$

We say that the subsets in $\mathcal{M}^\nu$ are cheap because they satisfy this property.

Proof. In Line 15 of Algorithm 4.1, we set the maximum level to $\log_{1+\varepsilon'}(\varepsilon')^{-1}$. We are interested in the sum of the 1-means costs of all point sets in $\mathcal{M}^\nu$. Notice that these sets result from repeatedly splitting subsets where the k-means cost is significantly smaller than the 1-means cost. In other words, the sum of the 1-means costs of the k subsets is significantly smaller than the 1-means cost of the starting set. On level ν, the split has been performed recursively for ν times. We can assume that we never stopped the recursion early because otherwise the sum of the 1-means costs is only smaller. We start with an optimal partitioning of the original input P, so the sum of the 1-means costs of these subsets is just $\text{cost}_{\ell_2^2}(P)$. After i levels of splitting, the sum of the 1-means costs of the resulting subsets is bounded by $\text{cost}_{\ell_2^2}(P)/(1+\varepsilon')^i$. On level ν, the sum is bounded by

$$\frac{\text{cost}_{\ell_2^2}(P)}{(1+\varepsilon')^\nu} = \frac{\text{cost}_{\ell_2^2}(P)}{(1+\varepsilon')^{\log_{1+\varepsilon'}(\varepsilon')^{-1}}} = \frac{\text{cost}_{\ell_2^2}(P)}{(\varepsilon')^{-1}} = \varepsilon' \cdot \text{cost}_{\ell_2^2}(P).$$

□

Now we consider the k-unstructured subsets. We say that a subset is k-unstructured if it satisfies that

$$\text{dist}^2(P, \mu(P)) \leq (1+\varepsilon') \cdot \text{cost}_{\ell_2^2}(P).$$

All subsets in $\mathcal{M}^{<\nu}$ satisfy this condition as Algorithm 4.1 uses an optimal solution to compare the 1-means cost with the k-means cost. As it turns out, the basic observation from Section 2.4 helps us to deal with k-unstructured subsets. What we see is that if we cluster a k-unstructured subset P' with a center set C which partitions P' into subsets, then the centroids of these subsets are relatively close to the centroid of P' (actually, this

does not only hold for partitionings induced by a center set, but for any partitioning). This will greatly help to bound the error of *not* splitting P' into different subsets when clustering it with C.

Lemma 4.3.2. *Let $P \in \mathbb{R}^d$ be a point set. Let $\mathcal{M}$ be the partitioning computed by Algorithm 4.1 and let $\mathcal{M}^{<\nu}$ be the sets added to $\mathcal{M}$ before level ν of the recursion. Let $P' \in \mathcal{M}^{<\nu}$ be such a subset and consider an arbitrary partitioning $P'_1, \dots, P'_k$ of P' into k subsets. Then it holds that*

$$\sum_{i=1}^{k} |P'_i| \cdot \operatorname{dist}^2(\mu(P'_i), \mu(P')) \leq \varepsilon' \cdot \sum_{i=1}^{k} \operatorname{dist}^2(P'_i, \mu(P'_i)).$$

For any fixed center set $C \subset \mathbb{R}^d$ with $|C| = k$, it holds that

$$\sum_{i=1}^{k} |P'_i| \cdot \operatorname{dist}^2(\mu(P'_i), \mu(P')) \leq \varepsilon' \cdot \operatorname{cost}_{\ell_2^2}(P', C).$$

Proof. The lemma is a direct consequence of Lemma 2.4.1. Consider that we can rewrite $\operatorname{dist}(P', \mu(P'))$ in the following way, using the lemma.

$$\operatorname{dist}^2(P', \mu(P')) = \sum_{i=1}^{k} \operatorname{dist}^2(P'_i, \mu(P')) = \sum_{i=1}^{k} |P'_i| \cdot \operatorname{dist}^2(\mu(P'_i), \mu(P')) + \operatorname{dist}^2(P'_i, \mu(P'_i))$$

As P' is k-unstructured and as the cost of an optimal clustering is a lower bound for the cost induced by the partitioning $P'_1, \dots, P'_k$, this implies that

$$\begin{aligned}\sum_{i=1}^{k} |P'_i| \cdot \operatorname{dist}^2(\mu(P'_i), \mu(P')) &= \operatorname{dist}^2(P', \mu(P')) - \sum_{i=1}^{k} \operatorname{dist}^2(P'_i, \mu(P'_i)) \\ \leq (1+\varepsilon') \cdot \sum_{i=1}^{k} \operatorname{dist}^2(P'_i, \mu(P'_i)) &- \sum_{i=1}^{k} \operatorname{dist}^2(P'_i, \mu(P'_i)) = \varepsilon' \cdot \sum_{i=1}^{k} \operatorname{dist}^2(P'_i, \mu(P'_i)).\end{aligned}$$

For the second part of the lemma, consider the situation that $P'_1, \dots, P'_k$ is the partitioning induced by C. The lemma holds for all partitionings, so in particular for this choice. Then, $\sum_{i=1}^{k} \operatorname{dist}^2(P'_i, \mu(P'_i)) \leq \operatorname{cost}_{\ell_2^2}(P, C)$ because the definition of P'_i now implies that all points in P'_i are clustered with the same center, and then Lemma 2.4.1 implies that the cost contains the sum of the squared distances to the $\mu(P'_i)$. □

We recall that one popular technique in coreset constructions is to move points by small amounts and to use Lemma 2.3.2 to bound the error in the cost function. We have seen that cheap subsets satisfy that the sum of the squared distances to the centroids is bounded. Thus, we can regard the replacement of the cheap subsets by their centroids as moving the points, and Lemma 2.3.2 helps us to bound the change in the cost function.

For the k-unstructured sets this approach does not work. A better way to look at the error is realizing that the clustering costs of the subsets with its closest center are computed

exactly, but the error stems from the fact that not all points in the subset would actually choose this center on their own, too.

We can still use arguments based on Lemma 2.3.2 by the following trick. We hypothetically shift the points in P such that the shifted versions of all points in the same subset will actually choose the same center. For this, we need to base the analysis on an arbitrary fixed center set C for Observation 4.3.3, Lemma 4.3.4 and Lemma 4.3.5. To define the movements, we introduce an intermediate point set Q that has the same clustering behaviour as P with regard to C.

Observation 4.3.3. *Let the point set Q contain the union of all $P' \in \mathcal{M}^\nu$. For every point set $P' \in \mathcal{M}^{<\nu}$, consider the partitioning $P'_1, \dots, P'_k$ induced by C and add $|P'_i|$ copies of $\mu(P'_i)$ to Q for every $i = 1, \dots, k$. Then, $\mathrm{cost}_{\ell_2^2}(P, C) = \mathrm{cost}_{\ell_2^2}(Q, C) + \mathfrak{D}_Q$ for $\mathfrak{D}_Q := \sum_{P' \in \mathcal{M}^\nu} \sum_{i=1}^k \mathrm{dist}^2(P_i, \mu(P_i))$ by Lemma 2.3.2.*

The effect of using Q instead of P is that all points originating from the same $P' \in \mathcal{M}^{<\nu}$ are now indistinguishable in Q and have the same clustering behaviour. Instead of moving the points in P, we define a mapping on Q. For all $P' \in \mathcal{M}^\nu$ and all $x \in P'$, we define $\pi(x) = \mu(P')$, so the points in cheap subsets are simply moved to the centroid of their respective set. For all $P' \in \mathcal{M}^{<\nu}$ we defined the partitioning $P'_1, \dots, P'_k$ induced by C, and said that Q contains $|P'_i|$ copies of $\mu(P'_i)$. All of these are mapped to $\mu(P')$. This corresponds to moving the k subsets of P' in such a way that their centroids coincide with the centroid of P'. We name the resulting point set Q'.

Lemma 4.3.4. *It holds that $|\mathrm{cost}_{\ell_2^2}(Q', C) + \mathfrak{D}_Q - \mathrm{cost}_{\ell_2^2}(P, C)| \leq (4/5)\varepsilon \cdot \mathrm{cost}_{\ell_2^2}(P)$.*

Proof. We check that the precondition of Lemma 2.3.2 is satisfied. According to Lemma 4.3.1, it holds that

$$\sum_{P' \in \mathcal{M}^\nu} \mathrm{dist}^2(P', \mu(P')) \leq \varepsilon' \cdot \mathrm{cost}_{\ell_2^2}(P).$$

For $P' \in \mathcal{M}^{<\nu}$, Lemma 4.3.2 implies that

$$\sum_{i=1}^k |P'_i| \,\mathrm{dist}^2(\mu(P'_i), \pi(\mu(P'_i))) = \sum_{i=1}^k |P'_i| \,\mathrm{dist}^2(\mu(P'_i), \mu(P')) \leq \varepsilon' \cdot \sum_{i=1}^k \mathrm{dist}^2(P'_i, \mu(P'_i)).$$

By the definition of the P'_i and Lemma 2.4.1, we have $\sum_{i=1}^k \mathrm{dist}^2(P'_i, \mu(P'_i)) \leq \mathrm{cost}_{\ell_2^2}(P', C)$. Together, this implies that

$$\sum_{x \in Q} ||x - \pi(x)||^2 \leq 2\varepsilon' \cdot \mathrm{cost}_{\ell_2^2}(P, C) = \frac{\varepsilon^2}{25} \mathrm{cost}_{\ell_2^2}(P, C) = \frac{[(4/5)\varepsilon]^2}{16} \mathrm{cost}_{\ell_2^2}(P, C).$$

Lemma 2.3.2 implies that $|\mathrm{cost}_{\ell_2^2}(Q, C) - \mathrm{cost}_{\ell_2^2}(Q', C)| \leq \varepsilon \cdot \max\{\mathrm{cost}_{\ell_2^2}(P, C), \mathrm{cost}_{\ell_2^2}(Q, C)\} = \varepsilon \cdot \mathrm{cost}_{\ell_2^2}(P, C)$. By Observation 4.3.3, we conclude that

$$|\mathrm{cost}_{\ell_2^2}(Q', C) + \mathfrak{D}_Q - \mathrm{cost}_{\ell_2^2}(P, C)| = |\mathrm{cost}_{\ell_2^2}(Q', C) - \mathrm{cost}_{\ell_2^2}(Q, C)| \leq (4/5)\varepsilon \cdot \mathrm{cost}_{\ell_2^2}(P, C).$$

□

Lemma 4.3.5. *It holds that* $|\mathrm{cost}_{\ell_2^2}(Q', C) + \mathfrak{D}_Q - \mathrm{cost}_{\ell_2^2,w}(S, C) - \mathfrak{D}| \leq 2\varepsilon' \mathrm{cost}_{\ell_2^2}(P, C)$.

Proof. We want to compare the values $\mathrm{cost}_{\ell_2^2}(Q', C)+\mathfrak{D}_Q$ and $\mathrm{cost}_{\ell_2^2,w}(S, C)+\mathfrak{D}$. Again, we distinguish between the contributions originating from cheap subsets and thus originating from k-unstructured subsets. For this, we use some additional notation. We split S into $S^{<\nu}$ and S^{ν} depending on whether the points are centroids of sets in $\mathcal{M}^{<\nu}$ or $\mathcal{M}^{\nu}$, and we also split the constant $\mathfrak{D}$ into $\mathfrak{D}^{<\nu}$ and $\mathfrak{D}^{\nu}$ accordingly. Similarly, Q' is split into $Q'^{<\nu}$ and Q'^{ν}. We do not need to split $\mathfrak{D}_Q$ as it only contains contributions from sets in $\mathcal{M}^{<\nu}$.

We start with the cheap subsets. Both Q'^{ν} and S^{ν} consist of centroids of all sets in $P' \in \mathcal{M}^{\nu}$. In S^{ν}, the centroid of P' is weighted by $|P'|$, while Q' contains $|P'|$ copies, so $\mathrm{cost}_{\ell_2^2}(Q', C) = \mathrm{cost}_{\ell_2^2,w}(S, C)$. As $\mathfrak{D}_Q$ does not contain contributions from sets in $\mathcal{M}^{\nu}$, $\mathfrak{D}^{\nu}$ completely contributes to the error. This is no problem though because $\mathfrak{D}^{\nu}$ consists of the squared distances of the points in all $P' \in \mathcal{M}^{\nu}$ to the respective centroids and the sum of these squared distances is cheap by Lemma 4.3.1. We get that

$$|\mathrm{cost}_{\ell_2^2}(Q'^{\nu}, C) - \mathrm{cost}_{\ell_2^2,w}(S, C) - \mathfrak{D}^{\nu}| = \sum_{P' \in \mathcal{M}^{\nu}} \mathrm{dist}^2(P', \mu(P')) \leq \varepsilon' \mathrm{cost}_{\ell_2^2}(P, C).$$

We continue with the k-unstructured subsets. Again Q' and S contain similar points: For every $P' \in \mathcal{M}^{<\nu}$, Q' contains $|P'|$ points that were all shifted to $\mu(P')$, while S contains $\mu(P')$, weighted by $|P'|$. The difference lies within the constants. S arises from summarizing a set P' entirely, replacing it by $\mu(P')$ plus the constant $\mathrm{dist}^2(P', \mu(P'))$. For the same P', the points in Q' originate from k subsets $P'_1, \ldots, P'_k$ which are the partitioning of P' according to C. Each subset generated one point $\mu(P'_i)$ in Q which was then shifted to $\mu(P')$ in Q'. So the points coincide, but the constant $\mathfrak{D}_Q$ contains the sum of the squared distances of the points in P'_i to the centroid $\mu(P'_i)$. However, for k-unstructured subsets, the centroids of the P'_i and of P are close, so this does not induce much error. We see that

$$\begin{aligned}
&|\mathrm{cost}_{\ell_2^2}(Q'^{<\nu}, C) + \mathfrak{D}_Q - \mathrm{cost}_{\ell_2^2,w}(S, C) - \mathfrak{D}^{<\nu}| \\
&= |\mathfrak{D}_Q - \mathfrak{D}^{<\nu}| \\
&= \sum_{P' \in \mathcal{M}^{<\nu}} \Big(\mathrm{dist}^2(P', \mu(P')) - \sum_{i=1}^{k} \mathrm{dist}^2(P'_i, \mu(P'_i)) \Big) \\
&= \sum_{P' \in \mathcal{M}^{<\nu}} \Big(\sum_{i=1}^{k} \Big(\mathrm{dist}^2(P'_i, \mu(P'_i)) + |P'_i| \, \mathrm{dist}^2(\mu(P'), \mu(P'_i)) \Big) - \sum_{i=1}^{k} \mathrm{dist}^2(P'_i, \mu(P'_i)) \Big) \\
&= \sum_{P' \in \mathcal{M}^{<\nu}} \sum_{i=1}^{k} |P'_i| \, \mathrm{dist}^2(\mu(P'), \mu(P'_i)) \leq \varepsilon' \mathrm{cost}_{\ell_2^2}(P, C).
\end{aligned}$$

For the fourth equality, we use Lemma 2.4.1, and the inequality follows by Lemma 4.3.2. We conclude by adding the two errors.

$$\begin{aligned}
&|\mathrm{cost}_{\ell_2^2,w}(S, C) + \mathfrak{D} - \mathrm{cost}_{\ell_2^2}(Q', C) - \mathfrak{D}_Q| \leq |\mathrm{cost}_{\ell_2^2,w}(S^{\nu}, C) + \mathfrak{D}^{\nu} - \mathrm{cost}_{\ell_2^2}(Q'^{\nu}, C)| \\
&+ |\mathrm{cost}_{\ell_2^2,w}(S^{<\nu}, C) + \mathfrak{D}^{<\nu} - \mathrm{cost}_{\ell_2^2}(Q'^{<\nu}, C) - \mathfrak{D}_Q| \leq 2\varepsilon' \mathrm{cost}_{\ell_2^2}(P, C).
\end{aligned}$$

□

Theorem 4.3.6. *Let $P \subset \mathbb{R}^d$ be a point set, let $0 \leq \varepsilon \leq 1$. Algorithm 4.1 computes a $(1+\varepsilon)$-coreset S with offset $\mathfrak{D}$ for the k-means problem on P. In particular, it holds for every set of k centers C from $\mathbb{R}^d$ that*

$$|\mathrm{cost}_{\ell_2^2}(P,C) - \mathrm{cost}_{\ell_2^2,w}(S,C) - \mathfrak{D}| \leq \varepsilon \cdot \mathrm{cost}_{\ell_2^2}(P,C).$$

S contains $k^{\mathcal{O}(\varepsilon^{-2}\ln\varepsilon^{-1})}$ points.

Proof. Lemma 4.3.4 and Lemma 4.3.5 hold for an arbitrary fixed C. Thus, defining Q and Q' based on C and applying these statements yields that

$$\begin{aligned}
&|\mathrm{cost}_{\ell_2^2}(P,C) - \mathrm{cost}_{\ell_2^2,w}(S,C) - \mathfrak{D}| \\
\leq&|\mathrm{cost}_{\ell_2^2}(P,C) - \mathrm{cost}_{\ell_2^2}(Q',C) - \mathfrak{D}_Q| + |\mathrm{cost}_{\ell_2^2}(Q',C) + \mathfrak{D}_Q - \mathrm{cost}_{\ell_2^2,w}(S,C) - \mathfrak{D}| \\
\leq&((4/5)\varepsilon + 2\varepsilon') \cdot \mathrm{cost}_{\ell_2^2}(P,C) < \varepsilon \cdot \mathrm{cost}_{\ell_2^2}(P,C).
\end{aligned}$$

The size of the coreset is bounded because of the maximal recursion depth $\nu = \log_{1+\varepsilon'} 1/\varepsilon'$. In the worst-case, all sets in $\mathcal{M}$ belong to $\mathcal{M}^\nu$, i. e., the algorithm finds no k-unstructured subsets but always partitions, gaining a set of cheap subsets on level ν. Then, there are k^ν subsets in $\mathcal{M}$, and $k^\nu = k^{\log_{1+\varepsilon'} 1/\varepsilon'}$.

Let $\ln x : \mathbb{R}^+ \to \mathbb{R}$ be the natural logarithm, i. e., the inverse function of e^x where $e = \sum_{i=1}^{\infty} \frac{1}{1\cdot 2\cdot \ldots \cdot i}$ is Euler's number. We use two known facts. First, $\log_{1+\varepsilon'} x = \frac{\ln x}{\ln 1+\varepsilon'}$. Second, for all $x \in \mathbb{R}$ with $x < 1$, it holds that $e^x \leq 1/(1-x)$ and thus also $x \leq \ln(1/(1-x))$. With this in mind, we see that $\varepsilon < 1$ implies

$$\log_{1+\varepsilon'} 1/\varepsilon' = \ln \varepsilon'^{-1} \cdot \frac{1}{\ln(1+\varepsilon')} = \frac{\ln \varepsilon'^{-1}}{\ln\left(\frac{1}{1-\frac{\varepsilon'}{1+\varepsilon'}}\right)} \leq \frac{\ln \varepsilon'^{-1}}{\frac{\varepsilon'}{1+\varepsilon'}} \leq (1+\varepsilon'^{-1}) \cdot \ln \varepsilon'^{-1}.$$

As $\varepsilon' = (\varepsilon^2/50)$, we can bound the number of levels by $\mathcal{O}(\varepsilon^{-2}\log\varepsilon^{-1})$. □

4.3.1 Improving the running time

So far, we have not analyzed the running time of Algorithm 4.1. That is because the repeated computation of optimum k-means solutions rather serves the purpose of a clear presentation of the algorithm than it serves to obtain a reasonable running time. The running time is clearly exponential in $1/\varepsilon$ because of the number of coreset points that we compute. However, we can reduce the dependencies on other parameters by replacing the optimum k-means computations by approximations. Notice that the analysis above does not depend on the number of subsets in $\mathcal{M}$ or on the actual computation process, but only on the fact that the subsets are all either cheap or k-unstructured. We change the algorithm but ensure that this property still holds.

Algorithm 4.2 is a variant of Algorithm 4.1 that uses an α-approximate solution to decide whether to further split a set. $\mathcal{M}^{<\nu}$ still consists of k-unstructured subsets. Let $P_1^*, \ldots, P_k^*$ be an optimal partitioning with centers $\mu_1^*, \ldots, \mu_k^*$. Then the fact that we use

Algorithm 4.2: Computing a coreset with offset, variant of the first algorithm.

Function `Partition`$(P,k,t,\nu,\varepsilon',\alpha)$
  Compute a center set $C' = \{\mu_1, \ldots, \mu_k\}$ with $\text{cost}_{\ell_2^2}(P, C') \leq \alpha \cdot \text{cost}_{\ell_2^2}(P)$;
  Let $P_1, \ldots, P_k$ be the partitioning of P induced by C^*;
  Let μ be the centroid of P;
  if $\ell = \nu$ *or* $\text{dist}^2(P,\mu) \leq ((1+\varepsilon')/\alpha) \sum_{i=1}^{k} \text{dist}^2(P_i, \mu_i)$ **then**
    $\mathcal{M} := \mathcal{M} \cup \{P\}$;
  else
    for *i=1,...,k* **do**
      $\mathcal{M} := \mathcal{M} \cup$ `Partition`$(P_i,k,t+1)$;
    end
  end
  return $\mathcal{M}$;
Algorithm `Coreset`(P,k,ε)
  Set $\varepsilon' := \varepsilon^2/50$, $\alpha := 1 + \varepsilon'/2$, $S := \emptyset$ and $\mathfrak{D} := 0$;
  $\mathcal{M} :=$ `Partition`$(P, k, 0, \log_{(1+\varepsilon')/\alpha} 1/\varepsilon', \alpha)$;
  for *every set* $P' \in \mathcal{M}$ **do**
    $S := S \cup \mu(P')$;
    $w(\mu(P')) := |P'|$;
    $\mathfrak{D} = \mathfrak{D} + \text{dist}^2(P', \mu(P'))$;
  end
  return S, w and $\mathfrak{D}$;

an α-approximation guarantees us that all sets added to $\mathcal{M}^{<\nu}$ are k-unstructured because it holds that

$$\text{dist}^2(P,\mu) \leq ((1+\varepsilon')/\alpha) \sum_{i=1}^{k} \text{dist}^2(P_i,\mu_i) \leq (1+\varepsilon') \sum_{i=1}^{k} \text{dist}^2(P_i^*, \mu_i^*).$$

In order to ensure that the subsets in $\mathcal{M}^\nu$ are still cheap, we increase ν. By each level of the recursion, the cost decreases by a factor of $(1+\varepsilon')/\alpha > 1$ (because $\alpha = 1 + \varepsilon'/2 < 1 + \varepsilon'$). Thus, the total cost of the sets on level ν is bounded by

$$\sum_{P' \in \mathcal{M}^\nu} \text{dist}^2(P', \mu(P')) \leq \frac{\text{cost}_{\ell_2^2}(P)}{((1+\varepsilon')/\alpha)^\nu} = \frac{\text{cost}_{\ell_2^2}(P)}{((1+\varepsilon')/\alpha)^{\log_{(1+\varepsilon')/\alpha} \varepsilon'^{-1}}} = \varepsilon' \text{cost}_{\ell_2^2}(P).$$

By increasing ν, the size of the coreset increases, too. Similar as above, we bound the worst-case size by

$$\log_{(1+\varepsilon')/\alpha} 1/\varepsilon' = \ln \varepsilon'^{-1} \cdot \frac{1}{\ln(1+\varepsilon')/\alpha} = \frac{\ln \varepsilon'^{-1}}{\ln \frac{1}{1 - \frac{1+\varepsilon'-\alpha}{1+\varepsilon'}}} \leq \frac{\ln \varepsilon'^{-1}}{\frac{1+\varepsilon'-\alpha}{1+\varepsilon'}} = \frac{\ln \varepsilon'^{-1}}{\frac{\varepsilon'-\varepsilon'/2}{1+\varepsilon'}} = 2(1+\varepsilon'^{-1}) \cdot \ln \varepsilon'^{-1}.$$

For the running time, we use the result by Jaiswal, Kumar and Sen [JKS14] computing a $(1+\varepsilon')$-approximation of a point set with n points in time $\mathcal{O}(nd2^{\mathcal{O}(k^2/\varepsilon')})$ for constant k. As we need a $(1+\varepsilon'/2)$-approximation, the running time is at most $\mathcal{O}(nd2^{\mathcal{O}(k^2/\varepsilon^2)})$ for each approximation. We have to compute $k^{\mathcal{O}(\varepsilon^{-2}\ln\varepsilon^{-1})} = 2^{\mathcal{O}(\log k\varepsilon^{-2}\ln\varepsilon^{-1})}$ approximations of point sets with at most n points, and all other computations need a smaller running time. The resulting running time is highly exponential in k and ε, but not in the dimension d.

Theorem 4.3.7. *Let $P \subset \mathbb{R}^d$ be a point set. Algorithm 4.2 computes a $(1+\varepsilon)$-coreset S with offset $\mathfrak{D}$ for the k-means problem on P. In particular, it holds for every set of k centers C from $\mathbb{R}^d$ that*

$$|\mathrm{cost}_{\ell_2^2}(P,C) - \mathrm{cost}_{\ell_2^2,w}(S,C) - \mathfrak{D}| \leq \varepsilon \cdot \mathrm{cost}_{\ell_2^2}(P,C).$$

S contains $k^{\mathcal{O}(\varepsilon^{-2}\ln\varepsilon^{-1})}$ points. The algorithm can be implemented to have a running time of $\mathcal{O}(nd2^{\mathcal{O}(k^2\varepsilon^{-2}\ln\varepsilon^{-1})})$ for constant k.

4.4 Smaller coreset sizes via dimensionality reduction

Now we continue with a different coreset computation that yields smaller coresets with offset for k-means, that also contain a number of points which is independent of the dimension. The construction is more specialized to squared Euclidean distances than the previous one because it is based on dimensionality reduction via singular value decomposition.

In Section 3.2, we saw a way to reduce the dimensionality of a point set to $m = \mathcal{O}(k/\varepsilon^2)$ dimensions while distorting the k-means cost by at most a factor of $(1+\varepsilon)$. The low dimensional point set still lies within $\mathbb{R}^d$, but lies in a subspace of dimension m. More precisely, a point set stored in the rows of a matrix $A \in \mathbb{R}^{n\times d}$ is reduced to $A^{(m)}$, the projection of A onto the best fit subspace V_m of dimension m.

This dimensionality reduction can be used to speed up algorithms that have a running time depending on the dimension of the point set. In this section we see that it can also be used to compute small coresets. The idea is to combine the dimensionality reduction with existing coreset constructions. When the coreset constructions are run on the lower dimensional subspace, they produce coresets of size independent of the original dimension. Instead, the dependence on the dimension turns into a dependency on $\mathcal{O}(k/\varepsilon^2)$.

We have to take care of one detail, though. Recall from the related work section that Har-Peled and Kushal [HPK07] use a dimensionality reduction to $d = 1$ by projecting the point set to rays originating at the centers of a bicriteria approximation. Then they compute a coreset for the points on each ray. The important detail here was that the coresets have to provide a guarantee for centers from $\mathbb{R}^d$, not only for centers that lie on the ray themselves. We have a similar problem: If we consider $A^{(m)}$ as a m-dimensional subspace and blindly compute a coreset for it, the coreset guarantee will only hold for centers which lie in V_m.

Luckily, there is an easy solution for this problem. We fix an arbitrary $(m+k)$-dimensional subspace V' that contains V_m. Then we consider the points as $(m+k)$-dimensional and compute a coreset for them. Now for an arbitrary center set C, we argue

as follows. $A^{(m)}$ and the k centers together lie in a subspace of dimension $m+k$. The squared distances between the points in $A^{(m)}$ and the points in C stay the same if we rotate the subspace. We pick a rotation that is invariant with respect to V_m, but rotates C into our arbitrary fixed subspace V'. Now that the centers lie within V', the coreset property guarantees us that the cost of $A^{(m)}$ is with rotated center set is approximated. As the cost has not been changed by the rotation, the guarantee holds for the original center set C as well. Notice that the dependency on the dimension of the employed coreset construction is turned into a dependency on $m+k = \mathcal{O}(k/\varepsilon^2)$.

In the following, we use $V' = V_{m+k}$ as it is a convenient choice. For the application of the coreset construction it might be necessary to have the point set defined on $\mathbb{R}^{m+k}$, or it might be sufficient that the point set lies in a $(m+k)$-dimensional subspace. In the first case, we can compute a $(m+k)$-dimensional representation $\bar{A}^{(m)}$ of $A^{(m)}$ by using the coordinates of the points in the orthonormal basis $v_1, \ldots, v_{m+k}$ consisting of the first $m+k$ singular vectors. More precisely, if $A^{(m)}_{j*} = \sum_{i=1}^{d} \alpha_i v_i$, we set $\bar{A}^{(m)}_j = (\alpha_1, \ldots, \alpha_{m+k})^T \in \mathbb{R}^{m+k}$ (notice that $\alpha_{m+1}, \ldots, \alpha_{m+k}$ are zero). We can translate the coreset back to $\mathbb{R}^d$ by using the basis of V_{m+k}.

Algorithm 4.3: Computing a coreset with offset, second algorithm.

Algorithm k-`means coreset computation(`A,k,ε`)`
  Set $m = \lceil 17k/(\varepsilon/3)^2 \rceil = \lceil 153k/\varepsilon^2 \rceil$;
  For $i = 1, \ldots, m$, compute the singular value σ_i of A and corresponding u_i and v_i;
  Set $A^{(m)} = \sum_{i=1}^{m} \sigma_i u_i v_i^T$ and $\mathfrak{D} := \sum_{i=m+1}^{d} \sigma_i^2$;
  If necessary, replace $A^{(m)}$ by its $m+d$-dimensional representation $\bar{A}^{(m)}$;
  Compute a $(1+\varepsilon/3)$-coreset S with weights $w : S \to \mathbb{N}$ for $A^{(m)}$ with $\mathcal{A}$;
  return S, w and $\mathfrak{D}$;

Theorem 4.4.1. *Let $A \in \mathbb{R}^{n \times d}$ be a matrix, let $k \in \mathbb{N}_{\geq 1}$, $\varepsilon \in (0,1)$ and $\alpha \in \mathbb{R}^+$. Let $n, d \geq \lceil 153k/\varepsilon^2 \rceil + k$. Let $\mathcal{A}$ be an algorithm that computes a $(1+\varepsilon')$-coreset of size $s(n, d, \varepsilon'^{-1}, \delta^{-1})$ from a d-dimensional point set with n points with probability $1-\delta$. Then algorithm 4.3 computes a $(1+\varepsilon)$-coreset with offset of size $s(n, 153k/\varepsilon^2, 3/\varepsilon, \delta^{-1}))$ with probability $1-\delta$.*

Proof. Assume that $\mathcal{A}$ is successful, which happens with probability $1-\delta$. Let S' be the coreset computed for either $A^{(m)}$ or $\bar{A}^{(m)}$. In the latter case, S' is the d-dimensional version of the coreset that we get by using the basis of V' to map the coreset back to $\mathbb{R}^d$. Notice that S' is a coreset for $A^{(m)}$ regarding all choices of centers from V_{k+m}. Let C be an arbitrary set of k centers from $\mathbb{R}^d$ which does not necessarily lie within V_{k+m}. Recall that there exists an orthonormal basis of $\mathbb{R}^d$ that consists of singular vectors $v_1, \ldots, v_d$, and that $v_1, \ldots, v_i$ is an orthonormal basis of V_i. Choose $b_{m+1}, \ldots, b_d$ such that $v_1, \ldots, v_m, b_{m+1}, \ldots, b_{m+k}$ form an orthonormal basis of the $m+k$ dimensional subspace $V'' := V_m \cup \operatorname{span}(C)$ and such that $v_1, \ldots, v_m, b_{m+1}, \ldots, b_d$ is an orthonormal basis of $\mathbb{R}^d$. Now we define a mapping

$\pi : \mathbb{R}^d \to \mathbb{R}^d$ that is invariant with regard to V_m and rotates $\operatorname{span}(C)$ into V_{m+k}. This is achieved by $\pi(x) = Tx$ with

$$T := \sum_{i=1}^{m} v_i v_i^T + \sum_{i=m+1}^{d} v_i b_i^T.$$

For a vector $x \in \mathbb{R}^d$, consider its representation in the basis of V'', $x = \sum_{i=1}^{m} \alpha_i v_i + \sum_{i=m+1}^{d} \alpha_i b_i$. Then we see that $\pi(x)$ satisfies

$$\begin{aligned}\pi(x) &= \Big(\sum_{i=1}^{m} v_i v_i^T + \sum_{i=m+1}^{d} v_i b_i^T\Big)\Big(\sum_{i=1}^{m} \alpha_i v_i + \sum_{i=m+1}^{d} \alpha_i b_i\Big) \\ &= \sum_{i=1}^{m} \alpha_i v_i + \sum_{i=m+1}^{d} \alpha_i v_i.\end{aligned}$$

In particular, if a vector lies in $\operatorname{span} C$, then α_i is only non-zero for $i = m+1, \ldots, m+k$, so the image of the vector is $\sum_{i=m+1}^{k} \alpha_i v_i \in V_{m+k}$. Additionally, $\pi(x) = x$ for all $x \in V_m$. Notice that π does not change the lengths of vectors, as $||\pi(x)||^2 = ||\sum_{i=1}^{m} \alpha_i v_i + \sum_{i=m+1}^{d} \alpha_i v_i.||^2 = \sum_{i=1}^{d} \alpha_i^2 = ||x||^2$ by the Pythagorean theorem and because the basis is orthonormal.

Let $\pi(C) := \{\pi(c) | c \in C\}$ be the rotation of C into V_{m+k} and set $\varepsilon' = \varepsilon/3$. By the coreset guarantee of C we know that $|\operatorname{cost}_{\ell_2^2}(A^{(m)}, \pi(C)) - \operatorname{cost}_{\ell_2^2}(S', \pi(C))| \leq \varepsilon' \cdot \operatorname{cost}_{\ell_2^2}(A^{(m)}, \pi(C))$. Notice that for a point A_{j*} and a center $c \in C$, $||A_{j*} - \pi(c)||^2 = ||\pi(A_{j*}) - \pi(c)||^2 = ||\pi(A_{j*} - c)||^2 = ||A_{j*} - c||^2$ because π is linear and invariant with regard to V_m, and because it does not change the lengths of vectors. This implies $\operatorname{cost}_{\ell_2^2}(A^{(m)}, \pi(C)) = \operatorname{cost}_{\ell_2^2}(A^{(m)}, C)$ and $\operatorname{cost}_{\ell_2^2}(S', \pi(C)) = \operatorname{cost}_{\ell_2^2}(S', C)$, and therefore

$$|\operatorname{cost}_{\ell_2^2}(A^{(m)}, C) - \operatorname{cost}_{\ell_2^2}(S', C)| \leq \varepsilon' \cdot \operatorname{cost}_{\ell_2^2}(A^{(m)}, C).$$

By Theorem 3.2.3, we know that $|\operatorname{cost}_{\ell_2^2}(A^{(m)}, C) + \mathfrak{D} - \operatorname{cost}_{\ell_2^2}(A, C)| \leq \varepsilon' \cdot \operatorname{cost}_{\ell_2^2}(A, C)$. By the triangle inequality, we conclude that

$$\begin{aligned}&|\operatorname{cost}_{\ell_2^2}(S', C) + \mathfrak{D} - \operatorname{cost}_{\ell_2^2}(A, C)| \\ &\leq |\operatorname{cost}_{\ell_2^2}(S', C) - \operatorname{cost}_{\ell_2^2}(A^{(m)}, C)| + |\operatorname{cost}_{\ell_2^2}(A^{(m)}, C) + \mathfrak{D} - \operatorname{cost}_{\ell_2^2}(A, C)| \\ &\leq \varepsilon' \cdot \operatorname{cost}_{\ell_2^2}(A^{(m)}, C) + \varepsilon' \cdot \operatorname{cost}_{\ell_2^2}(A, C) \leq \ \varepsilon' \cdot (\operatorname{cost}_{\ell_2^2}(A^{(m)}, C) + \mathfrak{D}) + \varepsilon' \cdot \operatorname{cost}_{\ell_2^2}(A, C) \\ &\leq (\varepsilon'(1 + \varepsilon') + \varepsilon') \cdot \operatorname{cost}_{\ell_2^2}(A, C) \leq 3\varepsilon' \cdot \operatorname{cost}_{\ell_2^2}(A, C) \leq \varepsilon \cdot \operatorname{cost}_{\ell_2^2}(A, C).\end{aligned}$$

□

We can combine Theorem 4.4.1 with existing coreset computations. For example, using the construction by Langberg and Schulman in [LS10], Theorem 4.4.1 implies the existence of a coreset of size $\tilde{\mathcal{O}}(k^5 \varepsilon^{-6})$. This size is independent of the dimension and of the number of input points.

Notice that we can extend the coreset construction to weighted inputs by using the technique described in Section 3.2.1 as done in Algorithm 4.4. The algorithm assumes that the input contains a weight function $w : [n] \to \mathbb{N}$ that assigns an integer weight to every index. The coreset construction needs to be applicable to weighted inputs, too, since we apply it to $A_w^{(m)}$ weighted by w. The construction gives the same guarantees as the unweighted version.

Algorithm 4.4: Computing a coreset with offset for inputs with integer weights.

Algorithm k-`means coreset computation(`A,k,ε,w`)`
  Set $m = \lceil 17k/(\varepsilon/3)^2 \rceil = \lceil 153k/\varepsilon^2 \rceil$;
  Obtain F by multiplying each row A_{i*} with $\sqrt{w(i)}$;
  For $i = 1, \ldots, m$, compute the singular value σ_i of F and corresponding u_i and v_i;
  Set $A_w^{(m)} = \sum_{i=1}^{m} \sigma_i u_i v_i^T$ and $\mathfrak{D} := \sum_{i=m+1}^{d} \sigma_i^2$;
  If necessary, replace $A_w^{(m)}$ by its $m + d$-dimensional representation $\bar{A}^{(m)}$;
  Compute a $(1 + \varepsilon/3)$-coreset S with weights $w' : S \to \mathbb{N}$ for $A_w^{(m)}$ and w;
  return S, w' and $\mathfrak{D}$;

5 The k-means problem in data streams

A *stream* of data is a large list of items (points, in our case), which are presented one by one in an arbitrary order, meaning that the order cannot be influenced by an algorithm that wants to analyze the data. A *streaming algorithm* can read the data only once in the given order and can only store a sublinear amount of data.

Muthukrishnan [Mut05] gives an extensive survey on techniques for designing data stream algorithms. We look at one of his introductory puzzles as an easy example for a data stream algorithm. Consider a data stream consisting of numbers from the universe $\{1, \dots, u\}$ for a $u \in \mathbb{N}$. Assume that the stream consists of $u - 1$ numbers, all distinct, and that the task is to find the missing number. This can be solved by adding all numbers in the stream and subtracting the sum from $\sum_{i=1}^{u} i = u(u+1)/2$, which needs less than $2 \log u$ bits of storage.

The storage can be reduced even further to only $\log u$ bits of storage[1]. For the sake of simplicity, assume that $u = 2^k - 1$. Then notice that the 'bitwise xor' over all numbers from 0 to $2^k - 1$ is zero, and that if we leave out one number, the bitwise xor of all other numbers is exactly this number. So, the missing number can also be found by computing the bitwise xor over all numbers in the stream, which can be done with $\log u$ bits. This is optimal since storing the solution for the puzzle requires $\log u$ bits (in the worst case).

As an easy example for a data stream algorithm in the area of clustering, consider the k-means problem for $k = 1$ and assume that we want to approximate the 1-means cost. So, the stream consists of points $x_1, x_2, \dots$, and after every point x_j in the stream, we want to be able to output $\text{dist}^2(P_j, \mu(P_j))$ for $P_j := \{x_1, x_2, \dots, x_j\}$.

By Lemma 2.4.1, we can write the 1-means cost of P_j as $\text{dist}^2(P_j, \mu(P_j)) = \sum_{x \in P_j} \|x - \mu(P_j)\|^2 = |P_j| \cdot \|z - \mu(P_j)\|^2 - \sum_{x \in P_j} \|x - z\|^2$ for any $z \in \mathbb{R}^d$. We set $z = 0$ and find that

$$\text{dist}^2(P_j, \mu(P_j)) = |P_j| \cdot \|\mu(P_j)\|^2 - \sum_{x \in P_j} \|x\|^2.$$

Consequently, we store the number of points $N_j := |P_j|$ seen so far, the sum of the points, $s_j = \sum_{i=1}^{j} x_i$, and the sum of the squared lengths, $L_j = \sum_{i=1}^{j} ||x_i||^2$. These features can be updated after every point, and they are sufficient to calculate $\text{dist}^2(P_j, \mu(P_j)) = N_j \cdot \|s_j/N_j\|^2 - L_j$. The space complexity (in the uniform cost model) is one d-dimensional point and two constants, N_j and L_j.

We have seen this triple before, in form of an exact 1-means coreset consisting of one weighted point and a constant. In Section 5.1 we see that strong (approximative) coresets for the k-means problem generally lead to streaming algorithms.

[1] Optimizing constants is usually not our focus, but it is interesting in this small introductory example.

Analysis of streaming algorithms. The main evaluation criterion for streaming algorithms is their space complexity which shall be polylogarithmic or at least sublinear in the size of the stream. The running time of streaming algorithms is often differentiated into the *update time*, i. e., the time needed to process one new item in the stream, and the *extraction time*, i. e., the time needed to extract a solution for the given problem from the stored data. Both these times should be polylogarithmic in the stream size. We will mostly analyze the (total) *running time*, i.e. the time needed to process a stream of given length and to output a solution.

When the streaming restriction of only one pass over the data is relaxed and multiple runs over the data are allowed, then the number of passes becomes another measure of the performance of a streaming algorithm. In this thesis we only deal with single pass algorithms, so we do not further elaborate on this matter.

Notice that in the first example above, we used the *logarithmic cost model* to analyze the space complexity because we counted the bit length of the involved numbers. This makes sense as the input is a number problem. In the context of clustering problems and thus in most parts of this thesis, we will simplify the analysis by using the *uniform cost model* to analyze algorithms, i. e., arithmetic operations are assumed to need constant time and numbers to use constant space.

Streaming models. We can perceive a data stream as a description of an underlying vector x of a possibly infinite dimension. Muthukrishnan [Mut05] distinguishes between three data streaming models that differ in the way that the vector is updated.

In the *Time-Series model*, the entries of x arrive consecutively ordered by their index, i. e., the stream consists of $x_1, x_2, x_3, \ldots$ and so on. For example, a sensor measuring a certain value every five minutes creates a time series.

In contrast to this, the other two models assume that the data stream contains *updates* on x, and x is initially assumed to be the zero vector. In the *Cash-Register model*, the updates are always positive, so each element in the stream specifies a position i and a positive value which is added to x_i. The first example above falls into this category: The underlying vector is the characteristic vector of the universe $\{1, \ldots, u\}$ and each arriving number increases the corresponding coordinate of this vector from 0 to 1. This data stream model was named in [GKMS01].

If the entries of x can also be *decreased*, then we have the *Turnstile model*. Every element in the stream gives a position i and a value that is added to x_i, and this value can be positive or negative. The model was named in [Mut05] after the turnstiles in subways in New York which keep track of the number of people currently in the subway. If, like in the subway example, the updates are restricted such that they can never decrease a coordinate of x below zero, then the model is called *strict* Turnstile model.

The above models are general models for data streams. For geometric problems, two other terms have prevailed, which are the *Insertion-Only (data stream) model* and the *Dynamic (data stream) model* which are special names for geometric versions of the Cash-Register model and the Turnstile model. In the Insertion-Only model, the stream consists

of points $x_1, x_2, \ldots$ which arrive in arbitrary (worst-case) order. We can see this as a Cash-Register model if we define the underlying vector as the (infinite) characteristic vector of all possible points.

The Dynamic data stream model is a specific model for the discrete d-dimensional Euclidean space defined by Indyk [Ind04]. It assumes that the points lie on a grid, more specifically, that all input points are from $\{1, \ldots, \Delta\}^d$ for a constant Δ. The stream consists of insertion and deletion operations, i. e., each element in the stream either adds a point or deletes a point. It is assumed that points can only be added if they are currently not present, and that they can only be deleted if they are currently present. This means that the model is a strict Turnstile model where the underlying vector is the characteristic vector of all Δ^d possible points.

Randomization and approximation. The introductory examples both have the property that they can be solved deterministically and exactly. This is typically not the case, and to illustrate this we have a look at a well understood data stream problem which was proposed by Flajolet and Martin in 1983 [FM83, FM85].

Assume that the stream again consists of numbers from the universe $U := \{1, \ldots, u\}$ for a $u \in \mathbb{N}$, but this time, the numbers can repeat. We are interested in the *number of distinct elements*, i. e., in the question how many different numbers we have seen. This is easy if we have $\Theta(u)$ bits because then we can store for each number whether we have seen it. However, we would like the space complexity of the algorithm to be sublinear.

Assume that we could solve this problem deterministically in space $o(u)$, i. e., after seeing a stream of numbers and storing only $o(u)$ information, we could output the number of distinct elements exactly. Then we claim that we can use the corresponding algorithm to do the following impossible task: Given an arbitrary subset S of U, store S with $o(u)$ bits and later recover it completely.

So, let $S \subseteq U$ be an arbitrary subset of U. We feed this to the algorithm as a stream in an arbitrary order. The algorithm processes S and only stores $o(u)$ bits. Then, we iteratively append all numbers from 1 to u to the stream. After each number, we check whether the number of distinct elements has changed. If so, then the number was not in S, if not, then it was in S. This means that the algorithm can store arbitrary sets $S \subseteq \{1, \ldots, u\}$ with $o(u)$ bits, which is not possible[2].

Thus, the distinct element problem cannot be solved exactly by a deterministic data stream algorithm with $o(u)$ storage. Interestingly, even approximation is not enough if we stick to determinism: Any deterministic algorithm which computes a constant approximation needs $\Omega(u)$ space [AMS99]. On the other hand, various randomized approximation algorithms were proposed, including a $(1 + \varepsilon)$-approximation with a space complexity of $O(\varepsilon^{-2} + \log u)$ [KNW10] which is optimal [AMS99, Woo04].

For the k-means clustering problem, the use of approximation is natural (at least for designing efficient algorithms) as the problem is NP-hard. Surprisingly and in contrast to

[2]This argumentation follows a similar argumentation for the slightly different second puzzle in [Mut05].

the distinct element problem, randomization is not necessarily needed to design streaming algorithms for the k-means problem, see for example Section 5.2.1.

5.1 The Merge-and-Reduce technique

This section discusses a popular method to design efficient streaming algorithms based on coresets. Strong coresets have the nice property that their union is also a coreset. More precisely, if S_1 and S_2 are $(1+\varepsilon)$-coresets for a problem on the input point sets P_1 and P_2, respectively, then $S_1 \cup S_2$ is a coreset for $P_1 \cup P_2$. Additionally, coreset computations can be nested: If S_1 is a $(1+\varepsilon_1)$-coreset for S_2, and S_2 is a $(1+\varepsilon_2)$-coreset for P, then S_1 is a $(1+\varepsilon_1)(1+\varepsilon_2)$-coreset for P.

So, a reasonable idea to tranfer a coreset construction to the Insertion-Only data stream model is to partition the input data into chunks, compute a coreset for each chunk and union the resulting coresets. Whenever this union grows too large, it is reduced by applying the coreset construction again. However, every union step introduces an additional error, so the number of unions has to be kept small. More precisely, when looking at a particular point, the number of unions and coreset computations that it participates in shall be kept small.

This is exactly what the Merge-and-Reduce technique is about. For each chunk and the coreset(s) computed from it, the Merge-and-Reduce technique keeps the number of unions small that it participates in. This is achieved by merging and reducing coresets in a tree-like fashion. See Figure 5.1 for a visualization. The first chunk (or block) B_1 is read and reduced to coreset S_1, the second block B_2 to coreset S_2. Then, S_1 and S_2 are merged and reduced to coreset S_3. Blocks B_3 and B_4 induce the coreset computation of S_4 and S_5 first and S_6 then. When S_6 is computed, it is merged with S_3 and reduced to S_7. In this way the computation continues: Each block is summarized into a coreset which forms a leaf of the tree, and whenever both children of a node have computed their coresets, then the node computes its coreset, too.

The advantage of this computation tree is that every input point takes part in at most $\log|P|$ reduction steps, where $|P|$ is the number of points seen so far. Thus, if every reduction step computes a $(1+\varepsilon)$-coreset of its input, then the resulting coreset is a $(1+\varepsilon)^{\log|P|}$-coreset. To compensate this accuracy loss, the reduction steps have to be a bit more accurate: They have to compute $(1+\varepsilon')$-coresets for $\varepsilon' := \varepsilon/\log|P|$.

Notice that we may have to store multiple coresets at the same time. However, it is never necessary to store more than one coreset for each level (plus one, during the time where a new chunk of data is incorporated into the tree).

So far, we described Merge-and-Reduce with a focus on standard coresets (without offset). The offsets do not change the framework much. At the beginning of this section, we observed that the union of coresets is a coreset and that coreset computations can be nested. The following lemma states that this still holds for coresets with offsets. This is central and ensures that we can still use the Merge-and-Reduce technique. The only

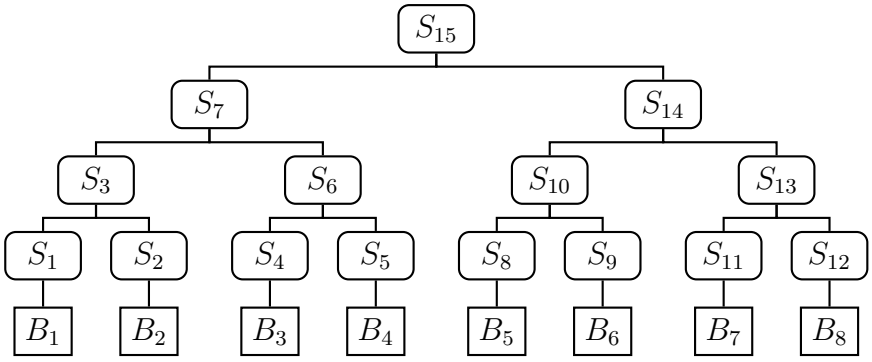


Figure 5.1: Merge-and-Reduce computation tree.

changes are that we have to store $O(\log n)$ constants, and that we have to take care of the constants when we merge coresets.

Lemma 5.1.1. *Let S_1 and S_2 with constants $\mathfrak{D}_1$ and $\mathfrak{D}_2$ be $(1+\varepsilon)$-coresets with offsets for the point sets P_1 and P_2, respectively. Let S' with $\mathfrak{D}'$ be a $(1+\varepsilon')$-coreset with offset for $S_1 \cup S_2$. Then $S_1 \cup S_2$ with constant $\mathfrak{D}_1 + \mathfrak{D}_2$ is a $(1+\varepsilon)$-coreset for $P_1 \cup P_2$, and S' with constant $\mathfrak{D}_1 + \mathfrak{D}_2 + \mathfrak{D}'$ is a $(1+\varepsilon)(1+\varepsilon')$-coreset for $S_1 \cup S_2$.*

Proof. For the first statement we only need the triangle inequality and the fact that the cost of a point set is the sum of the costs of its subsets (if the subsets form a partitioning). We get that for every set of k centers C, it holds that

$$\begin{aligned}
&|\mathrm{cost}_{\ell_2^2}(S_1 \cup S_2, C) + \mathfrak{D}_1 + \mathfrak{D}_2 - \mathrm{cost}_{\ell_2^2}(P_1 \cup P_2, C)| \\
\leq\ &|\mathrm{cost}_{\ell_2^2}(S_1, C) + \mathfrak{D}_1 - \mathrm{cost}_{\ell_2^2}(P_1, C)| + |\mathrm{cost}_{\ell_2^2}(S_2, C) + \mathfrak{D}_2 - \mathrm{cost}_{\ell_2^2}(P_2, C)| \\
\leq\ &\varepsilon \cdot \mathrm{cost}_{\ell_2^2}(P_1, C) + \varepsilon \cdot \mathrm{cost}_{\ell_2^2}(P_2, C) = \varepsilon \cdot \mathrm{cost}_{\ell_2^2}(P_1 \cup P_2, C).
\end{aligned}$$

Based on this, we can bound the error of S' compared to the cost of $P_1 \cup P_2$. It holds that

$$\begin{aligned}
&|\mathrm{cost}_{\ell_2^2}(S', C) + \mathfrak{D}_1 + \mathfrak{D}_2 + \mathfrak{D}' - \mathrm{cost}_{\ell_2^2}(P_1 \cup P_2, C)| \\
\leq\ &|\mathrm{cost}_{\ell_2^2}(S', C) + \mathfrak{D}' - \mathrm{cost}_{\ell_2^2}(S_1 \cup S_2, C)| \\
&\qquad + |\mathrm{cost}_{\ell_2^2}(S_1 \cup S_2, C) + \mathfrak{D}_1 + \mathfrak{D}_2 - \mathrm{cost}_{\ell_2^2}(P_1 \cup P_2, C)| \\
\leq\ &\varepsilon' \mathrm{cost}_{\ell_2^2}(S_1 \cup S_2, C) + \varepsilon \cdot \mathrm{cost}_{\ell_2^2}(P_1 \cup P_2, C) \\
\leq\ &\varepsilon'(1+\varepsilon) \mathrm{cost}_{\ell_2^2}(P_1 \cup P_2, C) + \varepsilon \cdot \mathrm{cost}_{\ell_2^2}(P_1 \cup P_2, C) \\
=\ &(\varepsilon + \varepsilon' + \varepsilon\varepsilon') \cdot \mathrm{cost}_{\ell_2^2}(P_1 \cup P_2, C).
\end{aligned}$$

Notice that $(1+\varepsilon)(1+\varepsilon') = 1 + (\varepsilon + \varepsilon' + \varepsilon\varepsilon')$, which concludes the proof. □

Merge-and-Reduce is not a new technique but has already been used in the work of Bentley and Saxe [BS80]. Agarwal, Har-Peled and Varadarajan [AHPV04] used it in

the context of computing extent measures of points in data streams, and Har-Peled and Varadarajan [HPM04] used it to compute coresets in data streams. It has since then been stated and rephrased several times. We give a version that allows offsets and the usage of different coreset algorithms. Notice that the theorem assumes that the number of points is known in advance. We discuss this point below.

Theorem 5.1.2. *Let $0 < \varepsilon \leq 1/2$ and $0 < \delta < 1$ be given. Assume we are given an algorithm that, given a weighted point set P of n points with total weight W, computes a $(1+\varepsilon)$-coreset of P for the k-means problem, possibly with offset $\mathfrak{D}$, of size $s(n, W, k, \varepsilon, \delta) \geq 1$ with total weight W in time $t(n, W, k, \varepsilon, \delta)$ with failure probability δ, and a storage usage of $s'(n, W, k, \varepsilon, \delta)$. We assume that $s(\cdot), s'(\cdot)$ and $t(\cdot)$ are monotone functions.*

Then we can specify an algorithm in the Insertion-Only data stream model which computes a $(1+\varepsilon)$-coreset of an unweighted point set P for D with probability $1-\delta$ where for $\varepsilon' := \varepsilon/(2\lceil \log n \rceil + 2)$ and $\delta' := \delta/(2n)$ it holds that

- *the size of the coreset is bounded by $\tilde{s} := s(n, n, k, \varepsilon', \delta')$,*
- *the storage size is bounded by $s'(\ell \cdot \tilde{s}, n, k, \varepsilon', \delta') + \ell \cdot \tilde{s} \cdot d$ with $\ell := \lceil \log n \rceil + 1$ and*
- *the running time is bounded by $(n/|\tilde{s}|) \cdot t(2\tilde{s}, n, k, \varepsilon', \delta') + t(\ell\tilde{s}, n, k, \varepsilon', \delta')$.*

Proof. We review a different description of the Merge-and-Reduce technique with the following algorithm which fits to the computation tree in Figure 5.1 but makes more clear what coresets need to be stored during the computation. We use $\tilde{s} := s(n, n, k, \varepsilon', \delta')$ as the basic chunk size in which the data is partitioned because it does not induce additional storage requirement as we want to output a coreset of this size anyhow.

We store a dynamic array (or a list) A of coresets with offsets[3], where new elements are created when needed and new elements start empty. If $A[i]$ holds a coreset, we denote it by S_i^A and its offset by $\mathfrak{D}_i^A$. The offset is zero if the coreset construction does not use offsets. Notice that S_i^A and $\mathfrak{D}_i^A$ may change during the execution of the algorithm.

In every step of the algorithm, we do the following. First, we read $2 \cdot \tilde{s}$ input points and compute a $(1+\varepsilon')$-coreset S with offset $\mathfrak{D}$ of size $s(\tilde{s}, \tilde{s}, k, \varepsilon', \delta')$ with total weight $\tilde{s}$. We set $i = 1$ and try to store S at position $A[i]$. If $A[i]$ is empty, we store S and $\mathfrak{D}$ and are done. Otherwise, $A[i]$ contains a coreset S_i^A with offset $\mathfrak{D}_i^A$ of the same total weight as S. We merge S_i^A and S, and then reduce their union by another coreset computation to a coreset S' with offset $\mathfrak{D}''$. Also, we add the constants and obtain $\mathfrak{D}' := \mathfrak{D} + \mathfrak{D}_i^A + \mathfrak{D}''$. $A[i]$ is emptied, we set $i = i + 1$ and try to insert S' annotated with $\mathfrak{D}'$ into $A[i]$ in the same fashion. This repeats until an empty position in the array is reached, which will happen eventually.

When the current coreset is queried, we take the union of all S_i^A currently stored in A, compute a coreset of the union and return it together with the sum of the constants.

As an illustration, notice that in Figure 5.1, S_1 is the first coreset stored in $A[1]$, but is removed when S_2 arrives. S_2 is never stored in A, but S_3 is stored in $A[2]$ until S_6 is computed, when S_3 is removed and S_7 is computed and stored in $A[3]$.

[3]We call it dynamic array because we use $A[i]$ to reference the ith element, but a list suffices.

During the algorithm, the following always holds. If $A[i]$ for $i \geq 1$ contains a coreset S_i^A with offset $\mathfrak{D}_i^A$, then this coreset represents a set P' containing $2^i\tilde{s}$ input points, has weight $2^i\tilde{s}$ and is of size $s(2\tilde{s}, 2^i\tilde{s}, k, \varepsilon', \delta') \leq \tilde{s}$ (assuming that n is a power of two and that $2\tilde{s} \leq n$)[4]. Furthermore, S_i^A is a $(1+\varepsilon)^i$-coreset for P'. To see this, notice that following the construction of S_i^A above, it is a coreset for $S_{i-1}^A \cup S$. S_{i-1}^A and S are coresets for disjoint subsets of P' and together represent P'. If we assume that S_{i-1}^A is a $(1+\varepsilon')^{i-1}$-coreset and notice that S is a $(1+\varepsilon')$-coreset, then Lemma 5.1.1 yields that S' is a $(1+\varepsilon')^i$ coreset.

Notice that $i^* = \lceil \log |P| \rceil + 1$ array positions suffice to store a coreset of a point set P. Thus, we need to store at most $(\lceil \log |P| \rceil + 1) \cdot \tilde{s}$ points for the coresets, plus additional storage size for the execution of the coreset construction. For the latter, notice that the largest input to the coreset construction is the union of the $(\lceil \log |P| \rceil + 1)$ coresets that is reduced to one coreset. Here, the storage needed is bounded by $s'((\lceil \log |P| \rceil + 1) \cdot \tilde{s}, |P|, k, \varepsilon', \delta')$. The returned coreset is of size $s(2\tilde{s}, 2^{(\lceil \log |P| \rceil + 1)}\tilde{s}, k, \varepsilon', \delta') \leq \tilde{s}$.

All coresets stored in A are at least $(1+\varepsilon')^{i^*}$-coresets. The final computation unions all i^* coresets and performs a final coreset computation, yielding a $(1+\varepsilon')^{i^*+1}$-coreset by Lemma 5.1.1. For $\varepsilon' := \varepsilon/(2(\lceil \log |P| \rceil + 2))$, we get for $\varepsilon \leq 1/2$ that

$$\left(1 + \frac{\varepsilon}{2(\lceil \log |P| \rceil + 2)}\right)^{\lceil \log |P| \rceil + 2} \leq e^{\varepsilon/2} \leq \frac{1}{1 - \varepsilon/2} = 1 + \frac{\varepsilon/2}{1 - \varepsilon/2} \leq 1 + \varepsilon.$$

Similarly, we have $(1 - \frac{\varepsilon}{2(\lceil \log |P| \rceil + 2)})^{\lceil \log |P| \rceil + 2} \geq (2e)^{-\varepsilon/2} \geq e^{-\varepsilon} \geq (1 - \varepsilon)$.

Set $n' := |P|/(2\tilde{s})$. On the base level, we do n' coreset computations, which together take $n' \cdot t(2\tilde{s}, 2\tilde{s}, k, \varepsilon', \delta')$ time. Because the computation has the structure of a binary tree, the number of coreset computations in inner nodes of the tree is at most $n' - 1$, each processing a set of size $2\tilde{s}$ with weight at most $|P|$. So, the computation time for all coreset computations except the last computation is bounded by $2n' \cdot t(2\tilde{s}, |P|, k, \varepsilon', \delta')$. The final coreset computation takes time $t((\lceil \log |P| \rceil + 1)\tilde{s}, |P|, k, \varepsilon', \delta')$.

Finally, notice that the probability that one of these $2n'$ coreset computations fails is bounded by $2n'\delta/(2|P|) \leq \delta$ by the union bound because $n' \leq |P|$.

□

Knowing the number of points. Notice that the above algorithm assumes knowledge about $|P|$, the number of points in the data stream. It is needed to set the precision parameter ε' and thus the coreset size. By increasing the storage size by another $\log |P|$ factor, we can get rid of this assumption. When the stream starts, we assume an upper bound n on the number of points in the stream (maybe we expect at least one point, or k points, or any constant number). We set the parameters accordingly and compute a $(1 + \varepsilon/3)$-coreset for the first n points in the stream. Then, we double our bound, adjust the parameters and compute a coreset for the next $2n$ points in the stream. We proceed like this, always doubling the bound when we reached the estimated number of points. We

[4] If n is not a power of two and $2^i\tilde{s} \geq n$, then the set in $A[i]$ is empty or represents $|P|$ points and has size $s(n, n, k, \varepsilon', \delta') = \tilde{s}$.

store all the coresets computed in this way, and whenever the current coreset is queried, we compute a $(1+\varepsilon/3)$-coreset of the union of all stored coresets and return it. The error is then $(1+\varepsilon/3)^2 \leq (1+\varepsilon)$, the coreset size is $S(|P|, k, \varepsilon/3)$ for the actual number of points $|P|$ seen in the stream, but the storage size increases by a factor of $\log |P|$ because of the $\log |P|$ coresets that we have to store.

5.2 Streaming algorithms for k-means clustering

In the last section, we saw the Merge-and-Reduce technique, which provides a rather general way to design streaming algorithms. We have already heard about several coreset constructions for the k-means problem in Section 4.2.3. Most of these were used or could at least be used to design streaming algorithms with the Merge-and-Reduce technique using Theorem 5.1.2 or a similar result. There are three exceptions. Feldman, Monemizadeh and Sohler [FMS07] construct a *weak* coreset. Thus, they cannot use the principle directly but have to adjust it accordingly. The same holds for the streaming algorithm due to Feldman and Langberg [FL11a]. Even though their work contains the construction of a strong coreset, the stated streaming result relies on a weak coreset, and the authors do not state a result for the strong coreset. Frahling and Sohler [FS05] do not use the Merge-and-Reduce technique at all. They design a strong coreset that can be maintained in the Dynamic geometric data stream model, so deletions of points are also allowed. Their coreset construction uses statistics about the point set that can be updated under insertions and deletions of points.

Table 5.1 lists the storage requirement of streaming algorithms developed for the k-means problem, measured by the number of stored points. We focus on algorithms that output a coreset which allows the computation of a $(1+\varepsilon)$-approximation (we include the two results involving weak coresets stated above). Table 5.2 lists the time needed to update the coreset when a new point arrives (in the case of [FS05], the update time is also a bound for the time needed to update after the deletion of a point). Feldman and Langberg [FL11a] do not explicitly state this time. Notice that performing a streaming algorithm on a stream of n points needs n updates, yielding a running time of n times the update time. Usually, the extraction time of a coreset is bounded by the storage need which is implicitly assumed to be $o(n)$ for streaming algorithms.

5.2.1 Algorithms based on Merge-and-Reduce

We shortly review results that are based on the Merge-and-Reduce technique. Theorem 5.1.2 allows us to convert coreset constructions for the k-means problem into streaming algorithms.

Storage and time in typical situations. To get some more intuition, we look at the consequences of Theorem 5.1.2 for a more restricted scenario. Let a coreset construction be given. Assume that the size of the computed coreset depends on the total weight but

Authors	Year	Max. number of stored points	Reference
k and d are arbitrary			
Feldman, Langberg	2011	$\mathcal{O}(k\varepsilon^{-3}\log\varepsilon^{-1}\log^4 n)$	[FL11b]
Feldman, Monemizadeh, Sohler	2007	$\mathcal{O}(k^2\varepsilon^{-5}\log^{10} n)$	[FMS07]
Chen	2006	$\mathcal{O}(dk^2\varepsilon^{-2}\log^8 n)$	[Che09]
k is arbitrary, d is a constant			
Frahling, Sohler	2005	$\mathcal{O}(k^2\varepsilon^{-2d-6}\log^4\Delta(\log\Delta+\log m)^3)$	[FS05]
Har-Peled, Mazumdar	2004	$\mathcal{O}(k\varepsilon^{-d}\log^{2d+2} n)$	[HPM04]

Strong coresets, but streaming result not made explicit	Year	Reference
k and d are arbitrary		
Langberg, Schulman	2010	[LS10]
k is arbitrary, d is a constant		
Ackermann, Lammersen, Märtens, Raupach, Sohler, Swierkot	2010	[AMR+12]
Har-Peled, Kushal	2005	[HPK07]

Table 5.1: A list of results for computing coresets for k-means in the streaming setting.

Authors	Year	Update Time	Reference
k and d are arbitrary			
Feldman, Monemizadeh, Sohler	2007	$\mathcal{O}(dk^2\varepsilon^{-1}\log^2 n)$	[FMS07]
Chen	2006	$\mathcal{O}(dk(\log^2 n)\operatorname{poly}\log(dk\varepsilon^{-1}))$	[Che09]
k is arbitrary, d is a constant			
Frahling, Sohler	2005	$\mathcal{O}(\log^2\Delta(\log\Delta+\log m))$	[FS05]
Har-Peled, Mazumdar	2004	$\mathcal{O}(\log^2(k\varepsilon^{-1})+k^5)$	[HPM04]

Table 5.2: Update times for the algorithms listed in Table 5.1.

not on the number of points, that it satisfies $s(|P|, W, k, \varepsilon, \delta) = \varepsilon^{\nu} \cdot \log^{\upsilon}(\delta^{-1}) \cdot \hat{s}(W, k)$ for $\nu, \upsilon \in \mathbb{N}$, and assume that the additional storage usage is linear in the input size, i. e., $s'(|P|, W, k, \varepsilon, \delta) = \mathcal{O}(|P|)$.

Then the total storage need of the streaming version described in Theorem 5.1.2 is $\mathcal{O}(\log |P| \cdot s(|P|, |P|, k, \varepsilon', \delta')) = \mathcal{O}((\log^{\nu+1} |P|) \cdot (\log^{\upsilon} |P| + \log^{\upsilon} \delta^{-1}) \cdot \hat{s}(|P|, k))$.

If the running time of the coreset construction is at least linear in the number of points, e. g., $t(|P|, W, k, \varepsilon, \delta) = |P|^{\kappa}\hat{t}(W, k, \varepsilon, \delta)$ for a constant $\kappa \geq 1$, then the running time of the streaming version is $(|P|/\tilde{s}) \cdot \tilde{s}^{\kappa}\hat{t}(|P|, k, \varepsilon, \delta) + \ell^{\kappa}\tilde{s}^{\kappa}\hat{t}(W, k, \varepsilon', \delta') = \mathcal{O}(|P| \cdot \tilde{s}^{\kappa}) \cdot \hat{t}(|P|, k, \varepsilon', \delta')$. We compare this to $|P|^{\kappa} \cdot t(|P|, k, \varepsilon, \delta)$. For $\kappa > 1$ and small enough coreset sizes $\tilde{s}$, the streaming version is actually asymptotically faster. For $\kappa = 1$ it is only slower because of the final computation, when skipping this and returning the union of the stored coreset, it has equal asymptotic running time as the non-streaming version.

Now the application works in a rather straightforward way. As a rule of thumb, the size of the coreset constructed by the streaming version is larger than the size of the non-streaming variant by a power of $\log n$. There are two things that must be kept in mind, though. First, the coreset construction has to work for a weighted input set. Often, the generalization to a weighted point set is verbatim, but as we have seen in Section 3.2.1, this is not always the case. Second, we need to know the dependencies on the precision parameter ε and the failure probability δ in case of a randomized algorithm. In particular the dependency on δ is often dropped when stating coreset results, as we can see in Table 4.1 in Section 4.2.3.

Specific results. We review exemplary applications of the Merge-and-Reduce technique to known coreset constructions without a formal proof. For a deterministic coreset construction like the one by Har-Peled and Mazumdar [HPM04], checking the dependency on δ is not necessary. Theorem 6.2 in [HPM04] states the coreset size for a weighted input set with total weight W as $\mathcal{O}(k\varepsilon^{-d} \log W)$ for constant d, so we get a streaming algorithm with storage requirement $\mathcal{O}(k\varepsilon^{-d} \log^{d+2} n)$ for constant dimension d. The larger coreset size in the result proven by the authors is due to the fact that they use $\varepsilon/(c(i+1)^2)$ for coresets on level i of the computation tree (for a constant c), which also works but yields $2d$ powers of $\log n$ instead of d.

The construction by Chen [Che09] computes a coreset of size $\tilde{\mathcal{O}}(dk^2 \log^2 W\varepsilon^{-2} \log \delta^{-1})$ for a weighted input set with total weight W. The streaming algorithm can thus be constructed to need $\tilde{\mathcal{O}}(d^2k^2\varepsilon^{-2}(\log \delta^{-1}) \cdot (\log^6 n))$ space. As in [HPM04], the streaming result in [Che09] is obtained with a precision parameter of $\varepsilon/(c(i+1)^2)$ for the individual coreset constructions, leading to the result stated in Table 5.1.

The constructions by Har-Peled and Kushal [HPK07], Ackermann, Lammersen, Märtens, Raupach, Sohler and Swierkot [AMR+12] and Langberg and Schulman [LS10] are in principle also suitable to be embedded into Merge-and-Reduce, but the streaming result was not explicitly stated ([AMR+12] contains a streaming implementation but not a proven bound on the storage usage in the streaming setting).

Obtaining an approximative solution. For all algorithms in Table 5.1, an α-approximation can be obtained at any point of the stream by extracting the coreset and running an α-approximation algorithm.

5.2.2 A streaming coreset

Section 4.4 contains an algorithm that computes coresets for the k-means problem. The size of the computed coresets is independent of n, the number of points, and d, the dimension of the input points. The algorithm consists of computing a lower dimensional version of the input point set by the dimensionality reduction from Section 3.2 and running a coreset construction on this lower dimensional set. We shortly review how to conduct this coreset construction in a streaming setting by using the Merge-and-Reduce technique.

For Merge-and-Reduce, we need a weighted coreset construction. We discussed how to apply the dimensionality reduction to weighted input points in Section 3.2.1. Yet, the coreset construction that is applied to the lower dimensional point set also needs to be able to handle weighted inputs. Lemma 4.2.1 states that we can simulate the computation of a coreset of a weighted point set by $\mathcal{O}(\varepsilon^{-1} \log W)$ unweighted computations, where W is the total weight of the input point set. We can thus in principle use any unweighted coreset construction.

For example, combining the observation with the coreset result by Langberg and Schulman [LS10] gives a weighted coreset constrution that computes coresets that are of size $\tilde{O}(d^2k^3\varepsilon^{-3} \log W)$ under the assumption that the dependency of the coreset size on the failure probability is polylogarithmic (because we need to conduct multiple runs of the algorithm). This is no longer independent of the number of points, and that is an artefact of using Lemma 4.2.1. Using Merge-and-Reduce will introduce a dependency on $\log n$ in any case, so we do not elaborate on removing this factor.

Combining this weighted construction with the dimensionality reduction yields coresets of size $\tilde{O}(k^5\varepsilon^{-7} \log W)$. Combining the resulting coreset construction with the Merge-and-Reduce technique results in a streaming algorithm that maintains coresets of size $\tilde{O}(k^5\varepsilon^{-7} \log^8 n)$. The number of points in the coreset is thus independent of the dimension (but the dimension of the points is still d).

Notice that the high dependency on $\log n$ is mostly a consequence of using Merge-and-Reduce together with a coreset construction that has a significant polynomial dependency on ε. In Section 5.4, we see a construction that does not use Merge-and-Reduce. This coreset construction has the advantage that it is efficient in practical applications. We discuss other streaming algorithms for the k-means problem that are used in practice in the next section.

5.2.3 Implementations of streaming algorithms for k-means

We consider an algorithm to be interesting in a practical scenario if it has been implemented and the implementation is at least documented in a publication. In the best case, the

implementation is also available, either implemented by the authors or within an open framework.

MacQueen's k-means algorithm. The oldest algorithm that can be considered a streaming algorithm for the k-means problem is due to MacQueen [Mac67] and was proposed in 1967. It has certainly be implemented several times and is for example available within the open source project ESMERALDA [FP11]. MacQueen's k-means algorithm declares the first k points to be the initial centers. Then it reads the remaining points, assigns each new point to its closest center and recomputes this center. The center is replaced by the centroid of the points assigned to it. Points are never reassigned to different centers.

The algorithm's behaviour depends on the ordering of the points, and does not guarantee a local optimum like Lloyd's algorithm does after convergence. It comes without a guarantee on the quality of the solution. MacQueen did not originally design the algorithm in light of the streaming scenario, but as the algorithm only reads each point once, it can be considered to be a streaming algorithm. The running time of a straightforward implementation is $\Theta(nkd)$ because we need to compare the distance of an input point to the current k centers for $n-k$ input points, and the constant hidden in the $\mathcal{O}$-notation is small. As there is very little overhead and as $\Theta(nkd)$ is a favorable running time, MacQueen's algorithm can be implemented to be very fast in practice.

Clustering features and BIRCH. A very popular streaming heuristic for the k-means problem is BIRCH [ZRL97a], developed by Zhang, Ramakrishnan and Livny. The authors' implementation is available at [ZRL97b].

The core of BIRCH is the so called *clustering feature tree*, abbreviated as *CF tree*. It is based on the observation on the 1-means cost in Lemma 2.4.1. As we have already observed in Section 4.1, this lemma implies that the 1-means cost of a point set P can be computed by only using the number of the points, their sum and the sum of the squared lengths of all points in P. These three statistics are named *clustering features* in [ZRL97a].

BIRCH reads the points and stores them in clustering features organized in a tree. To decide where to add points, it offers four predefined distance functions to compute the distance of two clustering features (a point can be seen as a clustering feature).

The clustering features representing the input are contained in the leaves of the tree, while the inner nodes serve as a data structure to find the correct leaf. The inner nodes also contain clustering features, each clustering feature representing the subset of one child node. For a new point, BIRCH identifies the closest clustering feature in any leaf of the CF tree. The point is added to the clustering feature if the average 1-means cost of the represented subset does not exceed a given parameter, the *threshold*. Otherwise, BIRCH creates a new clustering feature for the new point.

Both the leaves and the nodes have a maximum capacity of clustering features they can store. If it is exceeded, the node is replaced by two new nodes, and the clustering features are split and assigned to the two new nodes. In the parent node, the clustering feature is

replaced by two. This can cause additional splits unless the change is propagated up to the first level of the CF tree.

The size of the CF tree is controlled by performing *rebuilding* steps whenever the desired maximum number of nodes is exceeded. The threshold is (sufficiently) increased and the CF tree is reduced by iterating through the nodes and merging clustering features if possible due to the new threshold. To compute a clustering from the CF tree, BIRCH uses a hierarchical clustering algorithm.

The authors do not state a bound on the running time, but provide empirical evidence that it is fast. BIRCH has no theoretical quality guarantees and does indeed sometimes perform badly in practice [GRS01, HBV01].

Extensions of k-means++ to the streaming setting. In the non-streaming setting, the seeding procedure of the k-means++ algorithm holds a special position among the algorithms for the k-means problem. While it provides an (admittedly mediocre) worst-case guarantee on the (expected) quality of the solution, it is easy to implement and has a running time of only $\mathcal{O}(ndk)$. Naturally, there is an interest to extend the algorithm such that it works in the streaming setting as well.

There are two algorithms transferring k-means++ into a streaming model. Ailon, Jaiswal and Monteleoni [AJM09] propose an algorithm that is based on a divide & conquer scheme developed for the algorithm StreamLS for the k-median problem by Guha, Meyerson, Mishra, Motwani and O'Callaghan [GMM+03]. Their algorithm reads chunks of the data, computes a bicritera approximation for each chunk containing $\mathcal{O}(k)$ centers, and in the end computes an approximative solution for the union of all computed centers. As the bicriteria approximation, an extension of k-means++ is used, which continues to choose centers in the same way as k-means++ to obtain αk centers for a constant α. The authors show that these centers provide a constant approximation. For the final computation, k-means++ is used. Thus, the computed solution has a quality of $\mathcal{O}(\log k)$.

The algorithm does not use a polylogarithmic amount of space, but the space usage is $\mathcal{O}(k\sqrt{n})$ in the basic version and can be improved to $\mathcal{O}(kn^c)$ space for a constant c which influences the approximation guarantee. The authors implement the algorithm and provide experimental results but do not reference an implementation.

Independently and around the same time, Ackermann, Lammersen, Märtens, Raupach, Sohler and Swierkot [AMR+12] developed a k-means++ variant in the streaming setting based on coresets which is named StreamKM++. The coreset result is already mentioned above. It is obtained by using points successively sampled according to the k-means++ scheme in a similar way as in [AJM09].

Compared to [AJM09], the chunk sizes are chosen to be smaller to obtain a streaming algorithm with polylogarithmic storage usage. In addition to sample a subset of points, the points that are not sampled are assigned to their closest sample point, and the sample points are then weighted by the number of points assigned to them. The resulting set is shown to be a $(1+\varepsilon)$-coreset. The coreset construction is then embedded into Merge-and-Reduce.

At the end of the stream, an approximate solution is computed by running the seeding part of the k-means++ algorithm on the computed coreset. This step is repeated a constant number of times to obtain a more stable solution quality.

For the practical implementation, the authors heuristically decide to set the coreset size for each chunk to $m := 200k$. This means that the coreset property is no longer guaranteed, but the decision is backed up by experimental results showing a still high quality of the solution. To speed up the random sampling (which can take long because updating the probabilities has a running time of $\mathcal{O}(mdk)$), the algorithm is enhanced with a so called *coreset tree* which adds an additional heuristic element but leads to a significant speed-up. The running time of the implemented version of the algorithm is stated as $\Theta(ndk)$. The implementation is available at [ALM+10].

Practical streaming algorithms for related problems. There are a lot of streaming algorithms for related clustering problems and we shortly review why these are not directly applicable to the k-means problem in the Insertion-Only data stream model. CURE [GRS01] requires more than one pass over the data. DBSCAN [EKSX96] is not center based but computes desity based clusterings. CLARANS [NH02] is typically used when centers have to be chosen from the dataset and is not particularly optimized for points in the Euclidean space. ROCK [GRS00] and COBWEB [Fis87] are designed for categorical attributes.

5.3 A lower bound for BIRCH with fixed threshold

In Section 5.4, we develop BICO, an algorithm based on a similar data structure as the algorithm BIRCH, but which can be proven to compute a coreset. Before that, we investigate the worst-case behavior of BIRCH and prove a lower bound on the worst-case quality of its solution. For our analysis, the structure of the CF tree does not matter. We mainly care about a lower bound on the number of clustering features stored in the leaves of the tree. Therefore, we focus on describing the creation process of clustering features. Additionally, we want to ignore the rebuilding algorithm. We thus analyze a variant of BIRCH where an optimal threshold T is given to BIRCH in the beginning. In this context, optimal means that T is the smallest threshold among all thresholds that assure that BIRCH processes the input without needing rebuilding steps. Intuitively, this version of BIRCH should compute better solutions because it does not need to search for the correct T and can directly bild the CF tree without rebuilding and restructuring it. However, a solution computed after rebuilding the CF tree can better by chance, so we explicitly state that our lower bound holds for BIRCH *with fixed threshold*.

While it is not too surprising that such a lower bound can be found, we also learn a bit about the reasons for the worst-case behaviour. We have to go into a bit more detail about the decision process implemented in BIRCH.

BIRCH provides a set of predefined distance functions to determine the distance between two clustering features. Our lower bound works for the distance function D_4 defined in

[ZRL97a], also called 'variance increase distance' there. For a point $x \in \mathbb{R}^d$ and a clustering feature which represents a point set $P' \subset \mathbb{R}^d$, the distance according to D_4 is given by

$$\sum_{y \in P' \cup \{x\}} \left(y - \frac{\sum_{y' \in (P' \cup \{x\})} y'}{|P'|+1} \right)^2 - \sum_{y \in P'} \left(y - \frac{\sum_{y' \in P'} y'}{|P'|} \right)^2. \tag{5.1}$$

The influence of the distance function is limited to the choice of the clustering feature that a point is added to. In order to transfer the lower bound to other distance functions, it is sufficient to make sure that this assignment is the same.

The first clustering feature is created for the first point in the stream and then contains this single point. The following points are added one after the other. For a new point $x \in \mathbb{R}^d$, BIRCH first identifies the clustering feature which is closest to x according to D_4. Then it decides whether to let x be absorbed by this clustering feature or to open a new clustering feature for x. Assume the clustering feature holds the point set P'. Then x is added if

$$\sqrt{\frac{\sum_{y \in (P' \cup \{x\})} (y - \mu(P' \cup \{x\}))^2}{|P'|+1}}$$

is smaller than a given threshold T. Otherwise, a new clustering feature is opened (possibly causing splits and rebuilding steps).

BIRCH works with increasing thresholds when processing the input data. It starts with threshold $T = 0$ and then increases the threshold whenever the number of clustering features exceeds a given space bound. After increasing the threshold, the tree is reduced by a rebuilding algorithm. Notice however that points which were already merged cannot be separated again, so the result of the rebuilding algorithm is not equal to the tree that would result from running the algorithm with the higher threshold on the original input. We ignore these effects by assuming that BIRCH starts with an optimal threshold instead of $T = 0$, implying that no rebuilding steps are performed.

Difficult instances for BIRCH. BIRCH bases the decision whether to add a point to a clustering feature on the (square root of the) average 1-means cost of the represented subset. The problem with this is that it does not reflect the increase in the total cost of the subset. Figure 5.2 shows a point set that was generated with two rather close but clearly distinguishable clusters plus randomly added points serving as noise. The problem for BIRCH is that the distance between the two clusters is not much larger than the average distance between the points in the noise. We see that BIRCH merges the clusters together and thus later computes only one center for them while the second center is placed inside the noise.

Our lower bound example for the quality guarantee of BIRCH follows this intuition. In comparison to Figure 5.2, it is multi-dimensional and places the points deterministically in a structured way useful for our theoretical analysis. We first look at a two-dimensional variation to explain the structure of the example.

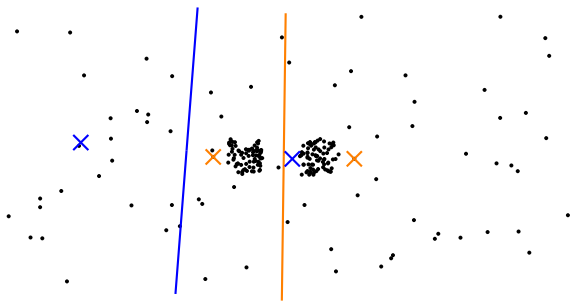


Figure 5.2: An example created by drawing 150 points uniformly at random from the areas around $(-0.5, 0)$ and $(0, 0.5)$ and 75 points from $[-4, -2] \times [4, 2]$. BIRCH computed the centers marked by blue crosses, leading to the partitioning by the blue line. BICO computed the same centers in 10 independent runs, marked by orange centers and the orange line.

Figure 5.3: A two-dimensional example. The blue and green circle each represent $(n-h)/2$ points at $(0, 0)$ and $(2, 0)$. There are h orange points at positions $(2i, \Delta)$ for $\Delta = 2\sqrt{nh^3}$ and $i \in \{0, h-1\}$. For $k = 3$, the optimum cost is $\mathcal{O}(h^3)$.

Consider Figure 5.3. There are two bunches of points at $(0,0)$ and $(1,0)$, and h orange points which are far away from the green and blue points. Their coordinates are $(2i, \Delta)$ for $i \in \{0, \ldots, h-1\}$. The distance $\Delta = 2\sqrt{n \cdot h^3}$ is large enough such that clustering an orange point together with a green or blue one yields a very high cost on its own.

We set $k = 3$. Then we can place one center on $(0,0)$ and $(1,0)$ each such that all the $n-h$ points in the green and blue cluster cost nothing. Placing the third center at $(0,\Delta)$ yields a cost of $\sum_{i=1}^{h-1} 4i^2 = \frac{4}{6}(h-1)\cdot h \cdot (2h-1) \leq \frac{4}{3}h^3$, so the optimal cost is less than $\frac{4}{3}h^3$. Notice that having an orange point in the same cluster as a point that is not orange yields a clustering cost of at least $(\sqrt{nh^3})^2 = nh^3$. So, for any $o(n)$-approximation this cannot happen.

If we spend only one center on the lower points, then the cost for the blue and green points is at least $(n-h)$ because the mean is at $(1,0)$ and there are $n-h$ points at distance 1 of this mean.

We set $h = n^{1/3(1-\varepsilon)}$, implying $h^3 = n^{1-\varepsilon}$. Then clustering the green and blue points together is at most a $\frac{3}{4}(n-h)/h^3 = \frac{3}{4}(n^{\varepsilon} - n^{-\frac{2}{3}(1-\varepsilon)})) = \Theta(n^{\varepsilon})$ approximation.

In order to force BIRCH to merge the green and blue points, we first place the orange points in the stream, from left to right. Then after the insertion of the ith orange point, each of the previous points has its own clustering feature as long as $T < 1$. This is because the distance between an orange point and any other orange point is at least 2, so merging two points into one clustering feature means that the merged clustering feature has a 1-means cost of 2. The radius of the merged clustering feature is thus $\sqrt{2/2} = 1$. So, either BIRCH needs $\Theta(h) = \Theta(n^{(1/3)})$ clustering features or $T \geq 1$ holds after reading the orange points.

After the orange points, we alternatingly append green and blue points to the stream. We assume that the first green point opens a new clustering feature because otherwise we have a $\Omega(n)$-approximation already. After one green and one blue point, we have a clustering feature containing both of them because their merged 1-means cost is again 2 and we have $T \geq 1$. The remaining green and blue points are also added to this clustering feature because after each point, the 1-means cost of the clustering feature is bounded by the number of points in the clustering feature. This is true because all points have distance 1 to $(1,0)$, and the optimal solution can only be better (when we have one green point more than blue points).

BIRCH thus either keeps $\Omega(n^{(1/3)(1-\varepsilon)})$ clustering features or induces an error of n^{ε}. We can set a convenient ε, for example $\varepsilon = 1/4$, and obtain a lower bound on the space of $\Omega(n^{1/4})$ or a lower bound on the approximation guarantee of $\Omega(n^{1/4})$, but we cannot get too close to $\Theta(n^{1/3})$ for convincing lower bounds on the approximation guarantee. To get a higher lower bound on the space usage of BIRCH, we use a multidimensional version of the example.

Notice that the 1-means cost of the orange points is the main factor that keeps us from obtaining a higher lower bound. We want to position the orange points such that they are cheaper but still have a pairwise distance of at least 2. So, we want to place h points (for an h not necessarily equal to the above setting) with pairwise distance of

at least 2 and minimal cost. Name this point set H. Recall from Lemma 2.4.5 that $\text{cost}_{\ell_2^2}(H, \mu(H)) = \frac{1}{2|H|}\sum_{x\in H}\sum_{y\in H}||x-y||^2$. So we get the lowest cost if all pairwise distances are exactly 2. This is achieved if we construct H as the set of vertices of a regular simplex. The drawback is that the dimension of H has to be $\omega(h)$ in order to embed a simplex with h vertices. We reduce the dimension by applying the Johnson-Lindenstrauss Lemma appropriately. It allows us to replace the vertices of the simplex by a $\Theta(\log h)$ dimensional set of points with approximately the same properties.

Lemma 5.3.1. *There exists a $\Theta(\log h)$-dimensional point set H consisting of h points such that the pairwise distances of all points lie in the intervall $[2, 6]$. Thus, the 1-means cost of H is $\Theta(h)$.*

Proof. First define $e_i' = \sqrt{2}e_i/(1-\varepsilon)$ for $i = 1, \ldots, h$ where e_i is the ith standard basis vector of $\mathbb{R}^h$, and define the set $H = \{e_1', \ldots, e_h'\} \subset \mathbb{R}^h$. Notice that $||e_i' - e_j'||^2 = \frac{2}{(1-\varepsilon)^2} + \frac{2}{(1-\varepsilon)^2} = \frac{4}{(1-\varepsilon)^2}$ for $i \neq j$. By Theorem 3.1.4, there exists a linear map $f : \mathbb{R}^h \to \mathbb{R}^{d_0}$ for $d_0 \in \Theta(\varepsilon^{-2}\log h)$ such that it holds for all $x, y \in H$ that

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

This implies that $||f(x)-f(y)||^2 \geq (1-\varepsilon)^2||x-y|| = 4$, and also that $||f(x)-f(y)||^2 \leq 4(1+\varepsilon)^2/(1-\varepsilon)^2$. The first statement of the lemma follows for $\varepsilon = (1/2)$, as $4\cdot(3/2)^2\cdot 2^2 = 36$. By Lemma 2.4.5, the 1-means cost of the projected points is $\frac{1}{2|H|}\sum_{x\in H}\sum_{y\in H}||f(x)-f(y)||^2$, which is at least $|H|-1$ and not more than $18|H|$. Thus, the cost is in $\Theta(h)$. □

Theorem 5.3.2. *There exists a stream of n points in $\mathbb{R}^{d_0+1}$ for $d_0 \in \Theta(\log n)$ such that for any $c \in (1, \infty)$, BIRCH with distance function D_4 needs at least $\Omega(n^{1-\frac{1}{c}})$ clustering features to process the point set or computes a summary with an $\Omega(n^{1/c})$-approximation guarantee.*

Proof. Let H be the d_0-dimensional point set from Lemma 5.3.1. We construct three point sets in $\mathbb{R}^{d_0+1}$. The point set B consists of $(n-h)/2$ points at the origin. The point set G consists of $(n-h)/2$ points at $(0, 1, 0, \ldots, 0, 0)$. For the point set O, we copy the points from H and append a $\Delta = 2\cdot\sqrt{nh}$ as the last coordinate. The distance between any point from O and any point from $B \cup G$ is at least Δ, so the 1-means cost of any cluster containing points from O and B or G is at least nh. We consider $k = 3$, so the optimal solution has a cost of at most $\Theta(h)$ because we can place one center at the origin, one center at $(0, 1, 0, \ldots, 0, 0)$, and place the remaining center optimally for O. Then the total clustering cost is $\Theta(h)$ by Lemma 5.3.1.

The stream starts with the points in O in arbitrary order. The first point opens the first clustering feature. Assume that $T < 1$ holds for the processing of all points in O. Then every point in O opens a new clustering feature. To see this, assume x would be the first point to be added to an existing clustering feature. This clustering feature consists of only one point which we name y. We know that $||x-y|| \geq 2$, so the radius of the merged clustering feature is 1. Thus, BIRCH either increases $T \geq 1$ or it opens h clustering features.

Assume that $T \geq 1$. After the points from O, we alternatingly put a point from B and G in the stream. We assume that the first point, which is from B, opens a new clustering feature because otherwise the approximation guarantee is already increased to $n \cdot h/h = \Omega(n)$. The next point is from G, and merging it with the point from B induces a clustering feature with radius 1. As $T \geq 1$, the two points are merged. We show that all following points are merged into the same clustering feature. Assume we have already seen ℓ points from B and ℓ points from G, and the $(\ell+1)$th point from B is about to be added. Clustering the $i+1$ points from B and the i points from G together costs at most $2i+1$ because we can still place the center at $(0, 1, 0, \ldots, 0, 0)$ (this is not optimal but leads an upper bound on the 1-means cost). So, the radius of the merged clustering feature is at most 1. A similar reasoning holds when we add the $i+1$ points from G.

The 1-means cost of $B \cup G$ is $n - h$, so the approximation guarantee of clustering them with one center is $\Omega((n-h)/h)$. So BIRCH either has an approximation guarantee of $\Omega((n-h)/h)$ or stores h clustering features. For $h = n^{1-1/c}$, this yields a lower bound on the storage capacity of $n^{1-1/c}$ or a lower bound on the approximation guarantee of $\Theta(n^{(1/c)})$. □

5.4 BICO – BIRCH meets coresets for k-means clustering

This section presents BICO, a streaming algorithm that computes coresets for the k-means problem in the Insertion-Only data stream model, its analysis and an experimental study that evaluates its speed, accuracy and memory usage (when combined with an approximation algorithm). BICO is joint work with Hendrik Fichtenberger, Marc Gillé, Chris Schwiegelshohn and Christian Sohler and was first published in [FGS+13].

The term BICO is a combination of the name BIRCH and the term *coreset*. This is because BICO builds upon BIRCH to inherit some of its practically desirable properties and combines it with insights from coreset theory to compute solutions of better quality.

One aspect of the velocity of BIRCH is that it uses very fast pointwise updates. Each input point is directly processed, and very little time is spent for each update. This is possible due to the use of clustering features, which provide a way to store large amounts of points with very little space, and which can absorb a new point in constant time. Recall that the clustering features of a point set form a 1-means coreset for the point set.

The weakness of BIRCH is that the decision which points are summarized by the same clustering feature is based on a criterion which allows arbitrary bad summaries. A clustering feature provides the exact cost of the subset of points represented by them as long as all the points are assigned to the same center. However, if an optimal assignment would split the points, and this is much cheaper than clustering them together, then the error of the summary is high. So improving the quality is about guessing which points will have the same center, or, more precisely, it is about partitioning the points such that clustering the points in the same subset together does not induce too much error.

BICO does this partitioning by using a different version of a tree with clustering features.

BICO from a theoretical point of view. During the course of this section, we will prove that BICO computes a $(1+\varepsilon)$-coreset, and that the size of this coreset is bounded by $\mathcal{O}(k \log n \varepsilon^{-(d+2)})$ if the dimension is constant. BICO works in the streaming setting without an additional framework like Merge-and-Reduce. The size of the computed coreset is exponential in the dimension. This is comparable to older results in coreset theory, like the streaming coreset by Har-Peled and Mazumdar [HPM04] or by Frahling and Sohler [FS05]. Notice however that for constant dimension, the size is actually competitive. To the author's knowledge, other known streaming coresets have a higher dependency on n, the number of points. All other results cited in Table 5.1 are only polylogarithmic in n, and the degree of the polynomial is at least four. Braverman et al. [BMO+11] develop a streaming algorithm for the Euclidean k-means problem that stores $\mathcal{O}(k\varepsilon^{-1} \log n)$ points, for arbitrary d, but assume that the input satisfies a separation condition. As we have seen in Chapter 3, there exist coresets for the k-means problem with a size that is independent of n. It is an interesting open problem if and how such a coreset can be computed in the streaming setting without additional assumptions on the input data.

BICO from a practical point of view. In Section 5.4.4, we see that BICO is well suited to be implemented and run in a practical setting. We make a few heuristic changes when implementing BICO, the most important one being that we run BICO with a parameter that defines the maximum number of clustering features rather than using the theoretically proven upper bound on the number of clustering features needed. Also, we combine BICO with k-means++ in order to compute actual solutions in addition to the coreset.

In the implemented form, BICO has the following advantageous properties. First, it computes solutions of a high quality, the best solutions for three quarters of our test cases, and at most 3 % away from the best solutions in the remaining cases. Second, it is faster than StreamKM++, which is the only other algorithm reliably computing solutions of good quality. Third, by adjusting the summary size BICO can be adjusted to run in similar time as the BIRCH while still computing better solutions (on the given data). We see this in more detail in Section 5.4.5.

5.4.1 The basic algorithm

The main idea underlying the development of BICO is to combine the data struture of BIRCH with the concepts and insights from coreset theory. BICO thus resembles BIRCH in its main structure: It uses a tree storing clustering features, and it processes points on the fly, adding them to an existing or new clustering feature, and it uses a rebuilding algorithm to compress the tree when the space bound is reached.

We first describe and analyze the insertion process without the rebuilding algorithm to explain the main ideas. The update mechanism is described in Algorithm 5.1 in the function `BICO-update(x,T)`. The BICO clustering feature tree stores clustering features in all nodes except the root node. The root node does not store input information, but for a more concise description we assume that it contains an artificial point ρ. The distance

Algorithm 5.1: Update mechanism where T is a fixed parameter

```
1  Function BICO-update(x,T)
2      Set r := ρ, F := children(cf(ρ)) and i := 1;
3      if F = ∅ or ||x − nearest(x, F)|| > √(T/2^{i+4}) then        // √(T/2^{i+4}) =: R_i
4          Open a new clustering feature (x, 1, x, ||x||^2, 0) on level i as child of cf(r);
5      else
6          Set y := nearest(x, F) ;          // y is associated with (y, n_y, P_y, s_y, c_y)
7          if c_y + ||x − y||^2 ≤ T then
8              Insert x into cf(y): Set n_y+=1, P_y+=x, s_y+=||x||^2 and c_y+=||x − y||^2;
9          else
10             Set F := children(cf(y));
11             Set r := y and i := i + 1;
12             Goto line 3;
13         end
14     end
15     if the number of current clustering features is higher than n_max then
16         Rebuild();
17     end
18 Function QueryCoreset()
19     S := ∅;
20     for all clustering features (r, n_r, P_r, s_r, c_r) in the BICO tree do
21         S := S ∪ {P_r/n_r};
22         w(P_r/n_r) := n_r;
23     end
24     return S and w;
```

between ρ and any other point is assumed to be larger than the distance between any pair of input points.

All other nodes store a clustering feature. When we open a new clustering feature, we keep the first point that is added as its *reference point*, which is stored in addition to the clustering feature. The reference point of the root is ρ. For a reference point r, we denote the corresponding clustering feature by $\mathrm{cf}(r)$. For each clustering feature $\mathrm{cf}(r)$, we store r, the reference point, n_r, the number of points represented by it, P_r, the sum of the points it represents, s_r, the sum of the squared lengths of the points it represents and c_r. The latter contains the clustering cost of the represented points with r as the center. This value could also be computed from n_r, P_r and s_r and is not strictly necessary.

In the following description, we do not distinguish between nodes and the clustering features stored in the nodes, i. e., we identify the nodes with their clustering features.

The first point in the stream opens a new clustering feature on level 1, as a child of $\mathrm{cf}(\rho)$. Subsequent points are added to a close clustering feature if possible. We define a value $R_i = \sqrt{T/2^{i+4}}$ that depends on T and on the level i and that we call *radius*. Points can not be added to a clustering feature if the Euclidean distance between the reference point and the input point is larger than R_i.

For a new point x, we start looking for an appropriate clustering feature on level 1 within the set F of child nodes of $\mathrm{cf}(\rho)$. We assume that the function $\mathrm{nearest}(x, F)$ returns (one of) the closest reference point(s) to x among the reference points of clustering features in F. If the distance between x and the returned reference point y is larger than R_i, or if there is no clustering feature on the current level, then x opens a new clustering feature on the current level, i. e., on level 1 during the first insertion try. Otherwise, we try to add x to $\mathrm{cf}(y)$. For this, we need c_y. We check whether $c_y + ||x - y||^2$ is larger than T. If not, then we add x by updating the clustering feature and c_y. Otherwise, we increase the level, replace F by the set of children of $\mathrm{cf}(y)$ and proceed in the same way within the new level until we either find a level where x can open a new clustering feature or we find a clustering feature where x can be added.

At any point in time, the method `QueryCoreset()` returns the current coreset of the points seen so far. The coreset consists of the centroids of the clustering features, weighted by the number of points that they represent.

Basic properties. The value c_y contains the clustering cost of $\mathrm{cf}(y)$ when y is used as the center. Thus, it provides an upper bound on the (optimal) 1-means cost of the clustering feature. Notice that we could compute the 1-means cost with the centroid and its increase exactly by using the clustering feature, but finding the closest reference point is easier than finding the clustering feature with the smallest increase in the cost. Additionally, it greatly simplifies the analysis to have a fixed reference point during the whole algorithm, and the centroid would constantly move. So, we are content with the upper bound given by clustering with the reference point.

Using (an upper bound on) the 1-means cost for the decision whether to add a point (in contrast to the average 1-means cost) is the most important difference between BICO and

BIRCH. However, we also introduce the radii and an additional insertion criterion which also plays an important role.

The idea behind the radius R_i is that it bounds the cost that one point contributes to a clustering feature. If a single point contributes too much error, then T is reached fast and we need a lot of nodes to store the points.

Notice that the radius R_i depends on T and on the level. More precisely, it exponentially decreases with the level number. The effect of this is that on lower levels, points cost less when added to a clustering feature. This means that more points are needed until T is reached, hereby increasing the capacity for points on lower levels. This implicitly bounds the number of levels that are created.

Observation 5.4.1. *If $R_i = \sqrt{T/(2^{i+4})}$ for any positive T, then the number of levels in our tree is bounded by $\log n$ where n is the number of points inserted so far.*

Proof. A new level is only created when a point should be inserted but this would increase the clustering cost of the clustering feature with the reference point as the center above the threshold T. The insertion of a point with a distance at most R_i to the reference point increases this cost by at most $(R_i)^2$. Therefore, we need at least $\lceil T/(R_i)^2 \rceil = 2^{i+4}$ points until a new level is created, and at least $2^{i+4} - 1 > 2^{i+3}$ points are in the parent clustering feature of the newly opened clustering feature. If there is a clustering feature on level $\ell \geq \lceil \log n \rceil - 3$ in the tree, then the number of points in its parent clustering feature is larger than $2^{\lceil \log n \rceil} \geq n$. This means that $\lceil \log n \rceil - 4 < \log n$ is an upper bound on the number of levels. □

Notice that we use the centroids for the actual coreset and not the reference points. We could use the reference points also. In fact, our analysis will prove the coreset property for replacing each clustering feature by the reference point (weighted by the number of represented points). However, by Lemma 2.4.1, when replacing a point set by a single point, then the centroid minimizes the error. Thus, using the centroids is the better choice.

Analysis. Applying Algorithm 5.1 to a stream containing the points of a point set P partitions P into subsets corresponding to the clustering features. Consider this partitioning and let $\mathcal{M}$ be the set of all subsets in it. By $\mathcal{M}_i$ we denote the subset of $\mathcal{M}$ belonging to clustering features on level i. The coreset consists of the centroids of the clustering features, each one representing one of the subsets of P stored in $\mathcal{M}$. We start by observing basic properties of this partitioning. In order to reuse the first part of this analysis later in Section 5.4.2, we formulate our first findings in a more general way by using the following definition.

Definition 5.4.2. *Let $\mathcal{P}$ be a partitioning of a point set P with n points. For each $P' \in \mathcal{P}$, let $r(P')$ be a distinguished reference point in P' and let $\ell(P') \in \{1, \ldots, \lfloor \log n \rfloor\}$ be a number assigned to P'. Let $\mathfrak{d}$ be a constant. We say that $(\mathcal{P}, r, \ell, \mathfrak{d})$ is a* well-structured T-partitioning *of P if*

- *for each subset* $P' \in \mathcal{P}$*, it holds that* $||x - r(P')|| \leq \mathfrak{d} \cdot R_{\ell(P')}$ *for all* $x \in P'$*,*
- *for all* $P', P'' \in \mathcal{P}$ *with* $\ell(P') = \ell(P'')$*, it holds that* $||r(P') - r(P'')|| > R_{\ell(P')}$*, and*
- *for all* $P' \in \mathcal{P}$*, it holds that* $\text{cost}_{\ell_2^2}(P', r(P')) \leq T$

for some values R_i*,* $i \in \{1, \ldots, \lfloor \log n \rfloor\}$*.*

We see that these conditions are in particular true for $\mathcal{M}$ and $\mathfrak{d} = 1$. Then $r(P')$ is the reference point of $P' \in \mathcal{M}$, $\ell(P')$ is the level of P' in the BICO tree and the R_i are as defined in Algorithm 5.1. We show that the weighted reference points of every well-structured T-partitioning satisfy the coreset property if T is below a certain value. The following resembles the coreset proof in [FS05].

Lemma 5.4.3. *Let* $P \subset \mathbb{R}^d$ *be a set of points and let* $(\mathcal{P}, r, \ell, \mathfrak{d})$ *be a well-structured* T*-partitioning of* P *for* $T \leq (\varepsilon^{d+2} \cdot \text{cost}_{\ell_2^2}(P))/(10^2 k(10(1+\mathfrak{d})\mathfrak{d})^d \log n)$ *and* $R_i = \sqrt{T/2^{i+4}}$ *and* $0 < \epsilon \leq 1$*,* $\mathfrak{d} \geq 1$*. Define a mapping* $\pi : P \to \mathbb{R}^d$ *in the following way. For all* $P' \in \mathcal{P}$ *and all* $x \in P'$*, set* $\pi(x) = r(P')$*. Then it holds that*

$$\sum_{P' \in \mathcal{P}} \sum_{x \in P'} ||x - \pi(x)||^2 \leq \frac{\varepsilon^2}{20} \cdot \text{cost}_{\ell_2^2}(P).$$

Proof. Let $C \in \mathbb{R}^d$ be an arbitrary but fixed set of k centers. For $x \in P$, let $c(x)$ be the center in C closest to x, ties broken arbitrarily.

We name the set of all subsets $P' \in \mathcal{P}$ with $\ell(P') = i$ by $\mathcal{P}_i$ and partition $\mathcal{P}_i$ into two subsets. $\mathcal{P}_i^{\text{far}} \subseteq \mathcal{P}_i$ contains all subsets that consists only of points $x \in \mathcal{P}_i$ for which the distance $||x - c(x)||$ is at least R_i/ε'. Here, we use the abbreviation $\varepsilon' = \varepsilon/(5 \cdot \mathfrak{d})$. We call points in subsets in $\mathcal{P}_i^{\text{far}}$ *far*. $\mathcal{P}_i^{\text{near}} \subseteq \mathcal{P}_i$ consists of all remaining subsets in $\mathcal{P}_i$, and we call the points in subsets in $\mathcal{P}_i^{\text{near}}$ *near*.

We now relate the clustering cost of points in $\mathcal{P}_i$ to the distance to their reference points. We do this independently for $\mathcal{P}_i^{\text{near}}$ and $\mathcal{P}_i^{\text{far}}$ and then combine our findings afterwards.

If a point $x \in \mathbb{R}^d$ is contained in a subset P' in $\mathcal{P}_i^{\text{far}}$, then its clustering cost $\text{dist}^2(x, C)$ is at least $(R_i/\varepsilon')^2$. Because of the precondition of the lemma we know that we have a well-structured T-partitioning, and thus it holds that the distance between x and the reference point $r(P')$ of P' is at most $\mathfrak{d} \cdot R_i$. Therefore, moving x to $r(P')$ induces a squared movement distance of at most $\mathfrak{d}^2 R_i^2 = \mathfrak{d}^2 \varepsilon'^2 R_i^2/\varepsilon'^2 \leq \mathfrak{d}^2 \varepsilon'^2 \cdot \text{dist}^2(x, C) = (\varepsilon/5)^2 \,\text{dist}^2(x, C)$.

For near points, the movement distance might be large compared to the actual clustering cost, as near points can even lie on a center and thus have no cost, while their distance to $r(P')$ is positive. We can still bound the sum of the squared movement distances of points in the same subset by T because points are only inserted if the 1-means cost with center $r(P')$ is not increased above T. We use a volume argument to bound the number of subsets in $\mathcal{P}_i^{\text{near}}$, which then bounds the total squared movement.

Each $P' \in \mathcal{P}_i^{\text{near}}$ contains at least one point $x \in P'$ that is contained within a ball of radius R_i/ε' around $c(x)$. The distance of x to $r(P')$ is at most $\mathfrak{d} \cdot R_i$. Thus, for all

$P' \in \mathcal{P}_i^{\text{near}}$, there is always at least one center such that the distance between $r(P')$ and the center is at most $R_i/\varepsilon' + \mathfrak{d}R_i$, and this implies that P' lies completely within a ball of radius $R_i/\varepsilon' + 2\mathfrak{d}R_i$ around this center. It holds that $2 \leq 1/\varepsilon'$ since $\varepsilon' = \varepsilon/(5\mathfrak{d}) \leq 1/5$ since $\varepsilon \leq 1$ and $\mathfrak{d} \geq 1$. Thus, P' lies within a sphere of radius $(1+\mathfrak{d})R_i/\varepsilon'$ around one of the centers.

We know from the precondition of the lemma that $||r(P') - r(P'')|| > R_i$ for two different $P', P'' \in \mathcal{P}$. Consequently, if we place balls with radius $R_i/2$ around all $r(P')$, then these balls do not intersect, which means that the number of subsets in $\mathcal{P}_i^{\text{near}}$ is bounded by the number of disjoint balls with radius $R_i/2$ that fit into k balls with radius $(1+\mathfrak{d})R_i/\varepsilon'$, i. e., by

$$k \cdot V^{(d)}((1+\mathfrak{d})R_i/\varepsilon')/V^{(d)}(R_i/2) = k \cdot \left(\frac{2(1+\mathfrak{d})}{\varepsilon'}\right)^d,$$

where $V^{(d)}(R) = \pi^{d/2} \cdot R^d/\Gamma(d/2+1)$ is the volume of a d-dimensional ball with radius R (here, Γ denotes the so-called Gamma function, an extension of the factorial function for real numbers). The squared movement cost of points in subsets in $\mathcal{P}_i^{\text{near}}$ is thus bounded by $T \cdot (2(1+\mathfrak{d})/\varepsilon')^d \cdot k$ for each i. As there are at most $\lceil \log n \rceil$ different i, the overall squared movement cost of near points is bounded by

$$\begin{aligned} T \cdot (2(1+\mathfrak{d})/\varepsilon')^d \cdot k \cdot \log n &\leq \frac{\varepsilon^{d+2}}{10^2 k (10(1+\mathfrak{d})\mathfrak{d})^d \log n} \text{cost}_{\ell_2^2}(P) \cdot (10(1+\mathfrak{d})\mathfrak{d}/\varepsilon)^d \cdot k \cdot \log n \\ &\leq (\varepsilon/10)^2 \cdot \text{cost}_{\ell_2^2}(P). \end{aligned}$$

Adding the contributions of near and far points we get that

$$\begin{aligned} &\sum_{P' \in \mathcal{P}_i^{\text{near}}} \sum_{x \in P'} ||x - \pi(x)||^2 + \sum_{P' \in \mathcal{P}_i^{\text{far}}} \sum_{x \in P'} ||x - \pi(x)||^2 \\ \leq &(\varepsilon/10)^2 \cdot \text{cost}_{\ell_2^2}(P) + (\varepsilon/5)^2 \cdot \text{cost}_{\ell_2^2}(P) \\ = &(\varepsilon^2/20) \cdot \text{cost}_{\ell_2^2}(P). \end{aligned}$$

□

Corollary 5.4.4. *Let $P \subset \mathbb{R}^d$ be a set of points given as a stream, let $0 < \varepsilon \leq 1$. Let S, weighted by w, be the summary computed by applying* `BICO-update(`x, T`)` *in Algorithm 5.1 with $T \leq (\varepsilon^{d+2} \cdot \text{cost}_{\ell_2^2}(P))/(10^2 k 20^d \log n)$ after each point x (never calling* `Rebuild()` *and assuming that $n_{\max}$ is sufficiently large). Then it holds for every set of k centers C from $\mathbb{R}^d$ that*

$$|\text{cost}_{\ell_2^2}(P, C) - \text{cost}_{\ell_2^2, w}(S, C)| \leq \varepsilon \cdot \text{cost}_{\ell_2^2}(P, C).$$

Proof. The partitioning $\mathcal{M}$ computed by Algorithm 5.1 satisfies the following conditions. Firstly, all reference points of clustering features on the same level have a pairwise distance of more than $R_i = \sqrt{T/2^{i+4}}$. Secondly, all points represented by a clustering feature have a distance of at most R_i to their reference point. Thirdly, the clustering cost of each clustering feature with its reference point is bounded by $T \leq (\varepsilon^{d+2} \cdot \text{cost}_{\ell_2^2}(P))/(10^2 \cdot 20^d \cdot k \cdot \log n)$. So,

$\mathcal{M}$ forms a well structured T-partitioning with $\mathfrak{d} = 1$. Also notice that $20^d = (10(1+1)\cdot 1)^d$. Together, this means that for the mapping $\pi : P \to \mathbb{R}^d$ which is defined by $\pi(x) = r(P')$ for all $P' \in \mathcal{M}$ and all $x \in P'$, we have

$$\sum_{P' \in \mathcal{M}} \sum_{x \in P'} ||x - \pi(x)||^2 \leq \frac{\varepsilon^2}{20} \cdot \text{cost}_{\ell_2^2}(P) \tag{5.2}$$

by Lemma 5.4.3. Notice that this holds for the reference points, but we need it for the centroids. However, the centroids minimize the sum of the squared distances of all points by Lemma 2.4.1, which means that Inequality (5.2) also holds when we define $\pi(x) = \mu(P')$ for all $x \in P'$ and all $P' \in \mathcal{M}$. This suffices to use Lemma 2.3.2 to obtain the coreset property for S and w (we set $\Lambda = \text{cost}_{\ell_2^2}(P)$, which is a lower bound for all $\text{cost}_{\ell_2^2}(P, C)$). □

We still need to bound the size of the computed coreset or the number of clustering features in the tree, respectively. The main observation that helps us is about *virtually full* clustering features. We say that a clustering feature $\text{cf}(y)$ is virtually full when the cost c_y is at least $T - (R_i)^2$. The idea is that only virtually full clustering features can refuse new points, because a point that lies within the radius of a clustering feature contributes at most $(R_i)^2$.

Now the observation is that clustering features are only opened for two reasons. Either they are opened because the inserted point lies within no clustering feature on the first level. For this we need to bound the number of clustering features on the first level. Or they are opened because another clustering feature is virtually full. So we can bound the number of clustering features that are not on the first level by bounding the number of virtually full clustering features. We see how to do this in the following lemma.

Lemma 5.4.5. *Let $P \in \mathbb{R}^d$ be a point set given as a stream and let the dimension d be a constant, let $0 < \varepsilon \leq 1$. Then there exists an $n_{\max} \in \mathcal{O}\left(k \cdot \log n \cdot \varepsilon^{-(d+2)}\right)$ such that applying `BICO-update(`x, T`)` in Algorithm 5.1 with $T \geq (\varepsilon^{d+2} \cdot \text{cost}_{\ell_2^2}(P))/(2 \cdot 10^2 k 20^d \log n)$ after each point x in the stream never exceeds $n_{\max}$ clustering features. In particular, `Rebuild()` is never called. The size of the coreset S, which is computed by the algorithm together with weights w, is also bounded by $n_{\max}$.*

Proof. Let n_f denote the number of virtually full clustering features. Notice that $(R_i)^2 = T/(2^{i+4}) \leq (1/32)T$ because we start with $i = 1$ on the first level. Thus, virtually full clustering features have a 1-means cost with the reference point of at least $(31/32)T$. We observe that this implies a bound on the number of virtually full clustering features because we have already bounded the sum of the costs in Lemma 5.4.4.

Recall that we denote the set of all subsets stored in the clustering features of our BICO tree after reading a point set P by $\mathcal{M}$. Then it holds that

$$n_f \cdot \frac{31}{32} T \leq \sum_{P' \in \mathcal{M}} \text{cost}_{\ell_2^2}(P', r(P')) \leq \frac{\varepsilon^2}{20} \text{cost}_{\ell_2^2}(P)$$

$$\Rightarrow n_f \leq \frac{8}{155} \cdot \varepsilon^2 \cdot \frac{\text{cost}_{\ell_2^2}(P')}{T} \leq \frac{1}{8} \varepsilon^2 \cdot k \cdot f(\varepsilon) \cdot \log n$$

where we use the abbreviation $f(\varepsilon) := (10^2 \cdot 20^d)/\varepsilon^{d+2}$.

We name the number of clustering features that are not virtually full by n_e and split it into the number of those that are on level 1, $n_{e,1}$, and the number of those on levels with higher index, $n_{e,+}$.

We can bound the number of clustering features that are not virtually full but also not on level 1 by observing that they are children of virtually full clustering features. A child clustering feature satisfies that the reference point lies within the radius of the parent clustering feature. Assume we count the number of child nodes of a clustering feature on level i. All of these have radius R_{i+1}, so their reference points have a distance of more than R_{i+1}. As in the proof of Lemma 5.4.3, we argue that this implies that balls with radius $R_{i+1}/2$ around the reference points of the child nodes do not intersect. All of these spheres lie completely within a sphere of radius $R_i + R_{i+1}/2$ around the reference point of the parent clustering feature.

Together, this implies that the number of child nodes of a clustering feature on level i is bounded by the number of distinct balls of radius $R_{i+1}/2$ that fit into a ball of radius $R_i + R_{i+1}/2$, which is bounded by

$$V(R_i + R_{i+1}/2)/V(R_{i+1}/2) = \left(\sqrt{\frac{T}{2^{i+4}}}/\sqrt{\frac{T}{2^{i+7}}} + 1\right)^d = (\sqrt{2} \cdot 2 + 1)^d.$$

We conclude that $n_{e,+} \leq (\sqrt{2} \cdot 2 + 1)^d \cdot n_f$. Then we estimate $n_{e,1}$ by the number of all clustering features on the first level. The reference points of all clustering features on the first level have a pairwise distance of at least $R_1 = \sqrt{T/32}$. We use this to give an upper bound on the clustering cost of the points on level 1. Notice that this clustering cost in particular includes the cost for clustering the reference points (and in the worst case for our coreset size, the clustering features that are not virtually full only contain their respective reference point).

Let Q be the set of reference points of all clustering features on the first level. Notice that $n_{e,1} \leq |Q|$. Assume that $Q_1, \ldots, Q_k$ is a partitioning of Q according to an optimal center set C^*. Thus, $\sum_{i=1}^k \text{cost}_{\ell_2^2}(Q_i, \mu(Q_i)) = \text{cost}_{\ell_2^2}(Q, C^*) \leq \text{cost}_{\ell_2^2}(P)$. By Lemma 2.4.5, the lower bound on the pairwise distances of the reference points implies

$$\begin{aligned} \text{cost}_{\ell_2^2}(Q, C^*) = \sum_{i=1}^k \text{cost}_{\ell_2^2}(Q_i, \mu(Q_i)) &\geq \sum_{i=1}^k \frac{1}{2|Q_i|}(|Q_i|^2 - |Q_i|)(\sqrt{T/32})^2 \\ &\geq \frac{|Q| - k}{64} \cdot T \\ \Rightarrow \quad n_{e,1} \leq |Q| \leq 72 \cdot \frac{\text{cost}_{\ell_2^2}(P)}{T} &= 72 \cdot k \cdot 2 \cdot f(\varepsilon) \cdot \log n. \end{aligned}$$

To see that the last line follows, notice that either $|Q| < 9k$ or $|Q| - k \geq (8/9)|Q|$. In both cases the statement follows.

It remains to add the three bounds to obtain that the number of clustering features and thus the number of points in the coreset is bounded by

$$\begin{aligned} n_f + n_{e,1} + n_{e,+} &\leq (1 + (\sqrt{2} \cdot 2 + 1)^d) n_f + n_{e,1} \\ &\leq (1 + (\sqrt{2} \cdot 2 + 1)^d) \cdot \frac{1}{8} \cdot \varepsilon^2 \cdot k \cdot f(\varepsilon) \cdot \log n + 72 \cdot k \cdot 2 \cdot f(\varepsilon) \cdot \log n \\ &< (145 + (\sqrt{2} \cdot 2 + 1)^d) \cdot k \cdot f(\varepsilon) \cdot \log n \in \mathcal{O}(k \cdot \log n \cdot \varepsilon^{-(d+2)}) \end{aligned}$$

for constant dimension d. Finally notice that the storage requirement for each clustering feature consists of storing two d-dimensional points and two constants. □

We obtain the following theorem by Lemma 5.4.4, Lemma 2.3.2 and Lemma 5.4.5.

Theorem 5.4.6. *Let P be a point set given as a stream in the Insertion-Only model, let $0 < \varepsilon \leq 1$ be given, let the dimension d be constant. There exists an $n_{\max} \in \mathcal{O}(k \cdot \log n \cdot \varepsilon^{-(d+2)})$ such that applying `BICO-update(`x, T`)` in Algorithm 5.1 with*

$$(\varepsilon^{d+2} \cdot \text{cost}_{\ell_2^2}(P))/(2 \cdot 10^2 k 20^d \log n) \leq T \leq (\varepsilon^{d+2} \cdot \text{cost}_{\ell_2^2}(P))/(10^2 k 20^d \log n)$$

after each point x in the stream causes no calls to `Rebuild()`, and such that the computed S and w constitute a $(1+\varepsilon)$-coreset. The size of the coreset and the storage requirement of Algorithm 5.1 during the computation are bounded by $\mathcal{O}(k \cdot \log n \cdot \varepsilon^{-(d+2)})$.

5.4.2 Including rebuilding steps into BICO

In Theorem 5.4.6, we assume that the algorithm is run with suitable parameter choices, such that the rebuilding never occurs. In this section, we deal with the problem that the optimum clustering cost and thus T might not be known, and that this might induce the need to compress the tree.

Algorithm 5.2 describes the rebuilding process. Whenever the space bound is exceeded in Algorithm 5.1 during `BICO-update(`x, T`)`, the function `Rebuild()` is called. It consists of doubling the threshold and then inserting all clustering features into a newly rebuilt tree as if they were points. Because of the higher threshold, this potentially creates a smaller tree. It does not create the same tree as if the actual points were inserted because points that have been merged together already cannot be split again.

The rebuilding works in a breadth first search manner. The algorithm uses a queue Q to keep track of the clustering features. It is initialized with all clustering features on level 1. These are 'deleted' from the old tree by deleting the connections to the root node (assuming that nodes have connections to their parent and child nodes).

As long as there are clustering features in the queue, the first one is extracted and processed. Before we add it into the new tree, we disconnect its child nodes and append them to the queue. Notice that the usage of a queue and the initalization by the first level implies that we process the clustering features by level. All clustering features from level i (in the old tree) are processed before the clustering features from level $i+1$ (from the

old tree) exit the queue. The rebuilding is completely executed before we check whether the number of clustering features is now small enough. It is repeated until the number is small enough.

The function `BICO-CF-Update`$(x, \mathrm{cf}(x), T)$ then handles the actual insertion process into the new tree. It is similar to `BICO-update`(x, T), but it inserts a clustering feature instead of a single point. It works along the same lines. First, it checks whether the clustering feature can just be put into the current level. The current level starts with level 1. This is the case if there is no other clustering feature, or if the nearest clustering feature is far away (i. e., if the distance between the reference points is at least R_i). Then $\mathrm{cf}(x)$ is just put into the current level. We keep the last node on the search path as in the original update algorithm so we can connect $\mathrm{cf}(x)$ when it is inserted to a level.

If there is a reference point y within the radius of x, there are again two cases. Either the clustering feature can absorb $\mathrm{cf}(x)$, or it cannot. In the latter case we proceed to search for an appropriate clustering feature among the child nodes, descending one level. In the first case, however, all points in $\mathrm{cf}(x)$ are added to $\mathrm{cf}(y)$. This case happens if the union of the two stored point sets can be clustered with center y with less than T cost. Notice that the clustering cost of the union is simply the sum of the clustering costs of the two subsets because the center y is fixed. The clustering cost of the points currently stored in $\mathrm{cf}(y)$, clustered with y, is stored in c_y. We can compute the additional cost by using the clustering feature associated with x. By Lemma 2.4.1, clustering $\mathrm{cf}(x)$ with the centroid costs $s_x - n_x \cdot ||P_x/n_x||^2$. Clustering with y instead costs additional $n_x||P_x/n_x - y||^2$. The sum of the two is c', the additional clustering cost if $\mathrm{cf}(x)$ is merged into $\mathrm{cf}(y)$. If $c_y + c'$ is still bounded by T, we do indeed insert $\mathrm{cf}(x)$ by updating the clustering feature of y appropriately.

Properties. The result of the rebuilding algorithm differs from the tree that would have been constructed by running BICO with the higher threshold in the first place. There are two main differences. First, clustering features cannot be split again, so we get a different partitioning. In particular, it is possible that a child not cannot be merged into its parent node even though the majority of its points would fit there. Second, merging clustering features undermines the property that points are within the radius of their reference points. The child nodes' reference points can lie on the border of the parents nodes' radius, and then the points in the child clustering feature are outside of the radius of the parent clustering feature. Merging them gives the parent clustering feature points outside of its radius. This effect can also accumulate over several levels. However, points cannot be arbitrarily far away. To see this, we first have to observe that clustering features can never decrease their radius.

We take a moment to comprehend what happens to the radii during the rebuilding. The threshold is doubled, which influences the radii. Let R'_i be the radius on level i before and R''_i be the radius after doubling. Then $R''_i = \sqrt{2T/2^{i+4}} = \sqrt{T/2^{i-1+4}} = R'_{i-1}$ holds (for $i > 1$). So, the radius that level i has after doubling is the radius that level $i-1$ had before doubling. If we would just add an empty top level and keep all clustering features in their

Algorithm 5.2: Rebuilding algorithm when number of CFs gets too large

```
1  Function Rebuild()
2      while number of clustering features is larger than n_max do
3          Set T := 2 · T;
4          Initialize a queue Q;
5          Disconnect the clustering features on level 1 from ρ and insert them into Q;
6          while Q ≠ ∅ do
7              Let x be the reference point of the first clustering feature in Q;
8              for all reference points s of clustering features in children(cf(x)) do
9                  Disconnect cf(s) from cf(x);
10                 Insert cf(s) into Q;
11             end
12             BICO-CF-Update (x, cf(x), T);
13             Delete cf(x) from Q;
14         end
15     end
16 Function BICO-CF-Update (x, cf(x), T)                    // x is associated with
17     Set r = ρ, F = children(cf(ρ)) and i = 1;                    // (x, n_x, P_x, s_x, c_x)
18     if F = ∅ or ||x − nearest(x, F)|| > R_i then
19         Install cf(x) on level i as a child of cf(r);
20     else
21         Set y := nearest(x, F) ;          // y is associated with (y, n_y, P_y, s_y, c_y)
22         Set c' := n_x(||P_x/n_x||^2 + ||y − P_x/n_x||^2);
23         if c_y + c' ≤ T then
24             Merge cf(x) into cf(y): Set n_y+=n_x, P_y+=P_x, s_y+=s_y and c_y+=c';
25         else
26             Set F := children(cf(y));
27             Set r := y and i := i + 1;
28             Goto line 18;
29         end
30     end
```

original positions (now one level lower), then all clustering features would just keep their radius. Instead, the rebuilding algorithm moves some of them up, filling the empty space, and merges some into their parent nodes. By this, they can only get larger radii.

Lemma 5.4.7. *Calling `Rebuild()` in Algorithm 5.2 never decreases the radius and does also not merge clustering features into clustering features with a smaller radius.*

Proof. The proof is an induction on the number of reinserted clustering features. The statement is true for the first clustering feature that is reinserted because ρ has no children at that point and the clustering feature is installed on level 1. The radius on level 1 is larger than the radius of every level before the rebuilding. Now the order of the reinsertion process is important. When a clustering feature is reinserted, only clustering features that were on a level with a smaller or the same index have been reinserted already. By the induction hypothesis, these are now on a level with the same or a larger radius. In particular, the level with the radius that the current clustering feature had before now holds a subset of the clustering features that were there before, and no new clustering features. Thus, the current clustering feature still fits here. It is either inserted on a level with a larger index, merged into a clustering feature on a level with larger radius, or is reinstalled on the level with the same radius. □

Now we can prove that the distance between any point and its reference point is at most $4R_i$. The main argument is that the sum of radii over several levels is a geometric series.

Lemma 5.4.8. *If the radii are defined as $R_i = \sqrt{T/(2^{i+4})}$, then for a clustering feature on level i, all points in the clustering feature lie within a distance of at most $\sum_{j=i}^{\infty} R_j < 4 \cdot R_i$ from the reference point.*

Proof. Let i^* be the number of the level with the smallest radius that ever occurs, i. e., clustering features on this level never have child nodes. We prove the following statement by induction. For any $i \in \{1, \dots, i^*\}$, it holds during the whole algorithm that points in clustering features with radius R_i have a distance of at most $\sum_{j=i}^{\infty} R_j$ to the reference point of the clustering feature.

As the base case we use level i^* because clustering features here cannot be the result of merging. A clustering feature can only be on level i^* if it was created on this level, and all points in the clustering feature have been added to it via the normal insertion process. Thus, they are within distance $R_{i^*} \leq \sum_{j=i^*}^{\infty} R_j$ from the reference point.

For points in a clustering feature on a level $i < i^*$, there are different options. Let the point x belong to a clustering feature with reference point r. If x was inserted into the clustering feature and not merged to it, then it was within the radius of the reference point when it was inserted. As the radii can only increase Lemma 5.4.7, it is still within the current radius $R_i < \sum_{j=i}^{\infty} R_j$ of r.

Otherwise, x ended up in the clustering feature due to a merge. At that point, the clustering feature which was merged into $\mathrm{cf}(r)$ had a smaller radius $R_{i'}$ than $\mathrm{cf}(r)$. As the radius of $\mathrm{cf}(r)$ can only have increased since then by Lemma 5.4.7, it holds $R_{i'} < R_i$. We

call the reference point of this other clustering feature r'. By the induction hypothesis, x was within distance $\sum_{j=i'}^{\infty} R_j$ of r' when it still belonged to $\mathrm{cf}(r')$. As r' must have been within the radius of r, which is unknown but smaller than R_i, the distance between x and r is bounded by $R_i + \sum_{j=i'}^{\infty} R_j \leq \sum_{j=i}^{\infty} R_j$.

To bound this term, we replace R_j by its definition and see that there is a geometric series which can be bounded by its limit.

$$\sum_{j=i}^{\infty} R_j = \sum_{j=i}^{\infty} \sqrt{\frac{T}{2^{j+4}}} = \sum_{j=i}^{\infty} \sqrt{\frac{T}{2^{i+4} \cdot 2^{j-i}}} = R_i \cdot \sum_{j=i}^{\infty} \sqrt{\frac{1}{2^{j-i}}}$$
$$= R_i \cdot \sum_{j=0}^{\infty} \left(\frac{1}{\sqrt{2}}\right)^j = (2+\sqrt{2}) \cdot R_i < 4 \cdot R_i.$$

□

Now we can proceed to the actual proof. First, we make sure that the number of levels in the tree is still bounded.

Observation 5.4.9. *For $R_i = \sqrt{T/(2^{i+4})}$ for any positive T, the number of levels is bounded by $\log n$, where n is the number of points inserted so far.*

Proof. Assume that a level with index $i^* + 1 = \lfloor \log n \rfloor + 1$ is created during the insertion of the n points and consider the first time that this happens. At this point in time, the clustering features on level i^* have never been merged, so the points in these clustering features are within distance R_{i^*} of their reference point. Each of these points thus induces a cost of at most $(R_{i^*})^2 = T/(2^{\lfloor \log n \rfloor + 4})$. When level $i^* + 1$ is created, there has to be a clustering feature on level i^* and a point within its radius that does not fit into the clustering feature. Thus, the cost of the clustering feature when absorbing this point is larger than T. However, as $T/R_{i^*}^2 = 2^{\lfloor \log n \rfloor + 4} \geq 8n$ holds, this is not possible with n points. Thus, level $\lfloor \log n \rfloor + 1$ cannot be created during the insertion of n points. □

We can now observe that the partitioning is still a well-structured P-partitioning (notice that we needed the bound on the levels for this, too). That yields the coreset property.

Corollary 5.4.10. *Let $P \subset \mathbb{R}^d$ be a set of points given as a stream, let $0 < \varepsilon \leq 1$. Let S, weighted by w, be the summary computed by applying* `BICO-update(`x, T`)` *in Algorithm 5.1 with $T \leq (\varepsilon^{d+2} \cdot \mathrm{cost}_{\ell_2^2}(P))/(10^2 k 10^d 20^d \log n)$ after each point x (possibly calling* `Rebuild()`*, but assuming that $n_{\max}$ is sufficiently large). Then it holds for every set of k centers C from $\mathbb{R}^d$ that*

$$|\mathrm{cost}_{\ell_2^2}(P, C) - \mathrm{cost}_{\ell_2^2, w}(S, C)| \leq \varepsilon \cdot \mathrm{cost}_{\ell_2^2}(P, C).$$

Proof. The statement follows from the fact that Algorithm 5.1 still computes a well-structured partitioning, only $\mathfrak{d}$ has to be increased to 4. The number of levels in the tree is bounded by $\log n$ by Observation 5.4.9. On each level i, the reference points have

a pairwise distance of more than $R_i = \sqrt{T/2^{i+4}}$. All points in the same clustering feature have a distance of at most $4R_i$ to their reference point by Lemma 5.4.8. Finally, the cost of the clustering features when clustering with the reference point is kept below T during update steps and rebuilding steps. Thus, for $T \leq (\varepsilon^{d+2} \cdot \text{cost}_{\ell_2^2}(P))/(10^2 k 10^d((1+4)4)^d \log n)$, Lemma 5.4.3 ensures that the movement costs to the reference points are small enough. In the same way as in Corollary 5.4.4 we argue that this also implies the coreset property for the set of centroids. □

The main part of the proof is to show that the number of clustering features is still bounded. This is challenged by the fact that clustering features cannot be split and that this induces a worse partitioning. However, we see in the following lemma that we can find pairs of clustering features that replace the virtually full clustering features from the original proof, so the unability to split clustering features mainly means that we have two nodes instead of one virtually full clustering node.

Lemma 5.4.11. *Let $P \in \mathbb{R}^d$ be a point set given as a stream, let $0 < \varepsilon \leq 1$ and let the dimension d be a constant. Then there exists an $n_{\max} \in \mathcal{O}\left(k \cdot \log n \cdot \varepsilon^{-(d+2)}\right)$ such that applying `BICO-update(`x, T`)` in Algorithm 5.1, possibly triggering calls of `Rebuild()`, produces a tree with at most $n_{\max}$ clustering features. The storage need of the algorithm is bounded by $\mathcal{O}(n_{\max})$. During this process, the threshold is not doubled if $T \geq (\varepsilon^{d+2} \cdot \text{cost}_{\ell_2^2}(P))/(2 \cdot 10^2 k 10^d 20^d \log n)$.*

Proof. In order to count the clustering features, we start by labeling them. First, we label all clustering features on level 1 that have no children as *base* clustering features and denote their number as n_b. Then, we label all remaining clustering features in the first level as *tuple parents*. For each tuple parent, we label one of its children as *tuple child*. Then, we iterate this process: As long as there are unlabeled clustering features with children, we mark one of them as tuple parent and then mark one of its children as *tuple child*. We denote the number of tuple parent and child pairs as n_t. When there are no more clustering features with children, we mark the remaining clustering features as *leaves* and denote their number as n_l.

The biggest challenge is to bound n_t. We know that a tuple child cannot be merged into its tuple parent because otherwise this would have happened during the last rebuilding step (if the child node was created in the time since the last rebuilding, then it cannot be merged since its reference point did not fit into the parent node). In this regard, tuple nodes are similar to virtually full clustering features in our original proof. However, the 1-means cost of each of the two nodes can be small as long as they are not merged. This means we cannot use the exact same argument.

Instead, we do the following. We define an alternative partitioning as a thought experiment. It cannot be created by the algorithm because the algorithm cannot split clustering features. For each tuple, we move points from the tuple child to the tuple parent until moving another point would increase the clustering cost of the tuple parent (with the reference point as center) above T. All other clustering features remain the same. Notice that the resulting tree is not necessarily a possible output of BICO because the tuple children

may have child nodes that could be merged into them now. However, we do not bother with this, because the resulting partitioning $\mathcal{P}'$ is still a well-structured T-partitioning: The level information is still valid. The reference points did not move, so their pairwise distance is still above R_i depending on their level i. The points are within $4R_i$ of their reference points by the same argumentation as in Lemma 5.4.8. The clustering cost of the clustering features is not increased above T. So, by Lemma 5.4.3, we have that

$$\sum_{P' \in \mathcal{P}'} \sum_{x \in P'} ||x - r(P')||^2 \leq \frac{\varepsilon^2}{20} \text{cost}_{\ell_2^2}(P).$$

Additionally, each tuple parent P' now satisfies $\sum_{x \in P'} ||x - r(P')||^2 \geq T - (4R_i)^2 \geq T/2 \geq \text{cost}_{\ell_2^2}(P)/(2 \cdot k \log n f(\varepsilon))$ by construction, because no further point could be added. Here, we use $f(\varepsilon) = 2 \cdot 10^{d+2} 20^d / \varepsilon^{d+2}$. The number of tuple parents is therefore restricted by $k \cdot \log n \cdot f(\varepsilon) \cdot \varepsilon^2/10$. As the number of tuples did not change by our modification of the partitioning, the original partitioning satisfies

$$n_t \leq k \cdot \log n \cdot f(\varepsilon) \cdot \varepsilon^2/10.$$

The number n_l of leaf clustering features is bounded by $(\sqrt{2} \cdot 2 + 1)^d n_t$ by the same argument as in Lemma 5.4.5 because we still demand that reference points lie within the radius of their parent reference point, and because leaf clustering features have to be the child of a tuple child (base clustering features have no child nodes). The number of base clustering features is bounded by the lower bound on the pairwise distances as we observed in the proof of Lemma 5.4.5. Here, the lower bound is $\sqrt{T/32}$, which means that set of reference points has a clustering cost of

$$\text{cost}_{\ell_2^2}(Q, C) \geq \frac{|Q| - k}{64} \cdot T$$

by using Lemma 2.4.5. Now, either $|Q| < 9k$ or $|Q| - k \geq (8/9)|Q|$. In both cases we get that

$$n_b \leq |Q| \leq 72 \cdot k \cdot f(\varepsilon) \cdot \log n.$$

Finally, we add the numbers for the three types of clustering features. Notice that each tuple contributes two clustering features to the total amount. The number of clustering features is bounded by

$$\begin{aligned} n_b + 2n_t + n_l &\leq n_b + (2 + (\sqrt{2} \cdot 2 + 1)^d) n_t \\ &\leq 72 \cdot k \cdot f(\varepsilon) \cdot \log n + (2 + (\sqrt{2} \cdot 2 + 1)^d) k \cdot \log n \cdot f(\varepsilon) \cdot \varepsilon^2/10 \\ &< (74 + (\sqrt{2} \cdot 2 + 1)^d) \cdot k \cdot f(\varepsilon) \cdot \log n \in \mathcal{O}(k \cdot \log n \cdot \varepsilon^{-(d+2)}). \end{aligned}$$

□

We conclude with the following theorem.

Theorem 5.4.12. *Let $P \in \mathbb{R}^d$ be a point set given as a stream in the Insertion-Only model, let $0 < \varepsilon \leq 1$, let $k \in \mathbb{N}$ and let the dimension d be constant. Let $T^* := (\varepsilon^{d+2} \cdot \mathrm{cost}_{\ell_2^2}(P))/(2 \cdot 10^2 k 10^d 20^d \log n)$. Then there exists a value $n_{\max} \in \mathcal{O}\left(k \cdot \log n \cdot \varepsilon^{-(d+2)}\right)$ such that the following holds. If the start value for the threshold is chosen such that $T \leq T^*$ and* `BICO-update(`x, T`)` *in Algorithm 5.1 is applied to each point x in the stream, then the threshold is not increased above T^* during the process. A $(1+\varepsilon)$-coreset S, weighted by a function w, is computed with a storage need of $\mathcal{O}(n_{\max})$, and the size $|S|$ is bounded by $n_{\max}$.*

Proof. By Lemma 5.4.11, the threshold is not doubled anymore as soon as it satisfies $T \geq T^*/2$, which will eventually happen while $T \leq T^*$. By Corollary 5.4.10, the fact that $T \leq T^*$ guarantees that the output will be a coreset. □

5.4.3 Running time

When trying to insert a point on level i, we need to decide whether the point is within distance R_i of its nearest neighbor, and if so, we need to locate the nearest neighbor within a set of candidates F. We denote the overall time spent on these nearest neighbor searches during the insertion process of a point into the BICO tree by $NT(m)$, where we use $m \in \mathcal{O}(k \cdot \log n \cdot \varepsilon^{-(d+2)})$ to abbreviate the coreset size. The other steps in the insertion process need constant running time except for the rebuilding. So the running time is bounded by $\mathcal{O}(NT(m) \cdot n)$ plus the time needed for the rebuilding steps.

A rebuilding step needs to go through all m elements of the coreset and to insert them into the new-build tree. This takes $\mathcal{O}(m \cdot NT(m))$ time. The number of rebuilding steps depends on how we choose the start value for T. We proved that BICO computes a (k, ε)-coreset for large enough m. In particular, this means that for any $m+1$ points, BICO contracts at least two of them during the process. We use this observation by scanning through the first $m+1$ points and calculating the minimal distance d_0 between two points (here, just ignore multiple points at the same position). Notice that if $T < d_0^2$, we are not able to merge any two points into one clustering feature. We set $T = d_0^2$. Then, T cannot be too small, because otherwise we cannot contract the m points.

The cost of any clustering is bounded from above by $n \cdot \Delta_{\max}^2$ where $\Delta_{\max}$ is the maximal distance between any two points. Our start value for T is bounded from below by the smallest distance $\Delta_{\min}$ between any two points. Thus, the factor between the start and end value of T is bounded by $n \cdot \frac{\Delta_{\max}}{\Delta_{\min}}$. The fraction $\Delta := \frac{\Delta_{\max}}{\Delta_{\min}}$ is the spread of the points. With each rebuilding step we double T, and thus the number of rebuilding steps is bounded by $\log(n \cdot \Delta)$.

Corollary 5.4.13. *Let $P \subset \mathbb{R}^d$ be a set of n points given as a stream, let $0 < \varepsilon \leq 1$. Applying the function* `BICO-update(`x, T`)` *in Algorithm 5.1 with the parameters from Theorem 5.4.12 after each point computes a $(1+\varepsilon)$-coreset in time bounded by $\mathcal{O}(NT(m)(n + m \log(n\Delta)))$ using $\mathcal{O}(m)$ space where $m \in \mathcal{O}(k \cdot \log n \cdot \varepsilon^{-(d+2)})$ is the coreset size and d is constant. $NT(m)$ is the total time spent on neighbor searches when inserting x (or $\mathrm{cf}(x)$, respectively).*

If we just search the candidate set F to find the nearest neighbor of the point (or clustering feature) that we want to insert, this takes $\mathcal{O}(|F|)$ time. In the worst case, we encounter all reference points during the insertion process of the point. That means that this simple implementation achieves $NT(m) = \Theta(m)$ in the worst case.

Of course the average time might be less if the tree is suitably balanced. Notice that the number of levels in the tree is always bounded by $\log n$. Balanced here means that the number of clustering features that we have to search on each level is small, ideally logarithmic in the number of clustering features stored in the tree. An example for an unbalanced tree would be a tree that has the maximum number of nodes on level 1, but only one of them has child nodes (as many as possible), and this pattern repeats for several levels. To improve the performance, we can add additional data structures to speed up the nearest neighbor queries in such cases.

The task we have to solve is: Store a set of reference points (with associated data) that have at least distance R_i to each other in a data structure such that this data structure determines for any given query point x whether there is a point within distance R_i of x in the data structure and, if so, returns the point. This data structure can be implemented in any node for the set of its child node, thus always speeding up the calls to `nearest`(x, F).

Notice the difference to a standard nearest neighbor query. If there is no point within the radius, we are not interested in the nearest neighbor. That makes the use of data structures for range queries intuitive. There is a huge number of results on geometric range data structures and nearest neighbor searches in general and we refer to [Sam05] for an overview.

In the following, we discuss how to apply different techniques. Let x be the point that we want to insert (or the reference point of the clustering feature that we want to insert).

Range Queries. As we are only interested in the nearest neighbor if it is within distance R_i of x, an orthogonal range query of a box of side length $2R_i$ centered at x is sufficient to get all candidates for our nearest neighbor. We have argued before that the points on level i have a pairwise distance of more than R_i and that spheres with radius $R_i/2$ around them do not intersect. All these non intersecting spheres lie within a box of with side length $2R_i + R_i/2$ which has a volume of $(5/2)^d R_i^d$. That means that the number of candidates is bounded by $(5/2)^d R_i^d / V^{(d)}(R_i/2) = (5/2)^d R_i^d 2^d \Gamma(d/2+1)/(\pi^{d/2} R_i^d) = O(1)$ for constant d.

Notice that we need a dynamic data structure because we have to update it after each new point. Static range query data structures can be made dynamic by using standard techniques [BS80, Meh84, Ove83] while losing a logarithmic factor in the query time. See [CT92] and Section 5.3 in [AE99] for an overview on dynamic range data structures.

The logarithmic overhead is not necessarily needed as in the case of the augmented dynamic range tree in [MN90]. For a set with ℓ points, it can be built in time $\mathcal{O}(\ell \log^{d-1} \ell)$, has space requirement $\mathcal{O}(\ell \log^{d-1} \ell)$, supports insertion of a point in time $\mathcal{O}(\log^{d-1} \ell)$ and a range query in time $\mathcal{O}(\log^{d-1} \ell + r)$ where r is the number of reported points.

When we add this data structure to every node (including the virtual root node) to organize the child node's reference points, then finding the nearest reference point for

a point x can be done in time $\mathcal{O}(\log^{d-1} m) + \mathcal{O}(1)$ for constant dimension d. As we may have to search on every level, the time spent on neighbor searches is bounded by $\mathcal{O}(\log n \log^{d-1} m)$. We need to construct the data structure $\mathcal{O}(\log(n\Delta))$ times, once at the beginning and once at the beginning of every rebuilding phase, so the total construction time is $\mathcal{O}(m \log^{d-1} m \log(n\Delta))$. This is dominated by the search times. Thus, BICO runs in time $\mathcal{O}(\log n \log^{d-1} m(n + m \cdot \log(n\Delta))$.

The additional data structures require $\mathcal{O}(\ell \log^{d-1} \ell) = \mathcal{O}(\ell \log^{d-1} m)$ space in every node, where ℓ is the number of children. As every node appears at most once as a child, the space requirement is bounded by $\mathcal{O}(m \log^{d-1} m)$.

Corollary 5.4.14. *Let $P \subset \mathbb{R}^d$ be a set of n points given as a stream, let $0 < \varepsilon \leq 1$. By adding an appropriate range query data structure, it is possible to implement BICO such that it processes P in time $\mathcal{O}(\log n \log^{d-1} m(n + m \cdot \log(n\Delta))$ while using $\mathcal{O}(m \log^{d-1} m)$ space, where $m \in \mathcal{O}(k \cdot \log n \cdot \varepsilon^{-(d+2)})$ is the coreset size and d is constant.*

Hashing. As we do not actually want to perform range queries but are only interested in specific ranges, we can store points even more conveniently. Imagine a grid with cell width R_i. Let $x = (x_1, \ldots, x_d)^t$ be a point. We calculate the vector $g(x) := (\lfloor x_1/R_i \rfloor, \ldots, \lfloor x_j/R_i \rfloor, \ldots, \lfloor x_d/R_i \rfloor)$. This maps the point onto one of the corners of its grid cell.

The nearest neighbor of x is either in the grid cell associated with $g(x)$ or in one of its $3^d - 1 \in \mathcal{O}(1)$ neighbor cells. Each of these cells contains $\mathcal{O}(1)$ points by the same argumentation as in the range query paragraph. Thus, when looking for the reference point which is within radius R_i on level i, we can restrict the search to these $\mathcal{O}(1)$ candidates.

In order to find them, we need a data structure. Storing a point set for every grid point takes too much space. Instead, we can use hashing.

One option to use hashing is by using the classic result by Carter and Wegman [CW79] on universal families of hash functions which includes a family of hash functions for vectors. This leads to a constant search and insertion time, but both are only on expectation. Instead, we have a look at perfect hashing, which achieves a constant worst-case lookup time. This is beneficial since most of our queries are searches.

As we need to use it in a streaming setting, we need dynamic hashing methods. Using dynamic perfect hashing [DKM+94] or cuckoo hashing [PR04], it is possible to maintain hash maps with constant worst-case search time and constant expected amortized insertion time. We decide for cuckoo hashing.

We use one hashing data structure for every level. It has to process all queries between two rebuilding steps. Notice that there are $m + 1$ insertions between two rebuilding steps. All other points only query to find the correct reference point and they do not induce a change in the hashing data structure.

The first thing we need to do is to transform our points into integers which is the expected input of cuckoo hashing. Notice that the point coordinates are integers between $-\Delta$ and Δ, so there are $(2\Delta + 1)^d$ possible points. We can map them to $\{1, \ldots, (2\Delta + 1)^d\}$ in time

$\mathcal{O}(d)$ with a bipartite mapping[5] which we call r. For every point x, we compute $g(x)$ and then hash $r(g(x))$.

Cuckoo hashing needs a $(1, m^\delta)$-universal family of hash functions[6] which can be constructed according to a scheme by Siegel [Sie89]. The construction of this hash family needs $\mathcal{O}(m^\delta)$ space and time for a constant $0 < \delta \leq 1$.

At the start of the algorithm and at the beginning of the reinsertion in each rebuilding step, we initialize two tables of size $2m$ on each level and we choose two hash functions h_{i1} and h_{i2} to set up the cuckoo hash structure. The hash functions need space $o(m)$. The tables may be refilled and the hash functions may be replaced during the phase, but the running time of this is covered by the expected insertion time of cuckoo hashing.

When performing `nearest(`x, F`)`, we compute $h_{i1}(g(x))$ and $h_{i2}(g(x))$ and the hash values of the surrounding cells. The search returns lists with all points in $g(x)$ and the neighbor cells. We look trough all lists and report the point r closest to x that is stored in these lists. If the distance between r and x is larger than R_i, x opens a new CF and is added as its reference point to the hash data structure.

By using cuckoo hashing, we need constant worst-case time to look up each of the $\mathcal{O}(1)$ grid points, and as we have to proceed through at most $\log n$ levels, the time spent on neighbor searches when inserting a point x into the BICO tree is $\mathcal{O}(\log n)$. Insertions have expected amortized constant time, which means that they induce an expected running time of $\mathcal{O}(m)$ for any phase between two of the at most $\mathcal{O}(\log n\Delta)$ rebuildings. The space of BICO is not affected as the hash data structure needs linear size.

Corollary 5.4.15. *Let $P \subset \mathbb{R}^d$ be a set of n points given as a stream, let $0 < \varepsilon \leq 1$. By adding an appropriate hashing based data structure, it is possible to adjust BICO such that the time to process P is bounded by $\mathcal{O}(\log n(n + m\log(n\Delta)))$ plus expected $\mathcal{O}(m\log(n\Delta))$ time, using $\mathcal{O}(m)$ space, where $m \in \mathcal{O}(k\cdot\log n\cdot\varepsilon^{-(d+2)})$ is the coreset size and d is constant.*

Filtering. We refer to the term filtering as a generic term for reducing the number of possible candidates for our nearest neighbor. A filter is a necessary but not sufficient condition that the nearest neighbor we search has to satisfy. We use them as a heuristic method to reduce the number of candidates. The methods above can be seen as elaborate filters: The nearest neighbor has to lie within the box of side length $2R_i$, and it has to be within one of the surrounding grid cells. These actually come with a guarantee on the number of points. However, they are rather complicated.

[5] We describe one way to do this. For a grid point x, we determine its rank by the following procedure. Let a be the highest absolute value of any coordinate in x. Then there are $(a+1)^d$ points which have coordinates that have a smaller absolute value than a. Among the points that contain coordinates with absolute value a, we determine the position of x by transforming x into a ternary vector first. We replace every coordinate with lower absolute value by 0, every coordinate that is a by 1 and every coordinate that is $-a$ by 2. Now we just interpret the result as a ternary number z. The rank of x then is $(a+1)^d + z$.

[6] A family of hash functions from $\mathcal{U}$ to $\{0, \ldots, m-1\}$ is (c, k)-universal if it satisfies $\text{Prob}(h(x_1) = y_1, h(x_2) = y_2, \ldots, h(x_k) = y_k) \leq \frac{c}{m^k}$ for all $x_1, \ldots, x_k \in \mathcal{U}$ and all $y_1, \ldots, y_k \in \{0, \ldots, m-1\}$.

A very easy example of a filter is the following. If r is the nearest neighbor of x and within distance R_i, then there is no coordinate in which the two differ by more than R_i. So, what we could do is to choose a random coordinate and filter the candidate points according to it. We can also use a precomputed data structure. Assume that we project every input point to a one-dimensional subspace (which is a bit more general as only projecting to axis-aligned subspaces) and store it in a data structure according to the length of the projection. If r and x have at most distance R_i, then the length of their projections can also only differ by R_i. Thus, we maintain a table (of sufficient size) where each entry holds a list of all points that have their projection length in the same R_i-interval. Thus, only points in the same or the two neighboring intervals are candidates for the nearest neighbor we search.

Important in the design of a filter is that it does not cause too much overhead. The benefit of filters is heuristic. If a filter does not reduce the number of candidates, then it should at least not induce too much additional running time. For example, computing the projection of a point takes $\Theta(d)$ time. This is fine, since computing the distance between the query point and any point takes $\Theta(d)$ time in itself.

Of course we can also combine filters: We can maintain multiple such data structures, project a new point to all assigned subspaces, take the intersection of all lists that we get and finally compare the resulting points with the query point in only a randomly chosen subset of the coordinates. The final list is then the candidate list where we search for the nearest neighbor. Of course, combining filters normally introduces additional overhead.

5.4.4 Implementation

Now we describe our implementation of BICO. When implementing algorithms with a theoretical guarantee, it is often necessary or beneficial to make slight changes to the algorithm. This is also the case for BICO. We adapted BICO with the following principles in mind.

Only perform small changes. We believe that the theoretical performance guarantees indicate that the algorithm has important properties that are instrumental for a good performance. So the general goal is to change as little as possible.

Enable potential for better performance. Heuristics profit from the fact that they perform well on most data even though there exist bad worst-case examples (that usually do not occur). It is important to make sure that the implemented algorithm has the potential to perform well on average data. For example, we think that smaller coreset sizes suffice. Thus, we should not enforce BICO to use the theoretically computed coreset size because that would mean that we force it to use too much memory and too much running time when it might not be necessary.

Keep worst-case running times and avoid overhead. Even if an additional data structure promises a huge change for better practical performance, we only want to use it if it does not increase the worst-case running time. The running time of BICO is low enough to guarantee moderate computation times even for huge data sets, and we do not want to jeopardize this for speed-ups that might fail on some data sets. Also, we want to

avoid complicated data structures that induce a high (constant) amount of overhead in terms of lines of code and additional memory requirement because we wish to keep BICO competitive on smaller instances. Having these guidelines in mind, we implement BICO with the following changes.

1. The upper bound on the coreset size that we proved is a worst-case guarantee that holds for any input, regardless how unlikely it is to actually occur. We use an adjustable summary size, i. e., the user can choose the size. We recommend a size of $m = 200k$. This was inspired by [AMR+12] as StreamKM++ is run with the same coreset size, and it performed well in our experiments (see Section 5.4.5). Notice that this change is heuristic and means that the summary is not necessarily a $(1+\varepsilon)$-coreset.

2. We add a filter to improve the nearest neighbor searches on the first level. This is a compromise between speeding up the neighbor search and causing a lot of overhead by additional data structures. The first level is particularly endangered to have a large number of clustering features as long as the threshold is rather small. We describe the data structure in more detail below.

3. We slightly change the rebuilding. Instead of reinserting all features, we only reinsert on the first level, keeping the connections between the nodes and their child nodes. Then we go through the tree and merge child nodes into their parent nodes if possible.

The filter is simple and uses p projections to 1-dimensional subspaces. Each projection is represented by a unit vector. At the beginning of the algorithm and at the start of each rebuilding step, we draw p unit vectors randomly[7]. Let x be the point that we want to find the nearest neighbor of and let u be the unit vector of one of the projections. We compute the projection of x onto the line spanned by u by computing the scalar product $x^t u$. During the insertion process, we keep track of the smallest and the highest value that occurs. This interval is divided into buckets of size R_1. For each bucket, we maintain a list of the points that are projected into the subinterval represented by this bucket. So when searching for x, we only need to look through the reference points in the same bucket and in the two neighbor buckets. We heuristically decide to only search the bucket that x falls into. To increase the chances that this list is small, we take the projection where the list in the corresponding bucket is the shortest.

Notice that this data structure potentially creates a lot of buckets, and we did not implement improvements to compress it (like avoiding to store empty buckets). Also, we decided to use $p = d$ projections. In our experiments, no problems occured because of this. The behavior was improved in a newer version of BICO as described in the paragraph on the source code below.

[7] We draw each coordinate uniformly at random from $[0, 1]$ and then normalize the vector. This does not draw a point uniformly at random from the unit sphere, but it is fast and worked well enough.

From coresets to solutions. BICO computes a summary of the input point set, not a solution. In order to compute an actual center set, we combine it with the k-means++ algorithm by Arthur and Vassilvitskii [AV07]. We already described this algorithm on page 13 as one of the fundamental algorithms for k-means clustering. It is known for its practicability combined with a $\mathcal{O}(\log k)$-approximation guarantee on expectation. Notice that combining the original BICO algorithm with k-means++ thus yields an expected $\mathcal{O}(\log k)$-approximation. We implemented k-means++ in a straightforward manner without specialized data structures or heuristics to speed it up.

Source code. BICO and k-means++ are implemented in C++. The source code for the algorithms, the testing environment and links to the other algorithms' source codes are available at `http://ls2-www.cs.uni-dortmund.de/bico/`. We continue to update BICO and will provide updated source code on the website. We recommend to always use the latest version of BICO. At the time of this thesis, that is BICO 1.1, but newer versions will probably appear. The improvements in BICO 1.1 compared to BICO 1.0 are:

- We corrected an error that could prevent the second level from being considered in the rebuilding.
- The initialization phase was corrected because the previous version did not consider enough points.
- We installed an upper bound on the number of buckets such that BICO is more stable when presented with data sets that have a high spread.
- The new code detects if a user chooses a summary size that is larger than the input size and in that case outputs the input.
- Multiple instances of BICO 1.1 can be created and be used in parallel.

5.4.5 Experimental setting

In this section, we describe the experiments that we conducted. The next section discusses the results of these experiments.

BICO. We used BICO 1.0 for our experiments which is the version that was current in 2013. Notice the above listed changes in BICO 1.1. On our test data, the third and fourth change to BICO do not play a role because the spread of the data set was not large enough to cause problems, and we did not use summary sizes less than the input size. The first two improvements may improve BICO's performance over the here presented results. We plan to do further tests with the updated source code in future work. As mentioned above, BICO and k-means++ are implemented in C++. For the experiments in this section, they were compiled with gcc 4.5.2. The experiments were performed with a summary size of $m = 200k$ for BICO and a limit of five iterations of k-means when performing k-means++ on the coreset.

Reference algorithms. We compare BICO with several algorithms. The first algorithm is BIRCH [ZRL97a], which is very relevant since we based BICO on BIRCH, and since BIRCH is a renowned algorithm for k-means clustering in the streaming setting. We described BIRCH on page 100 and in more detail in Section 5.3. The second algorithm is StreamKM++ [AMR+12]. It is the same type of algorithm as BICO, a coreset algorithm for the k-means problem that is implemented in a slightly heuristic manner in order to increase the practical performance. We shortly discussed StreamKM++ on page 101. We included MacQueen's k-means algorithm because it peformed surprisingly well in preliminary tests on one of our data sets (see the experiments on CalTech128 in the next section). The algorithm was also part of the discussion of related work and is described on page 100.

StreamLS [GMM+03, OMM+02] was not specifically developed for the k-means problem but for the k-median problem. We only included it into the experiments because it was part of the experimental study in [AMR+12]. It had a rather high running time which is why we did not include it in the experiments for the largest data sets.

We use the authors' implementations for StreamKM++, BIRCH and StreamLS. MacQueen's algorithm is from the sixties and no implementation by the author was available. We thus used an open source implementation that is part of the open source framework ESMERALDA [FP11].

Parameters. We ran StreamKM++ with a summary size of $200k$ like done by its authors and as we also chose for BICO. BIRCH has a long list of parameters. The authors recommend a set of parameters. As in [AMR+12], we use this set except for the memory settings, allowing BIRCH to store significant percentages of the data. Compared to [AMR+12], we increase the memory to 26 % on BigCross and 8% on Census in order to enable BIRCH to compute solutions for our new test cases with larger k. The parameters are documented in Table B.1 in the appendix. There might be better parameters and the results presented here are only valid for these parameters.

Datasets. The authors of StreamKM++ conducted a considerable experimental study in [AMR+12]. We decided to use the largest data sets from their study, plus an additional set with higher dimension. The evaluated data sets are *Tower*, *Covertype* and *Census* from the UCI Machine Learning Repository [AN07] and *BigCross*, which is a subset of the Cartesian product of *Tower* and *Covertype*, created by the authors of [AMR+12] to have a very large data set. Our additional data set consists of 128 SIFT descriptors [Low04] computed on the Caltech101 object database and was provided by René Grzeszick. We call this data set CalTech128 because the 128 descriptors imply 128 dimensions. Table 5.3 gives an overview on the sizes of the data sets. BigCross has the highest number of points and the largest total size while CalTech128 has the highest dimension. Tower is the smallest data set and has only three dimensions, while CoverType has the smallest number of points which is around half a million.

	BigCross	CalTech128	Census	CoverType	Tower
Number of Points (n)	11.620.300	3.168.383	2.458.285	581.012	4.915.200
Dimension (d)	57	128	68	55	3
Total size ($n \cdot d$)	662.357.100	405.553.024	167.163.380	31.955.660	14.745.600

Table 5.3: Data set sizes

Test cases. As a basis, we used the test cases that were performed in [AMR+12] for the data sets that we chose. For Census and CoverType, this is $k = 10, 20, 30, 40, 50$, for Tower it is $k = 20, 40, 60, 80, 100$, and for BigCross it is $k = 15, 20, 25, 30$. Then, we added the test cases $k = 100, 250, 1000$ for all except for CoverType (because of the small number of points). For BigCross we also added $k = 50$ for the sake of consistency. On CalTech128, we tested $k = 50, 100, 250$ and $k = 1000$.

We repeated all randomized algorithms 100 times. Notice that BICO is not randomized in its original version, but the heuristic speed-up is randomized, and k-means++ is randomized as well. StreamKM++ also uses k-means++ and is thus also randomized. StreamLS is also a randomized algorithm. MacQueen's k-means algorithm is deterministic, as is BIRCH.

Test environment. All computations were performed on seven identical machines with the same hardware configuration (2.8 Ghz Intel E7400 with 3 MB L2 Cache and 8 GB main memory).

5.4.6 Experiments

We did not succeed in searching for parameters that enabled BIRCH to process the CalTech128 instance. We tried a lower dimensional version which worked and we got the impression that the implementation cannot handle the high dimensionality of CalTech128.

Running Times. Figure 5.4 visualizes the running times. The left column shows the test cases that were also considered in [AMR+12], the right column (including the second diagram in the last line) shows most of the new test cases. The running time of BICO is depicted as a block consisting of two blocks which correspond to the time that is spent on BICO itself and on the execution of k-means++ in the end.

For randomized algorithms, the diagrams show the mean value of 100 runs. All mean values of all test cases are also listed in Table B.2 in the appendix. Additionally, Table B.3 and Table B.5 show the median of the running times and the coefficients of variance for all randomized algorithms on all test cases.

Notice that StreamLS is not included in the diagrams but in the tables. Its running time is quite high and would have scaled the diagrams. Without StreamLS, StreamKM++ dominates the diagrams. It is expected that it has a higher running time than the other

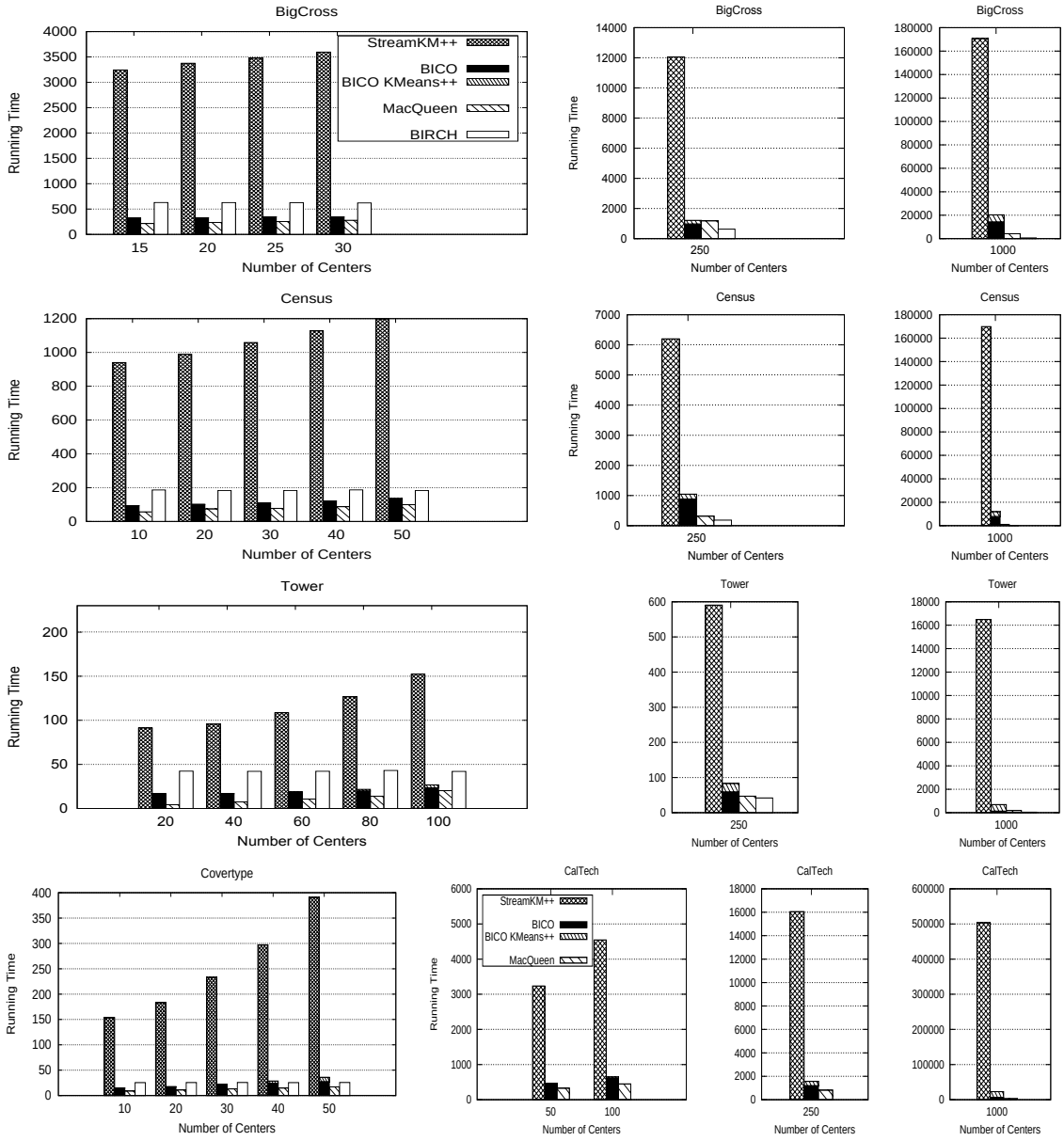


Figure 5.4: Diagrams visualizing the (average) running time for most test cases. Randomized algorithms were performed 100 times.

reference algorithms since it has a higher focus on accuracy than the other algorithms except for BICO. BICO, also having a high priority on good quality solutions, is considerably faster than StreamKM++. The factor is at least five in all test cases and increases for test cases with larger k. On CalTech128 and $k = 1000$, BICO is twenty times faster than StreamKM++, which means an improvement from nearly six days of computation time to less than seven hours.

The comparison with BIRCH and MacQueen's k-means algorithm is less clear. For test cases with small k (up to $k = 50$), MacQueen's algorithm is the fastest, followed in nearly all test cases by BICO, then BIRCH. Covertype is the only instance where BIRCH is faster than BICO for small values of k, namely for $k = 40$ and $k = 50$. Notice that for Covertype, these values are larger compared to the total number of points than for the other inputs. In most test cases for small k, BICO runs in twice the running time of MacQueen's algorithm, the exception being the smallest Tower test cases.

For test cases with larger k, BICO is slower than MacQueen's algorithm and BIRCH, the latter now being the fastest by far. Notice that BIRCH was performed in all test cases with $k = 1000$ except for CalTech128, but it is nearly invisible in the diagrams because of its low running time.

We notice some additional characteristics about the results. Except for BIRCH, all algorithms' running times increase with k. That makes sense for the two coreset based algorithms as they maintain a summary of $m = 200k$ points and in both cases, inserting points has a running time that depends on m. The fact that m depends on k is a reasonable and expected property because to keep the same accuracy for larger values of k, we need to have a larger summary ($\Omega(k)$ is a natural lower bound for the size of a coreset – a point set where n/k points have the same coordinates can only be preserved with to arbitrary precision if at least one point from each cluster is stored). Even just keeping a solution implies a storage usage of at least k points. What is unclear is whether the running time (and not only the memory usage) has to depend on k (as much). We see that compared to StreamKM++, BICO increases slower with regard to k. Even the lean algorithm by MacQueen increases with k because it computes the distance between any input point and the k centers in the current solution, but it shows the slowest ascent which is of course the reason why BICO becomes slower than MacQueen for larger values of k.

The running time of BIRCH is basically constant when varying k. This means that no algorithm with a running time that depends on k can ever beat BIRCH (with this parameters) with regard to the running time for every value of k. At some point, however, the centers will outnumber the stored clustering features, and then BIRCH has to guess the remaining centers or use more storage and probably more running time.

As a final note regarding the running times, notice that the running time of k-means++ makes up a significant portion of the total running time of BICO for some instances, for example for $k = 250$ and $k = 1000$ on CalTech128. This could be improved by using a more advanced implementation of k-means++ which handles high dimensions better. However, there are test cases where the core part of BICO dominates, for example for all BigCross experiments and all test cases with low k.

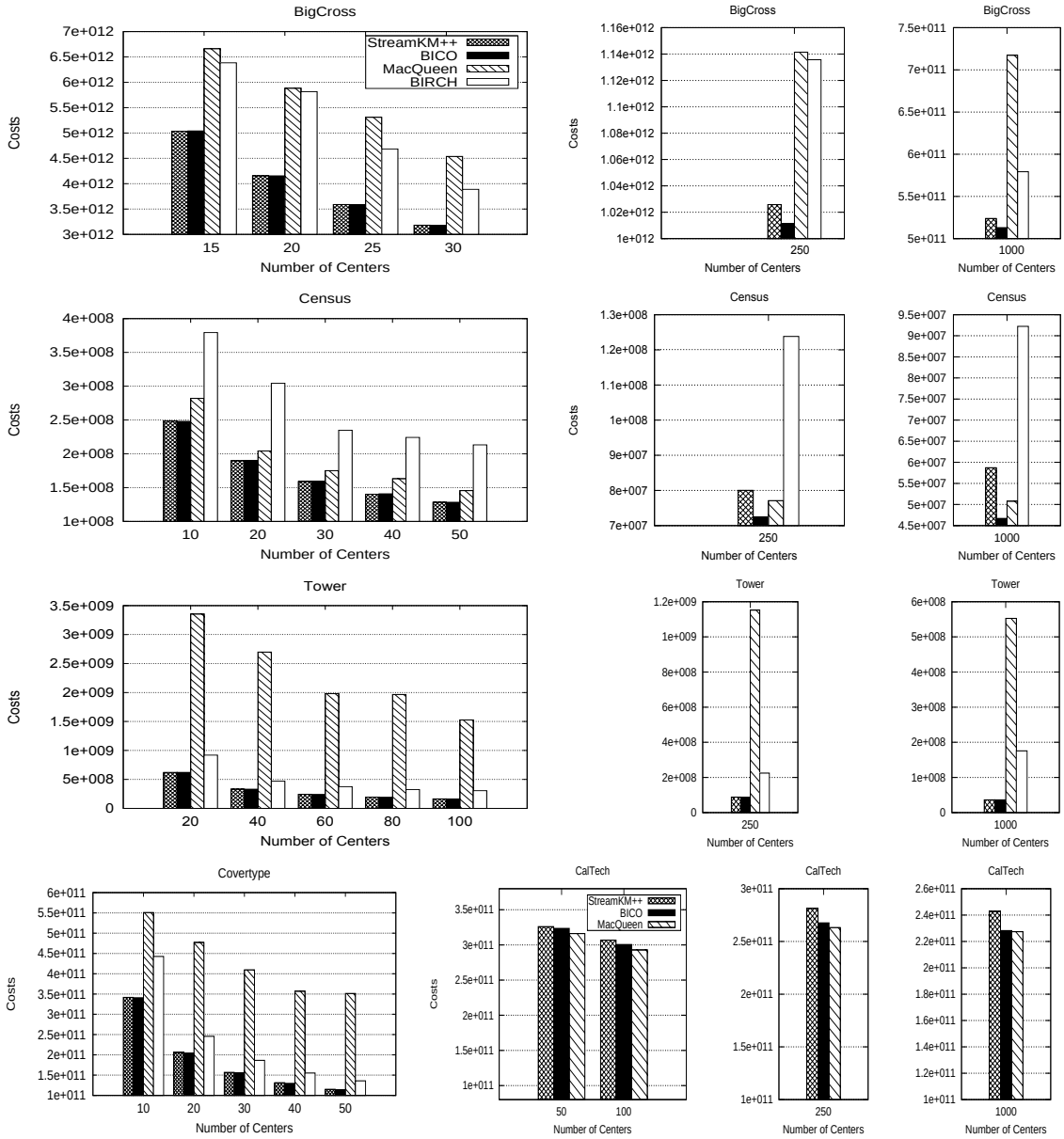


Figure 5.5: Diagrams visualizing the (average) costs of the computed solutions for most test cases. Randomized algorithms were performed 100 times.

Accuracy. The diagrams depicting the (average values of the) costs of the solutions are contained in Figure 5.5, placed in the same order as in Figure 5.4. Table B.5 in the appendix contains all costs. Again, the values are average values for the randomized algorithms. Table B.7 and Table B.8 contain the corresponding medians and the coefficients of variance for all randomized algorithms.

As expected, the coreset based algorithms show the best performance with respect to the quality of the solutions. On 71,8% of the test cases, BICO computed the solution with the highest quality (the lowest cost). Notice that both StreamKM++ and BICO use k-means++ to compute the solution. While this introduces randomness, it should not favor one of them, implying that the different performance is maybe due to a better coreset (but this is speculation). However, for most of the test cases, the cost of the solution computed by StreamKM++ is at most 1%-2% higher than the cost of the solution computed by BICO. Notably, the exceptions where the solution of StreamKM++ is more than 2% more expensive than BICO's solution are all for test cases with larger k, starting with $k = 100$ for CalTech128 and being worse for $k = 250$ and $k = 1000$ for Census.

StreamKM++ computes the best solution for five of the test cases, which is 15,6% of the cases. The solution by BICO is always at most 0,5% more expensive than StreamKM++'s solution. For the remaining four test cases, the algorithm by MacQueen surprisingly outperforms the other algorithms. Notably, this happens for all CalTech128 test cases. We included MacQueen into our study because it was used for CalTech128 by the group that provided the data set. Apparently, it is indeed well suited for this data set, for different numbers of centers. The solution computed by BICO is at most 3% more expensive than the solution computed by MacQueen's algorithm. The solution quality of MacQueen's algorithm might depend on the ordering of the points, but we did not test this.

With the exception of Caltech128, where MacQueen's algorithm performs so surprisingly well and BIRCH did not work such that we have no comparison, both algorithms computed solutions of higher cost than BICO and StreamKM++. The solution by BIRCH is always more than 10% worse than the solutions by BICO and StreamKM++, and in half of the test cases, it is more than 30% off. For Tower with $k = 250$, it is even 2.5 times as high as the solutions by BICO and StreamKM++. The solutions computed by MacQueen's algorithm are of an extremely varying quality, and this seems to be consistent for each data set. For CalTech128, the solutions by MacQueen's algorithm are excellent. For Census, the cost is around 5%-15% higher than the solution by BICO. For BigCross, the cost is 30%-40% worse than for BICO. Finally, on Covertype and Tower, the solution is up to ten times worse than BICO's solution. This seems to slightly favor larger data sets, however, by that logic BigCross should have the least error. The results might change if the points are reordered.

Memory usage. Figure 5.6 shows the memory usage of the algorithms. As before, the values for the randomized algorithms are the average values of 100 runs. All numerical values for the (average) running times are contained in Table B.6 in the appendix. Table B.9 and Table B.10 contain the median values and the variation coefficients for all randomized

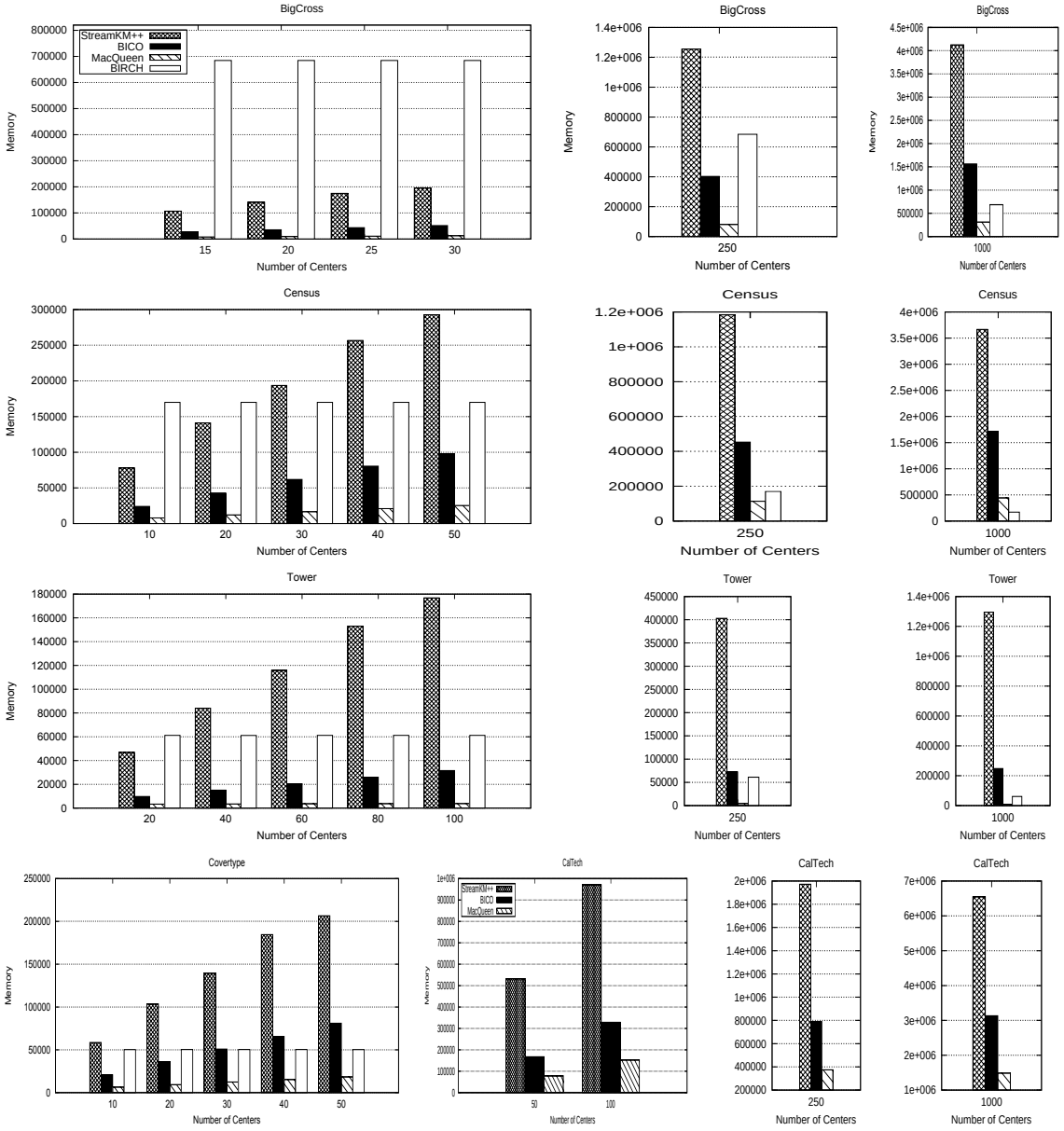


Figure 5.6: Diagrams visualizing the (average) memory usage for most test cases. Randomized algorithms were performed 100 times.

	BIRCH	MacQueen	BICO, $200k$	BICO, $100k$	BICO, $50k$	BICO, $25k$
Time	616.74	4241.23	20271.97	5444.75	1670.26	**618.94**
Costs	$5.79 \cdot 10^{11}$	$7.17 \cdot 10^{11}$	$5.12 \cdot 10^{11}$	$5.23 \cdot 10^{11}$	$5.35 \cdot 10^{11}$	$5.58 \cdot 10^{11}$

Figure 5.7: Tests with decreasing coreset sizes on BigCross with $k = 1000$. Notice that BICO is less than 1.5 times slower than MacQueen with $m = 100k$ while still computing reasonable costs where MacQueen computes costs that are very high. The running time is comparable with BIRCH at $m = 25k$, while the cost of the solution is still much better for BICO. The first three columns are from our original experiments, the remaining are additional experiments performed with 10 runs.

algorithms.

MacQueen's algorithm needs very little memory as it only stores the centers chosen so far and the implementation apparently has little overhead. It is the most memory efficient algorithm of the tested algorithms. By our parameter settings, we allow BIRCH to use a relatively large amount of memory, so we cannot blame the algorithm for having the highest memory usage for the BigCross test instances. Most of the time, BIRCH does actually not use nearly as much memory as we allow it to do by our settings.

Notice that BICO and StreamKM++ both maintain a summary of size $200k$. So both algorithms have to store at least $200k$ points of dimension d, where d depends on the data set. Of course, both algorithms also have overhead, like the tree structure or the nearest neighbor data structure in BICO. However, when averaging the maximum memory usage of (core) BICO by dividing it by $200kd$, the value always lies between $125 - 200$ Bytes for all data sets except for Tower. This is relatively stable and indicates that the memory usage of BICO grows predictably with the summary size. For StreamKM++, the memory usage differs more. The result of dividing the maximum memory usage by $200kd$ lies in the range $250 - 650$ Bytes for the same data sets. The memory usage divided by $200kd$ of StreamKM++ gets smaller for larger values of k, which indicates that the algorithm has a larger constant overhead. It makes the correlation with the summary size less clear.

For Tower, the calculation gives worse values for both algorithms, which is probably due to the fact that for a low dimensional and comparably small input like Tower, the constant overhead for data structures plays a more dominant role.

The memory usage of StreamKM++ does not only vary more than BICO's, it is also many times higher (for all data sets).

Using BICO under resource constraints. BICO computes solutions of high quality at a shorter time than StreamKM++. Yet, it may always be that the available time or memory is not enough to perform BICO as described here, for example for very large values of k. We wish to point out that it is still possible to use BICO under stricter resource constraints by sacrificing a bit of the solution quality.

To demonstrate this more specifically, we performed a small additional experiment. For this we chose the test case $k = 1000$ on BigCross because it is a large data set where BICO has an unfavorable running time of several hours, and because this running time is due to the core BICO algorithm. On Caltech128, a large fraction of the running time is caused by the execution of k-means++.

We decreased the summary size in several steps and observed how the quality changes. Notice that decreasing the summary size decreases the running time as most of the update time of BICO is spent by searching neighbors within the summary. Consider Table 5.7. In the original experiments, BIRCH needs 616 seconds to process BigCross with $k = 1000$, and MacQueen' algorithm needs 4241 seconds. BICO needs an average time of 20271 seconds when executed with a summary with $200k$ points. For $100k$ points, the time goes down to 5444 seconds, which is already comparable to MacQueen's running time. Finally, by halving the size two more times, we get to a summary size of only $25k$. BICO now runs in 619 seconds (on average over 10 runs), which is only 3 seconds worse than BIRCH. Yet, the quality of the solution is still significantly better than the quality of the solutions by BIRCH and MacQueen's algorithm (which computes a notably worse solution for this data set).

Discussion. We did not discuss StreamLS in detail. It is the slowest of the tested algorithms and both its running time and solution quality fluctuate more (see the variation coefficients in Table B.4 and B.8). Additionally, it does not necessarily compute the correct number of centers (by design). MacQueen's algorithm can compute very good solutions, but not in a reliable way. However, it is always fast and needs little memory.

BICO and StreamKM++ compute solutions of comparable quality. However, BICO is faster and needs less memory. The results regarding BIRCH should be taken with care because of the large number of parameters. Yet, the experiments at least indicate that it is difficult to compute solutions of reliably low cost with BIRCH.

In a way, BICO is a descendent of both BIRCH and StreamKM++. Even though BICO differs from BIRCH significantly, it uses the general structure and the idea to have a tree of clustering features. BICO and StreamKM++ do not have much in common regarding their actual courses of actions, but they share the idea to base an implementation on an algorithm with a proven quality guarantee.

BICO combines the advantages of the two approaches, providing accurate solutions within short running time by using a lean data structure. The adjustable summary size as the only main parameter makes it easy to apply BICO to different needs regarding accuracy, memory usage and running time. We believe that when investigating unknown data sets, BICO is the best choice to compute a k-means clustering. Additionally, we expect that BICO might be even faster with a different nearest neighbor data structure. This is a topic of further research.

Part II

Extensions of classical clustering

6 Kernel k-means problems

In this chapter, we see an extension of the coreset construction in Section 4.3 to more general settings. The results in Section 4.3 have appeared in [FSS13], and the insights in Section 6.1 follow directly. The results in Section 6.2 are based on Section 4.3 and Section 6.1 and have not yet been published. They are also joint work with Dan Feldman and Christian Sohler.

In Section 6.1 we notice that the coreset construction does not only work for squared Euclidean distances but for any distance measure where the distance between x and y is the inner product of $x-y$ and $x-y$. This observation is interesting because it allows us to compute coresets for the kernel k-means problem, which we define and study in Section 6.2.

The results in this chapter are joint work with Dan Feldman and Christian Sohler.

6.1 The k-means problem in inner product spaces

Let $\mathbf{V}$ be an inner product space over a field $\mathbf{F}$ with an inner product $\langle\cdot\rangle : \mathbf{V}\times\mathbf{V}\to\mathbf{F}$. Recall that we can define a norm $|\cdot| : \mathbf{V}\to\mathbb{R}$ by defining $|\chi| := \sqrt{\langle x,x\rangle}$. Additionally, $\mathrm{dist}(\chi,y) := |\chi - y|$ forms a metric. We use it to define a variant of the k-means problem in inner product spaces.

Definition 6.1.1. *Let $\boldsymbol{V}$ be an inner product space with the norm $|\cdot| : \boldsymbol{V}\to\mathbb{R}$ which is based on an inner product on V. Let $\mathcal{P}\subset\boldsymbol{V}$ be a finite set of vectors from $\boldsymbol{V}$. The* k-means problem on $\mathbf{V}$ *is to find a set $\mathcal{C}$ of k vectors from $\boldsymbol{V}$ that minimizes*

$$\mathrm{cost}_{ip^2}(\mathcal{P},\mathcal{C}) := \sum_{\chi\in\mathcal{P}}\min_{c\in\mathcal{C}}|\chi-c|^2$$

over all choices of $\mathcal{C}$.

We denote the optimal cost of clustering $\mathcal{P}$ with k centers by $\mathrm{cost}^{*,k}_{ip^2}(\mathcal{P})$. Notice that by Definition A.3.1 it holds for all $\chi,y\in\mathbf{V}$ that

$$\begin{aligned}
\mathrm{dist}^2(\chi,y) = |\chi-y|^2 = \langle\chi-y,\chi-y\rangle &\overset{1.}{=} \langle\chi,\chi-y\rangle + \langle -y,\chi-y\rangle\\
&\overset{4.}{=} \langle\chi,\chi\rangle + \langle\chi,-y\rangle + \langle -y,\chi\rangle + \langle -y,-y\rangle\\
&\overset{1.}{=} \langle\chi,\chi\rangle + \langle\chi,-y\rangle - \langle y,\chi\rangle - \langle y,-y\rangle\\
&\overset{5.}{=} \langle\chi,\chi\rangle - \langle\chi,y\rangle - \langle y,\chi\rangle + \langle y,y\rangle\\
&= |\chi|^2 - \langle\chi,y\rangle - \langle y,\chi\rangle + |y|^2. \qquad (6.1)
\end{aligned}$$

It follows similarly that $|\mathcal{x}+y|^2 = |\mathcal{x}|^2 + \langle \mathcal{x}, y\rangle + \langle y, \mathcal{x}\rangle + |y|^2$ for all $\mathcal{x}, y \in \mathbf{V}$. Furthermore, Equation (6.1) also implies that it holds for all $\mathcal{x}, y, z \in \mathbf{V}$ that

$$\begin{aligned}\operatorname{dist}^2(\mathcal{x}, y) = |\mathcal{x} - y|^2 &= |\mathcal{x} - z + z - y|^2 \\ &= \operatorname{dist}^2(\mathcal{x}, z) - \langle \mathcal{x} - z, z - y\rangle - \langle z - y, \mathcal{x} - z\rangle + \operatorname{dist}^2(z, y). \quad (6.2)\end{aligned}$$

Recall Lemma 2.4.1 from Chapter 2 which allowed us to split the k-means cost quite conveniently during our studies of the k-means problem. It says that the 1-means cost of a point set with n points and a fixed center can be calculated by computing the optimal 1-means cost (clustering with the centroid) and adding n times the distance between the fixed center and the centroid. In particular, it also implies that the centroid is always the optimal 1-means solution. Luckily, this still holds in inner product spaces.

Lemma 6.1.2. *Let $\boldsymbol{V}$ be a vector space and let $\mathcal{P} \subset \boldsymbol{V}$ be a set of vectors. Let $\mu := \sum_{\mathcal{x}\in\mathcal{P}} \mathcal{x}/|\mathcal{P}|$ be its centroid. Then, every vector $z \in \boldsymbol{V}$ satisfies*

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

Proof. We want to compute $\sum_{x\in\mathcal{P}} |\mathcal{x} - z|^2$ for $\mathcal{x}, z \in \mathbf{V}$. By Equation 6.2, we notice that $|\mathcal{x} - z|^2 = \operatorname{dist}^2(\mathcal{x}, \mu) - \langle \mathcal{x} - \mu, \mu - z\rangle - \langle \mu - z, \mathcal{x} - \mu\rangle + \operatorname{dist}^2(\mu, z)$. Summing over all points in $\mathcal{P}$, we already have that

$$\sum_{\mathcal{x}\in\mathcal{P}} |\mathcal{x} - z|^2 = \sum_{\mathcal{x}\in\mathcal{P}} |\mathcal{x} - \mu|^2 - \sum_{\mathcal{x}\in\mathcal{P}} \langle \mathcal{x} - \mu, \mu - z\rangle - \sum_{\mathcal{x}\in\mathcal{P}} \langle \mu - z, \mathcal{x} - \mu\rangle + |\mathcal{P}| \cdot |z - \mu|^2.$$

Now we notice that

$$\begin{aligned}\sum_{\mathcal{x}\in\mathcal{P}} \langle \mathcal{x} - \mu, \mu - z\rangle = \left\langle \sum_{\mathcal{x}\in\mathcal{P}} \mathcal{x} - \mu, \mu - z \right\rangle &= \left\langle \sum_{\mathcal{x}\in\mathcal{P}} \Big[\mathcal{x} - \sum_{y\in\mathcal{P}} y/|\mathcal{P}|\Big], \mu - z \right\rangle \\ &= \left\langle \sum_{\mathcal{x}\in\mathcal{P}} \mathcal{x} - \sum_{y\in\mathcal{P}} y, \mu - z \right\rangle = \langle \mathbf{0}, \mu - z\rangle = 0.\end{aligned}$$

For the last equality notice that the inner product of the neutral vector $\mathbf{0}$ with any vector is zero because of the linearity of the inner product. The same idea works for the second summand, which proves the lemma. □

The other important tool that we used frequently is the movement lemma that allows us to move points while bounding the change in the cost function. We formulate it for inner product spaces in the following lemma.

Lemma 6.1.3. *Let $\mathcal{P} \subset \boldsymbol{V}$ and $\mathcal{Q} \subset \boldsymbol{V}$ be finite sets of the same cardinality from an inner product space $\boldsymbol{V}$. Let $0 < \varepsilon < 1$ be a given constant and let $\pi : \mathcal{P} \to \mathcal{Q}$ be a bijection that satisfies*

$$\sum_{\mathcal{x}\in\mathcal{P}} |\mathcal{x} - \pi(\mathcal{x})|^2 \leq \frac{\varepsilon^2}{16} \operatorname{cost}^{*,k}_{ip^2}(\mathcal{P}).$$

Then it holds for every set $\mathcal{C} \subset \boldsymbol{V}$ of k with $|\mathcal{C}| \leq k$ that

$$|\operatorname{cost}_{ip^2}(\mathcal{P}, \mathcal{C}) - \operatorname{cost}_{ip^2}(\mathcal{Q}, \mathcal{C})| \leq \varepsilon \cdot \operatorname{cost}_{ip^2}(\mathcal{P}, \mathcal{C}).$$

Proof. We follow the proof of Lemma 2.3.2. Let C be given. We define an n-dimensional vector a that contains the distances between the points and their closest center, i. e., we number the points arbitrarily, $\mathcal{P} = \{x_1, \ldots, x_n\}$, and define $a_i := \min_{c \in \mathcal{C}} |x_i - c|$. Notice that $a \in \mathbb{R}^n$ is a real vector, and that the cost of $\mathcal{P}$ with center set $\mathcal{C}$ is the squared *Euclidean* length of this vector, $\operatorname{cost}_{ip^2}(\mathcal{P}, \mathcal{C}) = ||a||^2$.

Next define $a'_i := |x_i - \pi(x_i)|$ as the distance between x_i and its image under π. Notice that by the triangle inequality, which holds for the norm in $\mathbf{V}$, it holds that for any $c \in \mathcal{C}$ that

$$|\pi(x_i) - c| = |\pi(x_i) - x_i + x_i - c| \leq |\pi(x_i) - x_i| + |x_i - c|.$$

In particular, if $c(x_i)$ is the point in $\mathcal{C}$ that minimizes the distance to x, then it holds that $|\pi(x_i) - c(x_i)| \leq a'_i + a_i$. That implies that

$$\min_{c \in \mathcal{C}} |\pi(x_i) - c| \leq |\pi(x_i) - c(x_i)| \leq a'_i + a_i.$$

Adding this inequality over all points in $\mathcal{Q}$, we get that $\operatorname{cost}_{ip^2}(\mathcal{Q}, \mathcal{C}) \leq ||a' + a||^2$. By the triangle inequality in $\mathbb{R}^n$, $||a' + a|| \leq ||a|| + ||a'||$. That implies that

$$\begin{aligned}
\operatorname{cost}_{ip^2}(\mathcal{Q}, \mathcal{C}) \leq ||a + a'||^2 \leq (||a|| + ||a'||)^2 &\leq \left(\sqrt{\operatorname{cost}_{ip^2}(\mathcal{P}, \mathcal{C})} + \sqrt{\frac{\varepsilon^2}{16} \operatorname{cost}^{*,k}_{ip^2}(\mathcal{P})}\right)^2 \\
&\leq \operatorname{cost}_{ip^2}(\mathcal{P}, \mathcal{C}) + 2\sqrt{\operatorname{cost}_{ip^2}(\mathcal{P}, \mathcal{C})}\sqrt{\frac{\varepsilon^2}{16} \operatorname{cost}^{*,k}_{ip^2}(\mathcal{P})} + \frac{\varepsilon^2}{16} \operatorname{cost}^{*,k}_{ip^2}(\mathcal{P}) \\
&\leq \left(1 + \frac{\varepsilon}{2} + \frac{\varepsilon^2}{16}\right) \operatorname{cost}_{ip^2}(\mathcal{P}, \mathcal{C}) \leq (1 + \varepsilon) \operatorname{cost}_{ip^2}(\mathcal{P}, \mathcal{C}).
\end{aligned}$$

Notice that for the second to last inequality we need that $|\mathcal{C}| \leq k$ because it implies that $\operatorname{cost}^{*,k}_{ip^2}(\mathcal{P}) \leq \operatorname{cost}_{ip^2}(\mathcal{P}, \mathcal{C})$. To obtain the reverse bound, we distinguish to cases. Either $\operatorname{cost}_{\ell_2^2}(\mathcal{Q}, \mathcal{C}) \geq \operatorname{cost}_{\ell_2^2}(\mathcal{P}, \mathcal{C})$, then the $\operatorname{cost}_{\ell_2^2}(\mathcal{P}, \mathcal{C}) - \operatorname{cost}_{\ell_2^2}(\mathcal{Q}, \mathcal{C}) \leq 0 \leq \varepsilon \operatorname{cost}_{ip^2}(\mathcal{P}, \mathcal{C})$. Otherwise, $\operatorname{cost}_{\ell_2^2}(\mathcal{P}, \mathcal{C}) > \operatorname{cost}_{\ell_2^2}(\mathcal{Q}, \mathcal{C})$. Then we exchange the roles of $\mathcal{P}$ and $\mathcal{Q}$ in the above computations to obtain

$$\begin{aligned}
\operatorname{cost}_{\ell_2^2}(\mathcal{P}, \mathcal{C}) &\leq \operatorname{cost}_{ip^2}(\mathcal{Q}, \mathcal{C}) + 2\sqrt{\operatorname{cost}_{ip^2}(\mathcal{Q}, \mathcal{C})}\sqrt{\frac{\varepsilon^2}{16} \operatorname{cost}^{*,k}_{ip^2}(\mathcal{P})} + \frac{\varepsilon^2}{16} \operatorname{cost}^{*,k}_{ip^2}(\mathcal{P}) \\
&< \operatorname{cost}_{ip^2}(\mathcal{Q}, \mathcal{C}) + 2\sqrt{\operatorname{cost}_{ip^2}(\mathcal{P}, \mathcal{C})}\sqrt{\frac{\varepsilon^2}{16} \operatorname{cost}^{*,k}_{ip^2}(\mathcal{P})} + \frac{\varepsilon^2}{16} \operatorname{cost}^{*,k}_{ip^2}(\mathcal{P}) \\
&\leq \operatorname{cost}_{ip^2}(\mathcal{Q}, \mathcal{C}) + \varepsilon \cdot \operatorname{cost}_{ip^2}(\mathcal{P}, \mathcal{C})
\end{aligned}$$

because $\operatorname{cost}_{\ell_2^2}(\mathcal{Q}, \mathcal{C}) < \operatorname{cost}_{\ell_2^2}(\mathcal{P}, \mathcal{C})$. □

6.1.1 Coresets with offsets in inner product spaces

We can compute a coreset with offset for an input set $\mathcal{P}$ from an inner product space in the same way as we did in Euclidean spaces in Section 4.3. Algorithm 6.1 repeats the algorithm. The idea is to partition $\mathcal{P}$ into a set $\mathcal{M}$ of subsets of $\mathcal{P}$ such that $\mathcal{M} = \mathcal{M}^\nu \cup \mathcal{M}'$ with $\mathcal{M}'$ containing only subsets that have a similar 1-means and k-means cost, and $\mathcal{M}^\nu$ containing subsets that have so low 1-means cost that the sum of all of these 1-means costs together is still small. The subsets in $\mathcal{M}'$ are called *k-structured* and those in $\mathcal{M}^\nu$ are called *cheap*. The computation works by recursively partitioning the current subset of $\mathcal{P}$ according to an optimal k-means solution until the current subset satisfies one of the conditions. All subsets that are k-unstructured on the ith level of the recursion form $\mathcal{M}^i$ while the subsets that are created on the $(\nu-1)$th level and enter the νth level of the recursion form $\mathcal{M}^\nu$. $\mathcal{M}'$ is the union of all $\mathcal{M}^i$ for $i = \{1, \ldots, \nu-1\}$.

Algorithm 6.1: Computing a coreset with offset as in Algorithm 4.1.

Function `Partition`($\mathcal{P}$, k, t, ν,ε')
  Compute an optimal center set $\mathcal{C}^* = \{\mu_1, \ldots, \mu_k\}$ for $\mathcal{P}$;
  Let $\mathcal{P}_1, \ldots, \mathcal{P}_k$ be the partitioning of $\mathcal{P}$ induced by $\mathcal{C}^*$;
  Let μ be the centroid of $\mathcal{P}$;
  if $t = \nu$ *or* $\operatorname{dist}^2(\mathcal{P}, \mu) \leq (1+\varepsilon') \sum_{i=1}^{k} \operatorname{dist}^2(\mathcal{P}_i, \mu_i)$ **then**
    $\mathcal{M} := \mathcal{M} \cup \{\mathcal{P}\}$;
  else
    for *i=1,...,k* **do**
      $\mathcal{M} := \mathcal{M} \cup$ `Partition`($\mathcal{P}_i$,k,$t+1$,ν,ε');
    end
  end
  return $\mathcal{M}$;
Algorithm `Coreset`($\mathcal{P}$,k,ε)
  Set $\varepsilon' := \varepsilon^2/50$, $\mathcal{S} := \emptyset$ and $\mathfrak{D} := 0$;
  $\mathcal{M} :=$ `Partition`($\mathcal{P}, k, 0, \lceil \log_{1+\varepsilon'} 1/\varepsilon' \rceil, \varepsilon'$);
  for *every set* $\mathcal{P}' \in \mathcal{M}$ **do**
    $\mathcal{S} := \mathcal{S} \cup \mu(\mathcal{P}')$;
    $w(\mu(\mathcal{P}')) := |\mathcal{P}'|$;
    $\mathfrak{D} = \mathfrak{D} + \operatorname{dist}^2(\mathcal{P}', \mu(\mathcal{P}'))$;
  end
  return $\mathcal{S}$, w and $\mathfrak{D}$;

The proof that Algorithm 6.1 does indeed compute a $(1+\varepsilon)$-coreset for a point set $\mathcal{P}$ from an inner product space is identical to the proof for Euclidean spaces. In other words, the proof in Section 4.3 does not use any property of the Euclidean space that is not true in inner product spaces in general (notice that the Euclidean space is a special case of an inner product space where the inner product is the standard dot product). We have already

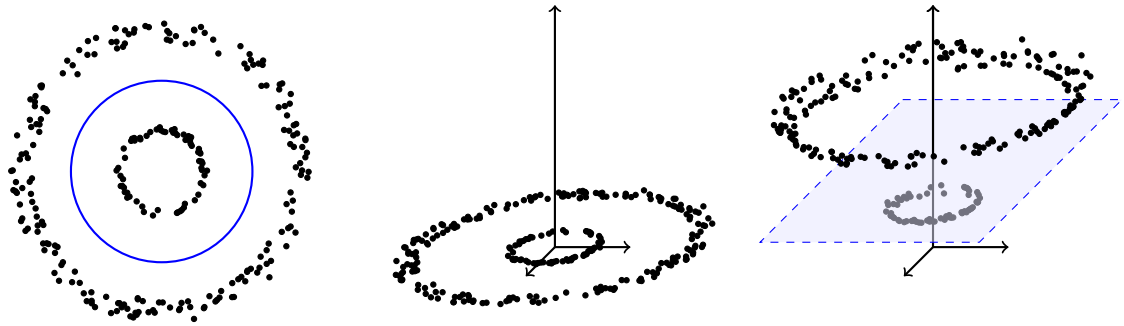


Figure 6.1: Mapping points from $\mathbb{R}^2$ to $\mathbb{R}^3$ with $\phi(x_1, x_2) := (x_1, x_2, ||x||)$.

seen that this is true for Lemma 6.1.2 and Lemma 6.1.3 above. The remaining proof is verbatim and consists only of verifying that no properties were used that are specific for the Euclidean space.

Theorem 6.1.4. *Let $\mathcal{P} \subset \boldsymbol{V}$ be a point set from an inner product space $\boldsymbol{V}$. Algorithm 6.1 computes a $(1+\varepsilon)$-coreset $\mathcal{S}$ with offset $\mathfrak{D}$ for the k-means problem on $\mathcal{P}$ in $\boldsymbol{V}$. In particular, it holds for every set of k centers $\mathcal{C}$ from $\boldsymbol{V}$ that*

$$|\operatorname{cost}_{ip^2}(\mathcal{P}, \mathcal{C}) - \operatorname{cost}_{ip^2,w}(\mathcal{S}, \mathcal{C}) - \mathfrak{D}| \leq \varepsilon \cdot \operatorname{cost}_{ip^2}(\mathcal{P}, \mathcal{C}).$$

$\mathcal{S}$ contains $k^{\mathcal{O}(\varepsilon^{-2} \ln \varepsilon^{-1})}$ points.

6.2 The kernel k-means problem

In this section, we see an application of the coreset for inner product spaces to the kernel k-means problem. In Chapter 2 on page 11, we discussed the limitations of finding clusterings, and in particular discussed that the k-means objective can only produce clusterings that are linearly separable. We had a look at an example with rings (Figure 2.2 on page 8). A similar (but simpler) example is depicted on the left side of Figure 6.1 (only consider the black points, the blue circle is explained later).

Assume we want to use a k-means algorithm to separate the two clusters. One way to do this is to lift the points to a higher dimensional space where they are separable. The right side of Figure 6.1 shows the points after adding the length of each point as the third coordinate. The picture in the middle is included to show the effect of the three-dimensional presentation of the data. It shows the two-dimensional points as points in the xy-plane.

Because the origin lies at the center of the rings, adding the length as a coordinate separates the two clusters nicely. For example, the hyperplane indicated by the blue rectangle separates the two clusters. It corresponds to the blue circle in the left picture, which separates the clusters in their two-dimensional version. By adding one more dimension, it is possible to simulate a curved separator by a higher dimensional hyperplane. By adding

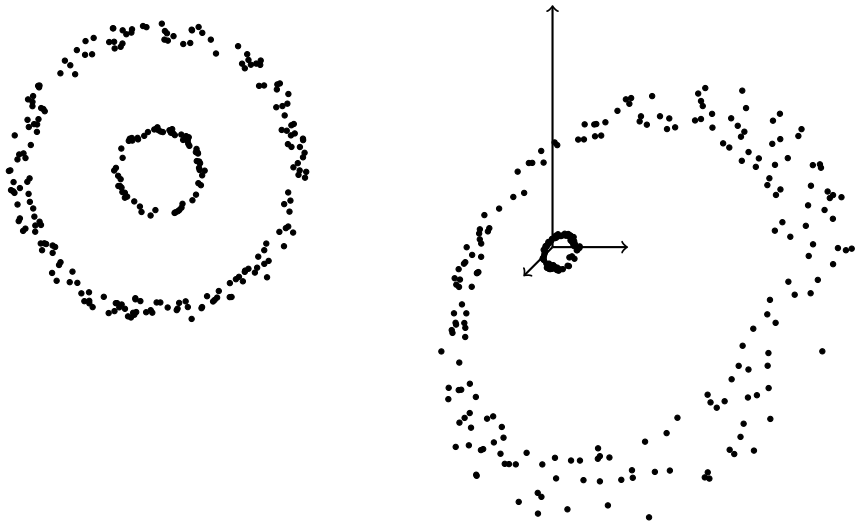


Figure 6.2: Mapping points from $\mathbb{R}^2$ to $\mathbb{R}^3$ with $\phi(x_1, x_2) := (x_1^2, x_2^2, \sqrt{2}x_1x_2)$.

even more dimensions, the complexity of the separator in the original space can be increased further. We can imagine that by allowing extreme numbers of dimensions, it is possible to find a higher dimensional space where any partitioning of the points into two sets can be separated by a hyperplane (for example, reserve d dimensions for each point and construct an (nd)-dimensional point set).

Our example uses a quite specific map and only three dimensions. A more common choice would be to define a mapping like $\phi(x_1, x_2) := (x_1^2, x_2^2, x_1x_2, x_2x_1)$ which maps each point $(x_1, x_2) \in \mathbb{R}^2$ to $\mathbb{R}^4$ using polynoms in the input coordinates, in this case of degree two. Assume we compute the standard inner product in $\mathbb{R}^d$ of $\phi(x)$ and $\phi(y)$ for $x, y \in \mathbb{R}^2$. What we get is

$$\begin{aligned}\langle (x_1^2, x_2^2, x_1x_2, x_2x_1), (y_1^2, y_2^2, y_1y_2, y_2y_1)\rangle = x_1^2y_1^2 + x_2^2y_2^2 + 2x_1x_2y_1y_2 &= (x_1y_1 + x_2y_2)^2 \\ &= \langle x, y\rangle^2.\end{aligned}$$

We notice that we can compute the inner product in the target space directly, without even computing the lifted points. Notice that the same happens for $\phi(x_1, x_2) := (x_1^2, x_2^2, \sqrt{2}x_1x_2)$ which maps to $\mathbb{R}^3$ and can thus be depicted like in Figure 6.2. A formula that allows the direct computation of the inner product is called a *kernel function* (we define it more precisely below).

Whenever an algorithm only needs the inner product to perform its computations, we do not need to compute the lifted points. Instead, we can just use the kernel function whenever we need an inner product. This technique is called *kernel trick* and it works for example for Lloyd's algorithm as we discuss below. Recall that we are not actually interested in the target space or the images of the points. We are interested in the partitioning that hyperplanes in the target space induce because these can provide a nice partitioning of our original points. The beauty of kernel methods is that we can compute optimal

partitionings in a very high dimensional, even infinitely dimensional target space without actually computing this space.

Popular kernel functions for Euclidean points. The kernel function $\kappa : \mathbb{R}^2 \to \mathbb{R}$ defined by $\kappa(x, y) = \langle x, y\rangle^2$ is an instantiation of a *polynomial kernel function*. It allows to separate points in the plane by a quadratic function instead of a hyperplane. It can also be used when the input space is $\mathbb{R}^d$ as we know how to compute the inner product for d-dimensional vectors. A common generalization is to use higher polynomials of the inner product and to allow offsets, i. e., to define $\kappa(x, y) = (\langle x, y\rangle + c)^\ell$ for constants $c \in \mathbb{R}$ and $\ell \in \mathbb{N}$. This allows to separate the input point set by a polynom of degree ℓ.

Notice that for polynomial kernel functions, the implicit target space is still an Euclidean space of finite dimension (more precisely, the degree is $\binom{d+\ell}{\ell}$, see Proposition 9.2 on page 293 in [STC04]). There are popular kernel functions where this is not true, foremost *Gaussian kernel functions*. For a broad introduction to different kernel functions and their construction, see [STC04], in particular Chapter 2, Chapter 3 and Part III.

Valid kernel functions. As for the popular Gaussian kernels, the target space may be of infinite dimension, so Euclidean spaces are not general enough. The generalization of choice are Hilbert spaces. Recall that a Hilbert space is an inner product space that is also a complete metric space. Hilbert spaces are similar to Euclidean spaces but can have infinite dimension.

In the short example above, we defined a target space and a suitable mapping ϕ, then we analytically derived the kernel function. Instead, it is more convenient to construct a kernel function directly, without even defining the target space. This is possible due to the rich theory on kernel functions which includes a characterization of all functions $\mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ which can be interpreted as an inner product in a Hilbert space. We call these functions *valid kernel functions*.

To characterize them, recall that a matrix $A \in \mathbb{R}^{n\times n}$ is *positive semi-definite* if $v^T Av \geq 0$ for all $v \in \mathbb{R}^n$. Let $\mathbf{X}$ be a set and let $\kappa : \mathbf{X} \times \mathbf{X} \to \mathbb{R}$ be a function. For any finite subset $\mathcal{P} := \{\mathfrak{x}_1, \ldots, \mathfrak{x}_n\} \subset \mathbf{X}$ with $|\mathcal{P}| = n$, define the matrix $K^{\mathcal{P}}$ by $K^{\mathcal{P}}_{ij} = \kappa(\mathfrak{x}_i, \mathfrak{x}_j)$. We say that κ has the *finitely positive semi-definite property* if for every finite $\mathcal{P} \subset \mathbf{X}$, the matrix $K^{\mathcal{P}}$ is positive semi-definite. For example, the standard Euclidean dot product satisfies the finitely positive semi-definite property. The following theorem says that exactly the functions with the finitely positive semi-definite property are valid kernel functions.

Theorem 6.2.1 (Characterization of kernels, [STC04])**.** *Let $\kappa : \boldsymbol{X} \times \boldsymbol{X} \to \mathbb{R}$ be a function which is either continuous or has a finite domain. There exists a Hilbert space $\boldsymbol{V}$ over the field $\mathbb{R}$ with inner product $\langle \cdot, \cdot \rangle$ and a mapping $\phi : X \to \boldsymbol{V}$ such that κ can be decomposed as*

$$\kappa(x, y) = \langle \phi(x), \phi(y) \rangle$$

if and only if κ satisfies the finitely positive semi-definite property. Functions that satisfy this property are called valid kernel functions.

Notice that we defined kernel functions on arbitrary sets. Indeed, they can be used for different data, including for example texts. Part III of [STC04] explains various different kernel functions for different types of input data. It is only important that the target space is a Hilbert space where we implicitly compute the inner product, i. e., that the kernel function is valid. The kernel trick then assures that the type of the data does not matter as it only needs to partition the input data and to access to the kernel function.

Kernel matrices. Kernel functions can also be specified for the input points only instead of giving an explicit formula, i. e., the input can be given in form of the matrix $K^{\mathcal{P}}$, then called *kernel matrix*. This works as long as we only want to evaluate the kernel function on input points. Notice that by setting $\kappa(\chi, y) = 0$ for all χ and y where either $\chi \notin \mathcal{P}$ or $y \notin \mathcal{P}$ (or both), any positive semidefinite matrix can be extended to a valid kernel function, so any positive semidefinite matrix can be used as a kernel matrix. How the kernel function is given – explicitly or as a kernel matrix – does not change how we are going to use it. Thus, when we refer to the kernel function κ in the following, then it may either be stored as an explicit function or via a kernel matrix.

Kernel k-means problems. A Hilbert space is an inner product space that is also a complete metric space, so $\mathbf{V}$ in Theorem 6.2.1 is in particular an inner product space. Thus, given a point set $\mathcal{P} \subset \mathbb{R}^d$, Definition 6.1.1 defines the k-means problem in $\mathbf{V}$ for $\phi(\mathcal{P}) := \{\phi(x) \mid x \in \mathcal{P}\}$, the image of $\mathcal{P}$ under ϕ. However, solving the k-means problem in $\mathbf{V}$ would require knowledge of $\mathbf{V}$, and the solution would consist of points from $\mathbf{V}$. We are interested in solving the problem implicitly, and the solution that we can hope for is a partitioning of $\mathcal{P}$ that corresponds to a partitioning of $\phi(\mathcal{P})$ that is optimal for the k-means problem in $\mathbf{V}$. To emphasize this, we define a special version of the k-means problem for our purpose which optimizes over all partitionings. Notice that the optimal solution of the following problem is equal to the partitioning induced by the optimal solution of the k-means problem for $\phi(\mathcal{P})$ in $\mathbf{V}$.

Definition 6.2.2. *Let $\boldsymbol{X}$ be a set and let $\kappa : \boldsymbol{X} \times \boldsymbol{X} \to \mathbb{R}$ be a valid kernel function, i. e., there exists a Hilbert space $\boldsymbol{V}$ and a mapping $\phi : \boldsymbol{X} \to \boldsymbol{V}$ such that $\kappa(\chi, y) = \langle \phi(\chi), \phi(y) \rangle$ for all $\chi, y \in \boldsymbol{X}$. Let $\mathcal{P} \subset \boldsymbol{X}$ be an input point set. The* kernel k-means problem with kernel κ *is to compute a partitioning $\mathcal{P}_1, \ldots, \mathcal{P}_k$ of $\mathcal{P}$ that minimizes*

$$\sum_{i=1}^{k} \sum_{\chi \in \mathcal{P}_i} \min_{i=1,\ldots,k} |\chi - \mu(\mathcal{P}_i)|^2$$

where $|\cdot| : \boldsymbol{V} \to \mathbb{R}$ is the norm in $\boldsymbol{V}$ based on the inner product $\langle \cdot \rangle : \boldsymbol{V} \times \boldsymbol{V} \to \mathbb{R}$.

The kernel trick and Lloyd's algorithm. Assume that we are given a kernel function $\kappa : \mathbf{X} \times \mathbf{X} \to \mathbb{R}$ without knowledge of the implicitly defined Hilbert space or the mapping. We want to perform Lloyd's algorithm on an input point set $\mathcal{P}$ to compute a (heuristic) solution for the kernel k-means problem with kernel κ. As the actual optimization happens

in an unknown Hilbert space, we cannot store the actual centers. However, all centers that occur during Lloyd's algorithm are centroids of subsets of the input points. We can implicitly store them by storing the partitioning. Notice that it holds for all $\mathcal{P}' \subset \mathcal{P}$ that

$$\begin{aligned}
&|\phi(x) - \mu(\phi(\mathcal{P}'))|^2 = \left|\phi(x) - \frac{1}{|\mathcal{P}'|}\sum_{y\in\mathcal{P}'}\phi(y)\right|^2 \\
=&\left\langle \phi(x) - \frac{1}{|\mathcal{P}'|}\sum_{y\in\mathcal{P}'}\phi(y), \phi(x) - \frac{1}{|\mathcal{P}'|}\sum_{y\in\mathcal{P}'}\phi(y)\right\rangle \\
=&|\phi(x)|^2 - \frac{1}{|\mathcal{P}'|}\sum_{x\in\mathcal{P}'}\langle\phi(x),\phi(y)\rangle - \frac{1}{|\mathcal{P}'|}\sum_{y\in\mathcal{P}'}\langle\phi(y),\phi(x)\rangle + \frac{1}{|\mathcal{P}'|^2}\sum_{y,z\in\mathcal{P}'}\langle\phi(y),\phi(z)\rangle \\
=&\kappa(x,x) - \frac{1}{|\mathcal{P}'|}\sum_{x\in\mathcal{P}'}\kappa(x,y) - \frac{1}{|\mathcal{P}'|}\sum_{y\in\mathcal{P}'}\kappa(y,x) + \frac{1}{|\mathcal{P}'|^2}\sum_{y,z\in\mathcal{P}'}\kappa(y,z) \qquad (6.3)
\end{aligned}$$

where ϕ is the unknown mapping and $\phi(\mathcal{P}') := \{\phi(x) \mid x \in \mathcal{P}'\}$. This implies that computing the distance of $\phi(x)$ to a center can be done with only using κ if the center is a centroid. In particular, the reassignment step in Lloyd's algorithm can be performed by only using κ, and the cost of the current solution can be calculated at the same time.

Altogether, Lloyd's algorithm works like the following. The k centers are initialized by drawing k random points $x'_1, \ldots, x'_k$ from $\mathcal{P}$. The point set is then partitioned into k subsets $\mathcal{P}_1, \ldots, \mathcal{P}_k$ such that for all points $x \in \mathcal{P}_i$, $|\phi(x) - \phi(x'_i)|^2$ is minimal among all choices of i. Notice that each point can be seen as a subset of the input point set, so computing the distances is a special case of Equation 6.3. The partitioning represents the new set of centers $\mu(\phi(\mathcal{P}_1)), \ldots, \mu(\phi(\mathcal{P}_k))$. By Equation 6.3, the cost of this solution can be computed, and each point $x \in \mathcal{P}$ can be reassigned such that $|\phi(x) - \mu(\phi(\mathcal{P}_i))|^2$ is minimal, constituting a new partitioning. This is iterated until the partitioning does not change anymore.

6.2.1 Coresets for kernel k-means problems

The first thing we notice is that our coreset construction for the k-means problem in $\mathbf{V}$ implies the existence of a coreset for $\phi(\mathcal{P})$.

Corollary 6.2.3. *Let $\boldsymbol{X}$ be a set and let $\mathcal{P} \subset \boldsymbol{X}$ be an input containing n points. Let $\boldsymbol{V}$ be a Hilbert space with inner product $\langle\cdot,\cdot\rangle$ and let $|\cdot| : \boldsymbol{V} \to \mathbb{R}$ be the norm induced by the inner product. Let $\kappa : \boldsymbol{X} \times \boldsymbol{X} \to \mathbb{F}$ be a valid kernel function with decomposition $\kappa(\chi, y) = \langle\phi(\chi), \phi(y)\rangle$ for all $\chi, y \in X$ for a mapping $\phi : \boldsymbol{X} \to \boldsymbol{V}$. Then there exists a weighted set $\mathcal{S} \subset \boldsymbol{V}$ of $k^{\mathcal{O}(\varepsilon^{-2}\cdot\log\varepsilon^{-1})}$ points from $\boldsymbol{V}$ and a constant $\mathfrak{D} \in \mathbb{R}$ such that it holds for all $\mathcal{C} \subset \boldsymbol{V}$ with $|\mathcal{C}| = k$ that*

$$\left|\sum_{\chi\in\mathcal{P}}\min_{c\in\mathcal{C}}|\phi(\chi) - c|^2 - \sum_{s\in\mathcal{S}}\min_{c\in\mathcal{C}}w(s)|s - c|^2 + \mathfrak{D}\right| \leq \varepsilon\cdot\mathrm{cost}_{ip^2}(\phi(\mathcal{P}), \mathcal{C})$$

where $w(s)$ is the weight of s and $\phi(\mathcal{P}) := \{\phi(x) \mid x \in \mathcal{P}\}$ is the image of $\mathcal{P}$ under ϕ.

Proof. Because $\mathbf{V}$ is a Hilbert space, the statement follows directly from applying Theorem 6.1.4 to the set $\phi(\mathcal{P}) = \{\phi(x) \mid x \in \mathcal{P}\}$. □

The coreset in Corollary 6.2.3 is a coreset for the k-means problem in $\mathbf{V}$, but it is not what we desire from a coreset for the kernel k-means problem. That is because the points in the coreset are from $\mathbf{V}$. We cannot even store this coreset without knowledge of ϕ and $\mathbf{V}$. We notice, however, that most of the coreset construction can be performed by only using κ, if the optimal clustering is determined in a way that only uses centroids as centers, for example by iterating through all possible partitionings of the input point set into k subsets. The only thing that cannot be done is actually computing the coreset points, because computing the actual centroids requires knowledge of ϕ.

Let $\mathcal{P} \in \mathbf{X}$ be the input point set and let $\phi(\mathcal{P})$ be the lifted point set. Consider Algorithm 6.2. All steps can be executed without actually computing V or ϕ. By Theorem 6.1.4, the result is a partitioning of the input point set into $n' \leq k^{\mathcal{O}(\varepsilon^{-2} \cdot \log \varepsilon^{-1})}$ subsets such that when clustering all points in the same subset together, the error is small.

Algorithm 6.2: Variation of the inner product k-means coreset construction.

```
1  Algorithm Subdivide(P,k,ε)
2  |  Set n' := 0 and ε' := ε'^2/50;
3  |  M := Partition(P, k, 0, ⌈log_{1+ε'} 1/ε'⌉, ε', n');
4  |  return M and n';
5  Function Partition(P, k, t, ν,ε',n')
6  |  Compute an optimal k-means solution on φ(P);
7  |  Keep track of a partitioning {P_1, ..., P_k} of P with the property
8  |  that {φ(P_1), ..., φ(P_k)} is optimal for φ(P);
9  |  Let μ be the centroid of φ(P), let μ_i be the centroid of φ(P_i);
10 |  if t = ν or dist^2(φ(P), μ) ≤ (1 + f(ε)) Σ_{i=1}^k dist^2(φ(P_i), μ_i) then
11 |  |  n' := n' + 1;
12 |  |  Set S_{n'} := P and M := M ∪ {S_{n'}};
13 |  else
14 |  |  for j=1,...,k do
15 |  |  |  M := M∪ Partition(P_i,k,t + 1,ν,ε',n');
16 |  |  end
17 |  end
18 |  return M;
```

Corollary 6.2.4. *Let $\mathcal{P} \subset \boldsymbol{X}$ be a point set containing n points from a set $\boldsymbol{X}$. Let $\boldsymbol{V}$ be a Hilbert space with inner product $\langle \cdot, \cdot \rangle : \boldsymbol{V} \times \boldsymbol{V} \to \mathbb{R}$ and let $| \cdot | : \boldsymbol{V} \to \mathbb{R}$ be the norm induced by the inner product. Let $\kappa : \boldsymbol{X} \times \boldsymbol{X} \to \mathbb{F}$ be a valid kernel function with decomposition $\kappa(\chi, y) = \langle \phi(\chi), \phi(y) \rangle$ for all $\chi, y \in \boldsymbol{X}$ for a mapping $\phi : \boldsymbol{X} \to \boldsymbol{V}$. A*

partitioning $\mathcal{P}_1, \ldots, \mathcal{P}_{n'}$ *with* $n' \leq k^{\mathcal{O}(\varepsilon^{-2} \cdot \log \varepsilon^{-1})}$ *satisfying*

$$\left| \sum_{\chi \in \mathcal{P}} \min_{\chi \in \mathcal{C}} |\phi(\chi) - c|^2 - \sum_{\ell=1}^{n'} \min_{c \in \mathcal{C}} \sum_{\chi \in \mathcal{P}_\ell} |\phi(x) - c|^2 \right| \leq \varepsilon \operatorname{cost}_{ip^2}(\phi(\mathcal{P}), \mathcal{C})$$

exists and can be computed without explicit knowledge of $\boldsymbol{V}$ *or* ϕ.

Corollary 6.2.4 does not provide a coreset, just a rule which points can be assigned as a group without causing too much error. For an actual coreset, we need to replace the n' subsets by something smaller. We cannot replace them by their centroids without explicitly computing V and ϕ. Instead, we replace them by a smaller subset. For this, we need the following lemma by Inaba et al. [IKI94].

Lemma 6.2.5. *Let* $\mathcal{P}$ *be a set of* n *points in an inner product space* $\boldsymbol{V}$*. Let* $\mathcal{S}$ *be a set of* m *points sampled from* $\mathcal{P}$ *uniformly at random. For any* $\delta > 0$ *it holds that*

$$\operatorname{Prob}\left(|\mu(\mathcal{S}) - \mu(\mathcal{P})|^2 > \frac{1}{\delta m |\mathcal{P}|} \sum_{\chi \in \mathcal{P}} |\chi - \mu(\mathcal{P})|^2 \right) \leq \delta.$$

Proof. Let χ_i be the random variable for the ith sampled point, so $\chi := \sum_{i=1}^m \chi_i / m = \mu(\mathcal{S})$ is the centroid of the sampled points. The points are drawn independently, so all χ_i have the same expected value and variance. More precisely, it holds that for all $i = 1, \ldots, m$ that

$$\operatorname{E}(\chi_i) = \sum_{\chi \in \mathcal{P}} \chi / |\mathcal{P}| = \mu(\mathcal{P})$$

and consequently, $\operatorname{E}(\chi) = \mu(\mathcal{P})$ as well. This means that $\operatorname{Var}(\chi) = \operatorname{E}((\chi - \operatorname{E}(\chi))^2) = \operatorname{E}(|\mu(\mathcal{S}) - \mu(\mathcal{P})|^2)$. Now notice that $\operatorname{Var}(\chi) = \operatorname{Var}(\chi_i)/m$ for all $i = 1, \ldots, m$[1]. The variance of χ_i is $\operatorname{Var}(\chi_i) = \operatorname{E}((\chi_i - \mu(\mathcal{P}))^2) = \frac{1}{|\mathcal{P}|} \sum_{\chi \in \mathcal{P}} |\chi - \mu(\mathcal{P})|^2$ for all $i = 1, \ldots, m$. So together we get that

$$\operatorname{E}(|\mu(\mathcal{S}) - \mu(\mathcal{P})|^2) = \operatorname{Var}(\chi) = \frac{1}{m|\mathcal{P}|} \sum_{\chi \in \mathcal{P}} |\chi - \mu(\mathcal{P})|^2.$$

Markov's Inequality then implies the statement of the Lemma. □

Consider the partitioning of $\mathcal{P}$ into $n' \leq k^{\mathcal{O}(\varepsilon^{-2} \cdot \log \varepsilon^{-2})}$ subsets $\mathcal{P}_1, \ldots, \mathcal{P}_{n'}$ computed by Algorithm 6.2. By Lemma 6.2.5, we know that there exists a small subset of every $\phi(\mathcal{P}_i)$ such that its centroid is close to $\mu(\phi(\mathcal{P}_i))$. Notice that we can sample a multiset $\mathcal{S}_i$ of points from $\mathcal{P}_i$ instead of $\phi(\mathcal{P}_i)$ and use the image $\phi(\mathcal{S}_i)$ as a uniform sample from $\phi(\mathcal{P}_i)$.

[1] This can for example be seen by the following two steps. First, $\operatorname{Var}(\chi) = \operatorname{Var}(\frac{1}{m}\sum_{i=1}^m \chi_i) = \operatorname{E}((\frac{1}{m}\sum_{i=1}^m \chi_i - \operatorname{E}(\frac{1}{m}\sum_{i=1}^m \chi_i))^2) = \frac{1}{m^2}\operatorname{E}((\sum_{i=1}^m \chi_i - \operatorname{E}(\sum_{i=1}^m \chi_i))^2) = \operatorname{Var}(\sum_{i=1}^m \chi_i)/m^2$ because expected values are linear. Second, $\operatorname{Var}(\sum_{i=1}^m \chi_i) = \sum_{i=1}^m \operatorname{Var}(\chi_i) = m \cdot \operatorname{Var}(\chi_i)$ by Theorem 3.5 on page 48 of [MU05].

For a given failure probability δ', we set $\delta := \delta'/(n')$ and $m := \frac{16}{\delta\varepsilon^2} = \frac{16}{\delta'\varepsilon^2}n'$. Then Lemma 6.2.5 guarantees us that sampling m points from subset $\mathcal{P}_i$ yields a (multi)set $\mathcal{S}_i \subset \mathcal{P}_i$ that satisfies

$$|\mu(\phi(\mathcal{S}_i)) - \mu(\phi(\mathcal{P}_i))|^2 \leq \frac{\varepsilon^2}{16}\frac{1}{|\mathcal{P}_i|}\sum_{\chi\in\mathcal{P}_i}|\phi(\chi) - \mu(\phi(\mathcal{P}_i))|^2 \tag{6.4}$$

with probability δ'/n'. By the union bound, this holds for all n' sets $\mathcal{S}_i$ simultaneously with probability $1-\delta'$. We also define a weight for each sample point χ in $\mathcal{S}_i$ and set it to $|\mathcal{P}_i|/m$. Thus, the points in $\mathcal{S}_i$ have total weight $|\mathcal{P}_i|$.

The subsets $\mathcal{S}_i$ are now supposed to take the role of the centroids in the coreset construction in Algorithm 6.1 that we cannot compute without knowing $\mathbf{V}$ and ϕ. So, instead of the centroid itself, we store a small subset of points with nearly the same centroid. In contrast to Algorithm 6.1 we do not have control over how much we lose by replacing $\mathcal{P}_i$ by $\mathcal{S}_i$ or $\phi(\mathcal{P}_i)$ by $\phi(\mathcal{S}_i)$, respectively. When we replaced a subset by its centroid, the loss was exactly $\sum_{\chi\in\mathcal{P}_i}|\phi(\chi) - \mu(\phi(\mathcal{P}_i))|^2$. Now we simply compute the change and let this contribute to our constant. We set

$$\mathfrak{D}_i := \sum_{\chi\in\mathcal{P}_i}|\phi(\chi) - \mu(\phi(\mathcal{P}_i))|^2 - \sum_{y\in\mathcal{S}_i} w(y)\cdot|\phi(y) - \mu(\phi(\mathcal{S}_i))|^2 \text{ and } \mathfrak{D} := \sum_{i=1}^{n'}\mathfrak{D}_i.$$

Notice that $\mathfrak{D}_i$ can be negative if $\phi(\mathcal{S}_i)$ contains many outliers. Now let $\mathcal{C} \subset \mathbf{V}$ be any center set. When using the coreset, all points in $\phi(\mathcal{S}_i)$ are clustered together, and the constant $\mathfrak{D}_i$ is added. This costs

$$\begin{aligned}
&\min_{c\in\mathcal{C}}\sum_{y\in\mathcal{S}_i} w(y)\cdot|\phi(y) - c|^2 + \mathfrak{D}_i \\
&= \sum_{y\in\mathcal{S}_i} w(y)\cdot|\phi(y) - \mu(\phi(\mathcal{S}_i))|^2 + w(y)\cdot m\cdot\min_{c\in\mathcal{C}}|\mu(\phi(\mathcal{S}_i)) - c|^2 + \mathfrak{D}_i \\
&= \sum_{\chi\in\mathcal{P}_i}|\phi(\chi) - \mu(\phi(\mathcal{P}_i))|^2 + \frac{|\mathcal{P}_i|}{m}\cdot m\cdot\min_{c\in\mathcal{C}}|\mu(\phi(\mathcal{S}_i)) - c|^2 \\
&= \sum_{\chi\in\mathcal{P}_i}|\phi(\chi) - \mu(\phi(\mathcal{P}_i)) + \mu(\phi(\mathcal{S}_i)) - \mu(\phi(\mathcal{S}_i))|^2 + |\mathcal{P}_i|\cdot\min_{c\in\mathcal{C}}|\mu(\phi(\mathcal{S}_i)) - c|^2 \\
&= \sum_{\chi\in\mathcal{P}_i}|\pi(\phi(\chi)) - \mu(\phi(\mathcal{S}_i))|^2 + |\mathcal{P}_i|\cdot\min_{c\in\mathcal{C}}|\mu(\phi(\mathcal{S}_i)) - c|^2 \\
&= \sum_{\chi\in\mathcal{P}_i}|\pi(\phi(\chi)) - c(\mu(\phi(\mathcal{S}_i)))|^2
\end{aligned} \tag{6.5}$$

where we use the mapping $\pi(\phi(\chi)) := \phi(\chi) + \mu(\phi(\mathcal{S}_i)) - \mu(\phi(\mathcal{P}_i))$ that moves $\phi(\mathcal{P}_i)$ such that its centroid coincides with the centroid of $\phi(\mathcal{S}_i)$.

We notice two things about Equality 6.5. First, it expresses the cost of clustering all weighted points in $\phi(\mathcal{S}_i)$ with center $c(\mu(\phi(\mathcal{S}_i)))$ as the cost of clustering all points in $\pi(\phi(\mathcal{P}_i))$ with the same center. So both clustering costs are not the best possible clustering

costs with $\mathcal{C}$ but the cost of clustering everything with $c(\mu(\phi(\mathcal{S}_i)))$. Second, our aim is to connect this clustering cost to clustering all points in $\phi(\mathcal{P}_i)$ with the center $c(\mu(\phi(\pi_i)))$, which is the best center when clustering $\phi(\mathcal{P}_i)$ with one center. We already know by Corollary 6.2.4 that clustering all points in $\phi(\mathcal{P}_i)$ with the same center does not incur too much cost.

Our plan is to finish the proof by using Lemma 6.1.3. For that, we need a bound on the squared movement lengths. By the definition of the mapping π, by Inequality (6.4) and by Lemma 6.1.2, it holds that

$$\begin{aligned}\sum_{\chi\in\mathcal{P}_i} |\phi(\chi) - \pi(\phi(\chi))|^2 =& |\mathcal{P}_i| \cdot |\mu(\phi(\mathcal{S}_i)) - \mu(\phi(\mathcal{P}_i))|^2 \\ \leq& |\mathcal{P}_i| \cdot \frac{\varepsilon^2}{16} \cdot \frac{1}{|\mathcal{P}_i|} \sum_{\chi\in\mathcal{P}_i} |\phi(\chi) - \mu(\phi(\mathcal{P}_i))|^2 \leq \frac{\varepsilon^2}{16} \cdot \operatorname{cost}^{*,k}_{ip^2}(\phi(\mathcal{P}_i))\end{aligned}$$

for every $i = 1, \ldots, n'$.

Now we can use Lemma 6.1.3. We need the following two statements which we can derive from the statement of the Lemma. Notice that Lemma 6.1.3 is applied to the center sets $\{c(\mu(\phi(\mathcal{S}_i)))\}$ and $\{c(\mu(\phi(\mathcal{P}_i)))\}$, respectively, which is possible since we assume $|\mathcal{C}| \leq k$ in the Lemma (not $|\mathcal{C}| = k$). We get that

$$\operatorname{cost}_{ip^2}(\pi(\phi(\mathcal{P}_i)), c(\mu(\phi(\mathcal{P}_i)))) \leq (1+\varepsilon) \cdot \operatorname{cost}_{ip^2}(\phi(\mathcal{P}_i), c(\mu(\phi(\mathcal{P}_i)))) \tag{6.6}$$

$$\operatorname{cost}_{ip^2}(\phi(\mathcal{P}_i), c(\mu(\phi(\mathcal{S}_i)))) \leq (1+\varepsilon) \cdot \operatorname{cost}_{ip^2}(\pi(\phi(\mathcal{P}_i)), c(\mu(\phi(\mathcal{S}_i)))) \tag{6.7}$$

holds for every center set $\mathcal{C} \in \mathbf{V}$ with $|\mathcal{C}| = k$ (where $c(\mu(\phi(\mathcal{S}_i)))$ and $c(\mu(\phi(\mathcal{P}_i)))$ are the centers in $\mathcal{C}$ closest to $\mu(\phi(\mathcal{S}_i))$ and $\mu(\phi(\mathcal{P}_i))$, respectively).

When clustering a point set with one center, the center closest to the centroid is the best choice due to Lemma 6.1.2. That allows us to conclude

$$\begin{aligned}\operatorname{cost}_{ip^2}(\pi(\phi(\mathcal{P}_i)), c(\mu(\phi(\mathcal{S}_i)))) &\leq \operatorname{cost}_{ip^2}(\pi(\phi(\mathcal{P}_i)), c(\mu(\phi(\mathcal{P}_i)))) \\ &\leq (1+\varepsilon) \cdot \operatorname{cost}_{ip^2}(\phi(\mathcal{P}_i), c(\mu(\phi(\mathcal{P}_i))))\end{aligned} \tag{6.8}$$

from Inequality 6.6. By the same trick and by using Inequality 6.7 and Inequality 6.8, we get that

$$\begin{aligned}\operatorname{cost}_{ip^2}(\phi(\mathcal{P}_i), c(\mu(\phi(\mathcal{P}_i)))) &\leq \operatorname{cost}_{ip^2}(\phi(\mathcal{P}_i), c(\mu(\phi(\mathcal{S}_i)))) \\ &\leq (1+\varepsilon) \cdot \operatorname{cost}_{ip^2}(\pi(\phi(\mathcal{P}_i)), c(\mu(\phi(\mathcal{S}_i)))) \\ &= \operatorname{cost}_{ip^2}(\pi(\phi(\mathcal{P}_i)), c(\mu(\phi(\mathcal{S}_i)))) + \varepsilon \operatorname{cost}_{ip^2}(\pi(\phi(\mathcal{P}_i)), c(\mu(\phi(\mathcal{S}_i)))) \\ &= \operatorname{cost}_{ip^2}(\pi(\phi(\mathcal{P}_i)), c(\mu(\phi(\mathcal{S}_i)))) + 2\varepsilon \operatorname{cost}_{ip^2}(\phi(\mathcal{P}_i), c(\mu(\phi(\mathcal{P}_i)))).\end{aligned}$$

Together, we have

$$\begin{aligned}&\left| \min_{c\in\mathcal{C}} \sum_{y\in\mathcal{S}_i} w(y) \cdot |\phi(y) - c|^2 + \mathfrak{D}_i - \operatorname{cost}_{ip^2}(\phi(\mathcal{P}_i), c(\mu(\phi(\mathcal{P}_i)))) \right| \\ =& \left| \operatorname{cost}_{ip^2}(\pi(\phi(\mathcal{P}_i)), c(\mu(\phi(\mathcal{S}_i)))) - \operatorname{cost}_{ip^2}(\phi(\mathcal{P}_i), c(\mu(\phi(\mathcal{P}_i)))) \right| \\ \leq& 2\varepsilon \operatorname{cost}_{ip^2}(\phi(\mathcal{P}_i), c(\mu(\phi(\mathcal{P}_i))))\end{aligned}$$

Algorithm 6.3: The kernel k-means coreset construction.

Algorithm `KernelCoreset(`$\mathcal{P}$`,`k`)`
  $\{\mathcal{M}, n'\} :=$ `Subdivide(`$\mathcal{P}, k, \varepsilon/5$`)`;
  for *i=1,...,n'* **do**
    Sample $m := \frac{16}{\varepsilon} n'$ points from $\mathcal{P}_i$ uniformly at random, store them as $\mathcal{S}_i$;
    Set $\mathfrak{D}_i := \sum_{\chi \in \mathcal{P}_i} |\phi(\chi) - \mu(\phi(\mathcal{P}_i))|^2 - \sum_{y \in \mathcal{S}_i} w(y) \cdot |\phi(y) - \mu(\phi(\mathcal{S}_i))|^2$;
    Set $\mathfrak{D} := \mathfrak{D} + \mathfrak{D}_i$;
    Set $w(\chi) := \frac{|\mathcal{P}_i|}{m}$ for all $\chi \in S_i$;
  end
  $\mathcal{S} := \bigcup_{i=1}^{m} \mathcal{S}_i$;
  return $\mathcal{S}_1, \ldots, \mathcal{S}_{n'}$, w and $\mathfrak{D}$;

for every center set $\mathcal{C}$ and for every $i = 1, \ldots, n'$. Our findings apply to all sets $\mathcal{S}_i$, so we can replace all $\mathcal{P}_i$ by the corresponding $\mathcal{S}_i$ at a total error of

$$\sum_{i=1}^{n'} 2\varepsilon \operatorname{cost}_{ip^2}(\phi(\mathcal{P}_i), c(\mu(\phi(\mathcal{P}_i)))) \leq 2\varepsilon(1+\varepsilon) \operatorname{cost}_{ip^2}(\phi(\mathcal{P}), \mathcal{C})$$

where the inequality follows because Corollary 6.2.4 yields that

$$\left| \operatorname{cost}_{ip^2}(\phi(\mathcal{P}), \mathcal{C}) - \sum_{\ell=1}^{n'} \operatorname{cost}_{ip^2}(\phi(\mathcal{P}_i), c(\mu(\phi(\mathcal{P}_i)))) \right| \leq \varepsilon \operatorname{cost}_{ip^2}(\phi(\mathcal{P}), \mathcal{C}).$$

We conclude that

$$\begin{aligned}
&\left| \operatorname{cost}_{ip^2}(\phi(\mathcal{P}), \mathcal{C}) - \sum_{i=1}^{n'} \min_{c \in \mathcal{C}} \sum_{y \in \mathcal{S}_i} w(y) \cdot |\phi(y) - c|^2 - \mathfrak{D} \right| \\
\leq &\left| \operatorname{cost}_{ip^2}(\phi(\mathcal{P}), \mathcal{C}) - \sum_{\ell=1}^{n'} \operatorname{cost}_{ip^2}(\phi(\mathcal{P}_i), c(\mu(\phi(\mathcal{P}_i)))) \right| \\
&\quad + \left| \sum_{\ell=1}^{n'} \operatorname{cost}_{ip^2}(\phi(\mathcal{P}_i), c(\mu(\phi(\mathcal{P}_i)))) - \sum_{i=1}^{n'} \min_{c \in \mathcal{C}} \sum_{y \in \mathcal{S}_i} w(y) \cdot |\phi(y) - c|^2 - \mathfrak{D}_i \right| \\
\leq &\, 5\varepsilon \cdot \operatorname{cost}_{ip^2}(\phi(\mathcal{P}), \mathcal{C}).
\end{aligned}$$

Algorithm 6.3 summarizes the algorithm that we just analyzed. As the error is five times as high as we wish it to be, we call the method `Subdivide` with $\varepsilon/5$ as the precision parameter.

Theorem 6.2.6. *Let $\mathcal{P} \subset \boldsymbol{X}$ be a point set containing n points from a set $\boldsymbol{X}$. Let $\delta > 0$ be a constant. Let $\boldsymbol{V}$ be a Hilbert space with inner product $\langle \cdot, \cdot \rangle : \boldsymbol{V} \times \boldsymbol{V} \to \mathbb{R}$ and let $|\cdot| : \boldsymbol{V} \to \mathbb{R}$ be the norm induced by the inner product. Let $\kappa : \boldsymbol{X} \times \boldsymbol{X} \to \mathbb{F}$ be a valid kernel function with decomposition $\kappa(\chi, y) = \langle \phi(\chi), \phi(y) \rangle$ for all $\chi, y \in \boldsymbol{X}$ for a mapping $\phi : \boldsymbol{X} \to \boldsymbol{V}$. With*

probability $1-\delta$*, Algorithm 6.3 computes a coreset for the kernel* k*-means problem, i. e., a set* $\mathcal{M} := \{\mathcal{S}_1, \ldots, \mathcal{S}_{n'}\}$ *of* n' *subsets of* $\mathcal{P}$*, a weight function* $w : \cup_{i=1}^{n'} \mathcal{S}_i \to \mathbb{R}$ *and a constant* $\mathfrak{D}$ *such that*

$$\left| \operatorname{cost}_{ip^2}(\phi(\mathcal{P}), \mathcal{C}) - \sum_{i=1}^{n'} \min_{c \in \mathcal{C}} \sum_{y \in \mathcal{S}_i} w(x) \cdot |\phi(y) - c|^2 - \mathfrak{D} \right| \leq \varepsilon \cdot \operatorname{cost}_{ip^2}(\phi(\mathcal{P}), \mathcal{C})$$

for all center sets $\mathcal{C} \in \boldsymbol{V}$*. The number of subsets* n' *is* $n' \leq k^{\mathcal{O}(\varepsilon^{-2} \cdot \log \varepsilon^{-1})}$*,and the size of each* $\mathcal{S}_i$ *is* $\mathcal{O}(\delta^{-1}\varepsilon^{-2}n')$*.*

7 Probabilistic k-median clustering

Real world applications usually output data with measurement errors, or contain data that is inherently probabilistic, like estimations based on statistical investigations. Another example where probabilistic data occurs is record linkage where the data originates from different sources. Consider a database that stores our knowlege on a set of entities, and assume that we want to add information from a new data set. The new data set contains data on some of the entities. Some of the attributes are the same, but others are different (and this is the new information). There are no identifiers that determine which entity a record belongs to, so we have to guess how likely it is that two records belong to the same entity. This can be modeled by probabilistic data.

For example, assume that our data set describes people, but without a identification number. We want to join two data bases, and both contain several persons with the name John Smith. Then we can guess whether two of them are the same person by comparing the other attributes that are stored in the data base. We can express this by a probability.

If the joined attribute set of both data bases is a set $\mathcal{X}$ of multidimensional points, points in $\mathcal{X}$ have a different likelihood to be the underlying entity of an old and a new data set. In other words, each new record induces a discrete probability distribution over $\mathcal{X}$.

In the following, we go into more detail on the specific probabilistic clustering problem that is studied in this chapter. The results presented in this chapter are joint work with Christiane Lammersen and Christian Sohler and have partially appeared in [LSS12, LSS14].

Assigned and unassigned clustering. In this chapter, we study a probabilistic k-median problem where the input consists of such discrete probability distributions. The theoretical study of this type of clustering problem was initiated by Cormode and McGregor in [CM08]. They propose two kinds of probabilistic clustering problems. Both assume that the input consists of a set $\mathcal{V}$ of probabilistic points, which means that they are represented by a discrete probability distribution. This distribution specifies a probability for the event that the point appears at a certain location, but the probability is only nonzero for finitely many locations. Additionally, the model allows the event that the point does not appear at all. The finite set $\mathbf{X}$ is the union of all locations where any of the distributions has a positive support.

In the *unassigned* case, we assume that the clustering cost of a center set is determined by first performing the random experiment, then assigning each point to its nearest center and then computing the cost of this assignment. The objective is then to minimize the expected clustering cost. Consider the k-means problem in the Euclidean space $\mathbb{R}^d$, so $\mathbf{X}$

is a set from $\mathbb{R}^d$. The objective of the unassigned probabilistic k-means clustering is

$$\min_{C\in\mathbb{R}^d,|C|=k} \sum_{x\in\mathcal{V}} \sum_{y\in\mathbf{X}} \text{Prob}(x \text{ appears at } y) \cdot \min_{c\in C} ||y-c||^2.$$

As pointed out in [CM08], this problem is a weighted k-means problem where $\mathbf{X}$ is the input point set. The weight of an $y \in \mathbf{X}$ is defined as

$$p(y) := \sum_{x\in\mathcal{V}} \text{Prob}(x \text{ appears at } y),$$

which is the expected number of points that appear at each y. With this definition, the objective reduces to the weighted k-means objective

$$\min_{C\in\mathbb{R}^d,|C|=k} \sum_{y\in\mathbf{X}} p(y) \cdot \min_{c\in C} ||y-c||^2.$$

The same is true for the *unassigned Euclidean k-median problem*, which is defined verbatim except that the distances are not squared. So, both the unassigned Euclidean k-means problem and the unassigned Euclidean k-median problem directly reduce to their deterministic counterparts.

In the *assigned* case, we assume that the center that a point is assigned to is decided before the random experiment is conducted. The objective of the assigned version of the Euclidean k-means problem thus reads

$$\min_{C\in\mathbb{R}^d,|C|=k} \min_{\rho:\mathcal{V}\to C} \sum_{x\in\mathcal{V}} \sum_{y\in\mathbf{X}} \text{Prob}(x \text{ appears at } y) \cdot ||y-\rho(x)||^2.$$

As it turns out, this is another situation where (the weighted version of) Lemma 2.4.1 helps. Abbreviate Prob(x appears at y) by p_{xy}. Using $\mu_x := \sum_{y\in\mathbf{X}} p_{xy} y / \Big(\sum_{y\in\mathbf{X}} p_{xy}\Big)$, we can rewrite the objective to

$$\begin{aligned}
&\min_{C\in\mathbb{R}^d,|C|=k} \min_{\rho:\mathcal{V}\to C} \sum_{x\in\mathcal{V}} \sum_{y\in\mathbf{X}} p_{xy} \cdot ||y-\rho(x)||^2 \\
=&\min_{C\in\mathbb{R}^d,|C|=k} \min_{\rho:\mathcal{V}\to C} \sum_{x\in\mathcal{V}} \Big(\sum_{y\in\mathbf{X}} p_{xy} \cdot ||y-\mu_x||^2 + \Big(\sum_{y\in\mathbf{X}} p_{xy}\Big) \cdot ||\mu_x-\rho(x)||^2\Big) \\
=&\sum_{x\in\mathcal{V}} \sum_{y\in\mathbf{X}} p_{xy} \cdot ||y-\mu_x||^2 + \min_{C\in\mathbb{R}^d,|C|=k} \min_{\rho:\mathcal{V}\to C} \sum_{x\in\mathcal{V}} \Big(\sum_{y\in\mathbf{X}} p_{xy}\Big) \cdot ||\mu_x-\rho(x)||^2.
\end{aligned}$$

The first summand is independent of the solution and is therefore unimportant for the optimization. The second term is a weighted k-means problem. So, the assigned version of the Euclidean k-means problem also reduces to a (deterministic) weighted k-means problem. This observation is also made in [CM08]. Notice that the same argumentation holds for the k-means problem in inner product spaces as well. However, it does not necessarily carry over to the k-means problem in metric spaces that are not inner product spaces because Lemma 2.4.1 may not hold. Also, it does not carry over to the metric or even Euclidean k-median problem.

For the assigned version of the probabilistic k-median problem, Cormode and McGregor show that a $(2\alpha + 1)$-approximative solution can be computed if an α-approximation for the deterministic k-median problem is known. The reduction replaces each probability distribution by an α-approximation of its optimal weighted 1-median (where the probabilities are interpreted as weights). The approximative 1-medians are appropriately weighted and then clustered with the α-approximation to obtain a solution. Notice that this is similar to the k-means case, where the deterministic k-means instance is obtained by computing the optimal weighted 1-mean μ_x for each probabilistic point x. Yet, as Lemma 2.4.1 does not carry over to the k-median problem, the reduction does not provide the same quality as for the Euclidean k-means problem.

Overview on approximation algorithms in this model. By combining the above arguments with known algorithms for the deterministic case, Cormode and McGregor [CM08] achieve a $(1 + \varepsilon)$-approximation for the assigned and unassigned Euclidean k-means problem and for the unassigned Euclidean k-median problem, and they achieve a constant approximation for the assigned metric k-median problem.

They also consider the metric k-center problem which is to compute k centers such that the maximum distance of any input point to its closest center is minimized. The metric k-center problem is NP-hard [KH79a] and it is also NP-hard to find an α-approximation for $\alpha < 2$ according to a reduction from the dominating set problem [HN79]. On the other hand, 2-approximations for the discrete metric k-center problem were given by [Hs85] and [Gon85]. Cormode and McGregor give examples where intuitive strategies fail to produce good solutions for the (assigned or unassigned) probabilistic k-center problem. They give a bicriteria approximation for the unassigned metric k-center problem that computes a constant approximation with $2k$ centers.

Guha and Munagala [GM09] improve this result. They obtain a constant factor approximation for both the unassigned and assigned metric k-center problem, without relaxing the number of centers.

Related Work on Clustering Uncertain Data. We shortly review algorithms for uncertain clustering besides the approximation algorithms developed in the model by Cormode and McGregor. As we discussed in Chapter 2, Lloyd's algorithm is probably the most known algorithm for the k-means problem. Consequently, facing the need to cluster uncertain data, it is among the algorithms to be extended to this setting. Chau, Cheng, Kao and Ng [CCKN06] propose using Lloyd's algorithm with the change that points are assigned to centers according to their *expected* clustering cost. The resulting algorithm is called *UK-means algorithm*.

Ngai et al. [NKC+06] address the issue that computing the expected cost can be computationally expensive if the input points follow a complicated probability distribution. One of their techniques to speed the UK-means algorithm up is to use pruning techniques. This means that they identify points that change their cluster affiliation, but they find them without actually computing the distances.

Density based clustering is a popular clustering method for deterministic clustering. The most important algorithm for density based clustering is DBSCAN [EKSX96]. This algorithm has also been extended for handling uncertainty, see Kriegel and Pfeifle [KP05a, KP05b] and Xu and Li [XL08]. These are just examples for a rapidly growing field. For an overview on uncertain data mining, including clustering, see the survey by Aggarwal and Yu [AY09].

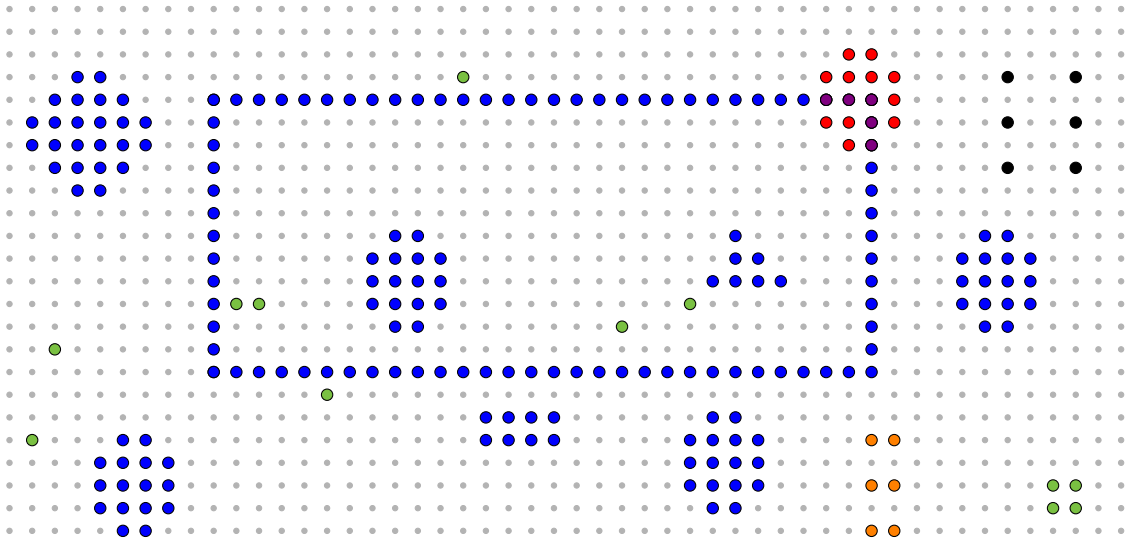


Figure 7.1: An example of a probabilistic clustering instance. The metric space is the Euclidean plane and $\mathcal{X}$ is a grid in the plane. The picture shows the support of twelve nodes (without the actual probabilities). There are eight blue nodes. The remaining four nodes are red, green, black and orange. Notice that nodes can overlap, like the red node does with one of the blue nodes.

7.1 A probabilistic k-median problem

In this section, we formally define the problem studied in this chapter and set the notation. Figure 7.1 depicts an example for a possible input to our probabilistic k-median problem. It illustrates some of the properties of such an input instance.

We have a metric space $(\mathbf{X}, D)$ consisting of a possibly infinite set $\mathbf{X}$ and a metric $D : \mathbf{X} \times \mathbf{X} \to \mathbb{R}$. In the metric space, $\mathcal{X} := \{x_1, \ldots, x_m\} \subset \mathbf{X}$ is a finite set of m possible locations. The input point set is a set $\mathcal{V}$ which consists of n probability distributions $v_1, \ldots, v_n : \mathcal{X} \to [0, 1]$. In order to distinguish between the points in $\mathcal{X}$ and the input, we use a special naming convention. By *points* or *locations* we mean the elements of $\mathcal{X}$. The input elements from $\mathcal{V}$ are called *nodes*.

Each node v_i is a mapping $v_i : \mathcal{X} \to [0, 1]$ and assigns a probability to each possible location. We use the abbreviation $p_{ij} := v_i(x_j)$ for all $i \in [n]$ and $j \in [m]$. The total

probability of a node is $p_i := \sum_{i=1}^{m} p_{ij}$, and we demand that $0 < p_i \leq 1$ for all $i \in [n]$. For the smallest occuring probability we use the abbreviation $p_{\min} = \min\{p_{ij} \mid i \in [n], j \in [m], p_{ij} > 0\}$. Additionally, the problem allows weights for the nodes, i. e., an additional mapping $w : \mathcal{V} \to \mathbb{R}^+$ that assigns a weight to each node.

There is one subtlety to the definition of the solution space of the k-means and k-median problem. If the distance function is induced by the Euclidean norm, then centers can usually be chosen from $\mathbb{R}^d$. If the distance function is a metric which is only defined on a finite set, then it makes sense to restrict the solution space. In Section 2.2, we considered the metric k-median problem where centers can only be chosen from the input point set itself. This is a common choice, yet we also want to allow a finite set that is different from the input point set. Thus, we do not specify the set of candidates and allow it to be a finite or infinite set from the metric space.

The formulation with a finite set of candidates is often used when the k-median problem is studied alongside the *facility location problem* and in a metric setting. An instance for the facility location problem consists of a metric distance function, a set of facilities, a set of clients and opening costs for the facilities. The facilities correspond to the centers in the k-median problem, and the clients correspond to the points. The problem is to choose a subset of the facilities such that the cost of connecting each client to its closest facility plus the opening costs of the opened facilities is minimized. So instead of limiting the number of centers, the facility location problem introduces costs for opening facilities.

In the following definition, we use a finite set of candidates for the metric case and $\mathbb{R}^d$ in the Euclidean case. Notice that the centers are (deterministic) points from the metric space, so the centers are not of the same type as the input elements (which are probabilistic points). However, they can be interpreted as probabilistic points where the probability mass is concentrated at one point.

Definition 7.1.1 (Probabilistic k-Median Problem, [CM08])**.** *Let $(\boldsymbol{X}, D)$ be a metric space. Given $k \in \mathbb{N}$, a set $\mathcal{V}$ of n nodes $v_i : \mathcal{X} \to [0,1]$ with $p_i \leq 1$ for all $i \in [n]$, a finite set of κ possible center locations $\mathcal{L} \subseteq \boldsymbol{X}$ and a positive weight function $w : \mathcal{V} \to \mathbb{R}_{>0}$, the (assigned)* weighted probabilistic metric k-median clustering problem *for $\mathcal{V}$ is to find a set $\mathcal{C} := \{c_1, \ldots, c_k\} \subseteq \mathcal{L}$ of k cluster centers and an assignment $\rho : \mathcal{V} \to \mathcal{C}$ minimizing the expected weighted k-median-clustering cost*

$$\operatorname{ecost}_D(\mathcal{V}, \mathcal{C}, \rho) := \sum_{i=1}^{n} w(v_i) \sum_{j=1}^{m} p_{ij} \cdot D(x_j, \rho(v_i)).$$

The unweighted *problem results from setting $w(v_i) = 1$ for all v_i. We denote the cost of an optimal clustering by $\operatorname{ecost}_D^{*,k}(\mathcal{V})$ and the total weight by $W := \sum_{v_i \in \mathcal{V}} w(v_i)$ (we assume that the problem is scaled such that all weights are at least one). The definition is verbatim for the Euclidean case, except that $\mathcal{L} = \boldsymbol{X} = \mathbb{R}^d$, that D is the distance function induced by the Euclidean norm, and that $\mathcal{L}$ is in particular not finite.*

The deterministic k-median clustering problem is a special case of the probabilistic problem, where $m = n$ and, for each node v_i, we have $p_{ii} = 1$ and $p_{ij} = 0$ for all $j \neq i$. As

we mentioned, the deterministic k-median problem is often defined with $\mathcal{L} = \mathcal{V}$. Notice that if we have one $j \in [m]$ with $p_{ij} = p_i$ and hence $p_{ij'} = 0$ for all $j' \neq j$ for all nodes v_i, then the probabilistic k-median clustering can be immediately reduced to a weighted deterministic k-median clustering.

Modelling assumptions. For the metric k-median problem, we assume that the input contains a way to evaluate the distance between two points in constant time. Otherwise, the running times stated in this section for the k-median problem have to be be multiplied by the time needed to compute a distance. Notice that in the Euclidean case, computing a distance takes time $\Theta(d)$.

Depending on the way that a metric space is given, the input size is different. When all pairwise distances are given for all locations in $\mathcal{X}$, the input size is at least $|\mathcal{X}|^2 = m^2$. Even if it is not, the input has to contain a number for each location in $\mathcal{X}$ and each node in $\mathcal{V}$ if the node has a positive probability. We do not elaborate on this further and just state running times in terms of n and m (and additional parameters when appropriate).

Assumptions on the input without loss of generality. At some points, it will be convenient to have additional assumptions on the input. First, it is helpful if all nodes have the same total realization probability, i. e., p_i is the same for all $i \in [n]$. This can be achieved without changing the clustering cost as long as we allow weights.

For example, assume that we want to have $p_i = 1$ for all $i = 1, \ldots, n$. Let v_i be a node with $0 < p_i < 1$. By multiplying each p_{ij} with $1/p_i$ and at the same time, multiplying $w(v_i)$ by p_i, the objective function is not changed, but the total probability of v_i is changed such that $p_i = 1$ now holds.

Second, we like to have weights that are at least one. Notice that if all weights are scaled by the same factor, then the costs of all solutions are scaled by this factor. Thus, we can scale the weights, solve the problem on the scaled input and use the solution for the original input. The drawback is that the total weight of the scaled input is higher. If we want to achieve that all weights are at least one, we have to multiply the instance by $1/w_{\min}$, where $w_{\min}$ is the smallest occuring weight. The total weight is thus increased from W to $W/w_{\min}$.

Reducing weighted to unweighted inputs. In the following, we will repeatedly wish to use unweighted algorithms. It is therefore helpful to observe that we can approximate the weighted version by an appropriate unweighted version of the problem. In the unweighted version, the nodes do not have a total probability of one, but they all have the same total probability, which is also fine. The following lemma resembles Lemma 4.2.1 in Section 4.2. Chen [Che09] also uses this technique and attributes it to Mettu and Plaxton [MP04].

Lemma 7.1.2. *Let $\mathcal{V}$ be a set of nodes $v_i : \mathcal{X} \to [0, 1]$ with $p_i = 1$ for all $i \in [n]$ and with weights $w : \mathcal{V} \to \mathbb{R}^+$ that satisfy $w(v_i) \geq 1$ for $i \in [n]$. Let ε be a constant with $0 \leq \varepsilon \leq 1$. We can construct a set $\bar{\mathcal{V}}$ containing $\lfloor w(v_i)/(p_{\min}\varepsilon) \rfloor$ nodes $\bar{v}_{i\ell}$ for each $v_i \in \mathcal{V}$, each with*

total probability $p_{\min} \cdot \varepsilon$*, such that the following holds. For each subset* $\mathcal{T} \subseteq \mathcal{V}$ *and the corresponding subset* $\bar{\mathcal{T}} := \{\bar{v}_{i\ell} \mid v_i \in \mathcal{T}, \ell = 1, \dots \lfloor w(v_i)/(p_{\min}\varepsilon)\rfloor\}$ *of* $\bar{\mathcal{V}}$*, and for each* $\mathcal{C} \subset \mathcal{L}$ *with* $|\mathcal{C}| = k$*, it holds that*

$$\left|\mathrm{ecost}_{D,w}(\mathcal{T}, \mathcal{C}, \rho) - \mathrm{ecost}_D(\bar{\mathcal{T}}, \mathcal{C}, \bar{\rho})\right| \leq \varepsilon \cdot \mathrm{ecost}_{D,w}(\mathcal{T}, \mathcal{C}, \rho)$$

where $\bar{\rho}(\bar{v}_{i\ell}) := \rho(v_i)$ *for all* $i \in [n]$ *and* $\ell = 1, \dots, \lfloor w(v_i)/(p_{\min}\varepsilon)\rfloor$*. This holds for the Euclidean and for the metric case.*

Proof. $\bar{\mathcal{V}}$ contains $\lfloor w(v_i)/(p_{\min}\varepsilon)\rfloor$ nodes $\bar{v}_{i\ell}$ for each node v_i. Each $\bar{v}_{i\ell}$ has weight one. For each $x_j \in \mathcal{X}$, we set the probability that $\bar{v}_{i\ell}$ is realized to $\bar{p}_{ij} := p_{ij} \cdot p_{\min} \cdot \varepsilon$ for all $\ell = 1, \dots, \lfloor w(v_i)/(p_{\min}\varepsilon)\rfloor$. The total realization probability of $v_{i\ell}$ thus is $p_{\min} \cdot \varepsilon$. For each subset $\mathcal{T}$ of $\mathcal{V}$, we define $\bar{\mathcal{T}}$ accordingly, so $\bar{\mathcal{T}}$ consists of $\lfloor w(v_i)/(p_{\min}\varepsilon)\rfloor$ copies of $\bar{v}_i$ for all $v_i \in \mathcal{T}$. Notice that

$$w(v_i) \geq \left\lfloor \frac{w(v_i)}{p_{\min} \cdot \varepsilon} \right\rfloor \cdot p_{\min} \cdot \varepsilon \geq \left(\frac{w(v_i)}{p_{\min} \cdot \varepsilon} - 1 \right) \cdot p_{\min} \cdot \varepsilon = w(v_i) - p_{\min} \cdot \varepsilon \geq (1 - \varepsilon) \cdot w(v_i)$$

where the last inequality holds because the weights are at least one and ε and $p_{\min}$ are at most one. This implies that

$$\begin{aligned}
&\mathrm{ecost}_{D,w}(\mathcal{T}, \mathcal{C}, \rho) = \sum_{v_i \in \mathcal{T}} w(v_i) \sum_{j=1}^{m} p_{ij} \cdot D(x_j, \rho(v_i)) \\
&\leq \sum_{v_i \in \mathcal{T}} \left\lfloor \frac{w(v_i)}{p_{\min} \cdot \varepsilon} \right\rfloor \sum_{j=1}^{m} p_{ij} p_{\min} \cdot \varepsilon \cdot D(x_j, \rho(v_i)) + \varepsilon \cdot \sum_{v_i \in \mathcal{T}} w(v_i) \sum_{j=1}^{m} p_{ij} \cdot D(x_j, \rho(v_i)) \\
&= \sum_{v_i \in \mathcal{T}} \sum_{z=1}^{\lfloor w(v_i)/(p_{\min} \cdot \varepsilon)\rfloor} 1 \cdot \sum_{j=1}^{m} \bar{p}_{ij} \cdot D(x_j, \rho(v_i)) + \varepsilon \cdot \mathrm{ecost}_{D,w}(\mathcal{T}, \mathcal{C}, \rho) \\
&= \sum_{v_{i,\ell} \in \bar{\mathcal{T}}} \sum_{j=1}^{m} \bar{p}_{ij} \cdot D(x_j, \rho(v_i)) + \varepsilon \cdot \mathrm{ecost}_{D,w}(\mathcal{T}, \mathcal{C}, \rho) \\
&= \mathrm{ecost}_D(\bar{\mathcal{T}}, \mathcal{C}, \bar{\rho}) + \varepsilon \cdot \mathrm{ecost}_{D,w}(\mathcal{T}, \mathcal{C}, \rho)
\end{aligned}$$

and a similar calculation gives that $\mathrm{ecost}_D(\bar{\mathcal{T}}, \mathcal{C}, \bar{\rho}) \leq \mathrm{ecost}_{D,w}(\mathcal{T}, \mathcal{C}, \rho)$. That proves the statement. □

7.2 Probabilistic coresets

Now, we introduce coresets for the probabilistic k-median-clustering problem. As the storage size of a probabilistic point set is influenced by the number of probabilistic points and by the size of the individual probability distributions, we are interested in coresets where both parameters are small. Notice that the following formal definition of coresets is relatively general and allows that $\mathcal{L} \subsetneq \mathcal{X}$. In this case, the convention would be to call the coreset a *weak* coreset. However, this only happens when we consider a probabilistic k-median problem where $\mathcal{L}$ is actually smaller than $\mathcal{X}$ by definition of the problem itself.

Definition 7.2.1 (Coresets for Probabilistic k-Median Problem)**.** *Given $k \in \mathbb{N}$, a set $\mathcal{V}$ of n nodes $v_i : \mathcal{X} \to [0,1]$, a finite set of κ possible center locations $\mathcal{L} \subseteq \boldsymbol{X}$, a positive weight function $w : \mathcal{V} \to \mathbb{R}_{\geq 0}$ and a precision parameter ε, $0 < \varepsilon \leq 1$, let $\mathcal{U} := \{u_1, \ldots, u_s\}$ be a set of mappings $u_i : \mathcal{X} \to \mathbb{R}^+$ annotated with a positive weight function $w' : \mathcal{U} \to \mathbb{R}_{\geq 0}$. The set $\mathcal{U}$ is called $(1+\varepsilon)$-coreset of $\mathcal{V}$ for the probabilistic metric k-median-clustering problem if, for each $\mathcal{C} \subseteq \mathcal{L}$ of size $|\mathcal{C}| = k$, we have*

$$\left| \min_{\rho:\mathcal{U}\to\mathcal{C}} \operatorname{ecost}_{D,w'}(\mathcal{U}, \mathcal{C}, \rho) - \min_{\rho:\mathcal{V}\to\mathcal{C}} \operatorname{ecost}_{D,w}(\mathcal{V}, \mathcal{C}, \rho) \right| \leq \varepsilon \cdot \min_{\rho:\mathcal{V}\to\mathcal{C}} \operatorname{ecost}_{D,w}(\mathcal{V}, \mathcal{C}, \rho).$$

The definition is verbatim for the Euclidean case, except that $\mathcal{L} = \mathbb{R}^d$. If the nodes in $\mathcal{V}$ are weighted by a positive weight function $w : \mathcal{V} \to \mathbb{R}_{\geq 0}$, then $\operatorname{ecost}_D(\mathcal{V}, \mathcal{C}, \rho)$ is replaced by $\operatorname{ecost}_{D,w}(\mathcal{V}, \mathcal{C}, \rho)$.

In the following, for all u_i belonging to the coreset that we compute, we denote the probability that u_i is realized at x_j by p'_{ij} for all $j \in [m]$. Further, we denote the total probability that u_i is realized at all by $p'_i := \sum_{j=1}^{m} p'_{ij}$. The p'_i can always be scaled to one if necessary by scaling the weights of the nodes accordingly.

A coreset for the probabilistic k-median problem consists of nodes, i. e., of probabilistic points. A node is a mapping from $\mathcal{X}$ to $\mathbb{R}^+$, and to represent it, we need to store its support, i. e., all points with positive probability. The representation size of the coreset is small if it contains few nodes, and if these nodes have few points in their support. Thus, we need a method to thin out the support of nodes while keeping their clustering cost approximately the same for all choices of centers. For this, we use the coreset construction by Chen [Che09] and apply it to the probability distributions of all coreset nodes. Notice that the metric coreset construction by Chen works in the general setting where the set of center candidates is not equal to the input point set, although Chen does not note this explicitly. We state this and the result for the Euclidean case here.

Theorem 7.2.2 ([Che09])**.** *Let $(\boldsymbol{X}, D)$ be a metric space, let $\mathcal{P} \subset \boldsymbol{X}$ be a set of n points and let $\mathcal{L} \subset \boldsymbol{X}$ be a set of possible center locations. Let $k \in \mathbb{N}$ be an integer, $\varepsilon \in \mathbb{R}$ with $0 < \varepsilon < 1$ be a precision parameter and let $\delta \in \mathbb{R}$ with $0 < \delta < 1$ be an error probability parameter.*

For $|\mathcal{L}| = \kappa$, one can compute a weighted subset $\mathcal{Z} \subset \mathcal{P}$ in time $\mathcal{O}(nk \log(1/\delta))$ of size $\mathcal{O}(k\varepsilon^{-2}(k\log(\kappa)+\log(1/\delta))\log(n))$, such that $\mathcal{Z}$ is a (k,ε)-coreset of $\mathcal{P}$ for the deterministic metric k-median clustering problem.

For $\boldsymbol{X} = \mathbb{R}^d$, one can compute a (k,ε)-coreset $Z \subset P$ for P in time $\mathcal{O}(ndk \log \delta^{-1})$ of size $\mathcal{O}(\varepsilon^{-2}k \log n(dk \log \varepsilon^{-1} + k \log k + k \log\log n + \log \delta^{-1}))$ for the deterministic Euclidean k-median clustering problem.

Both statements hold with probablity $1 - \delta$.

The following lemma shows that we can thin out the probability distribution of a probabilistic point, i. e., we can reduce the number of possible locations of the probabilistic point without losing too much information. This is done by grouping the locations according to their probablities and computing a deterministic coreset for each group. This works for the

metric and Euclidean case if we use the corresponding construction for the computation of the deterministic coreset.

Lemma 7.2.3. *Let $v_i \in \mathcal{V}$ be a node $v_i : \mathcal{X} \to [0,1]$ with weight $w(v_i)$, let $\mathcal{L} \subseteq \boldsymbol{X}$ be a set of possible center locations and let ε with $0 \leq \varepsilon \leq 1$ be a precision parameter. The computation of a $v_i' : \mathcal{X} \to [0,1]$ and a set $\mathcal{Z}_i \subset \mathcal{X}$ with*

$$\left| \sum_{x_j \in \mathcal{Z}_i} w(v_i) p'_{ij} \cdot D(x_j, c) - \sum_{x_j \in \mathcal{X}} w(v_i) p_{ij} \cdot D(x_j, c) \right| \leq \varepsilon \sum_{x_j \in \mathcal{X}} w(v_i) p_{ij} \cdot D(x_j, c)$$

for all centers $c \in \mathcal{L}$ can be reduced to computing $(1+\varepsilon)$-coresets of sets of at most m points for the (unweighted) deterministic 1-median problem. The number of constructions of deterministic coresets is bounded by m and by $\mathcal{O}(\varepsilon^{-1} \log(1/p_{\min}))$.

Proof. Since we look at a single node, the weight is just a factor that is the same for v_i and its sparse representation v_i'. In particular, it holds that

$$\left| \sum_{x_j \in \mathcal{Z}_i} w(v_i') p'_{ij} \cdot D(x_j, c) - \sum_{x_j \in \mathcal{X}} w(v_i) p_{ij} \cdot D(x_j, c) \right| \leq \varepsilon \sum_{x_j \in \mathcal{X}} w(v_i) p_{ij} \cdot D(x_j, c)$$

$$\Leftrightarrow \left| \sum_{x_j \in \mathcal{Z}_i} p'_{ij} \cdot D(x_j, c) - \sum_{x_j \in \mathcal{X}} p_{ij} \cdot D(x_j, c) \right| \leq \varepsilon \sum_{x_j \in \mathcal{X}} p_{ij} \cdot D(x_j, c),$$

so we can ignore the weight of v_i for the proof.

Recall that for a given center set, a node v_i is assigned to one center $c_i \in \mathcal{C}$. The nodes contribution to the cost function can then be interpreted as a weighted 1-median clustering cost of the points in the support of v_i, where the center is c_i. The idea now is to compute a coreset for the support of v_i that approximates this 1-median cost for all possible centers.

The first step is to round the probabilities such that we have groups of points with the same probability value. We round each realization probability p_{ij} down to $\bar{p}_{ij} := p_{\min} \cdot (1+\varepsilon/2)^{\ell}$ where ℓ is the largest natural number including 0 such that the new realization probability $\bar{p}_{ij}$ is not larger than the original realization probability p_{ij}. That means that $\bar{p}_{ij} \leq p_{ij}$, and $\bar{p}_{ij} \geq p_{ij}/(1+\varepsilon/2)$. The latter implies that $\bar{p}_{ij} \geq p_{ij}/(1+\varepsilon/2) \geq (1-\varepsilon/2) \cdot p_{ij}$ because $0 \leq \varepsilon \leq 1$. Thus, the rounding cannot increase the clustering cost of v_i, and it decreases the clustering cost of v_i by at most an $(\varepsilon/2)$-fraction. The rounding gives us groups of points with the same probability value as desired.

Notice that the original p_{ij} and also the new values $\bar{p}_{ij}$ are numbers from $[0,1]$, and in particular that they cannot be assumed to be integers. This means that we cannot use weighted coreset computations which assume integer weights without additional precautions. Instead of additional rounding and a single weighted coreset construction (with a total weight that depends on $p_{\min}$ and thus leads to a polylogarithmic dependency on $p_{\min}$ when using the construction by Chen), we use multiple unweighted coreset constructions, one for each of the $\mathcal{O}(\varepsilon^{-1} \log(1/p_{\min}))$ groups.

For all exponents ℓ that occur, we define the set $\mathcal{X}_{i,\ell} \subset \mathcal{X}$ as the subset of points where v_i has probability $p_{\min} \cdot (1+\varepsilon/2)^\ell$. This gives us at most $\mathcal{O}(\varepsilon^{-1}\log(1/p_{\min}))$ sets for each node. Notice that the number of sets is also bounded by m. We proceed for each $\mathcal{X}_{i,\ell}$ in the same way. We call a $(1+\varepsilon/2)$-coreset construction for the deterministic 1-median problem to compute a weighted subset $\mathcal{Z}_{i,\ell} \subseteq \mathcal{X}_{i,\ell}$ with positive weight function $w' : \mathcal{Z}_{i,\ell} \to \mathbb{R}_{\geq 0}$ such that, for each center $c \in \mathcal{L}$, we have

$$\left| \sum_{x_j \in \mathcal{Z}_{i,\ell}} w'(x_j) \cdot D(x_j, c) - \sum_{x_j \in \mathcal{X}_{i,\ell}} D(x_j, c) \right| \leq (\varepsilon/2) \cdot \sum_{x_j \in \mathcal{X}_{i\ell}} D(x_j, c).$$

Now multiply this inequality with $\bar{p}_{ij} = p_{\min} \cdot (1+\varepsilon/2)^\ell$ and set $p'_{ij} := w'(x_j) \cdot \bar{p}_{ij}$. Then we have

$$\left| \sum_{x_j \in \mathcal{Z}_{i\ell}} p'_{ij} \cdot D(x_j, c) - \sum_{x_j \in \mathcal{X}_{i\ell}} \bar{p}_{ij} \cdot D(x_j, c) \right| \leq (\varepsilon/2) \cdot \sum_{x_j \in \mathcal{X}_{i\ell}} \bar{p}_{ij} \cdot D(x_j, c).$$

Then let $\mathcal{Z}_i$ be the union of all $\mathcal{Z}_{i\ell}$. Since the $\mathcal{Z}_{i\ell}$ form a partitioning of $\mathcal{Z}_i$ and the $\mathcal{X}_{i\ell}$ form a partitioning of $\mathcal{X}$, we get by the triangle inequality that

$$\left| \sum_{x_j \in \mathcal{Z}_i} p'_{ij} \cdot D(x_j, c) - \sum_{x_j \in \mathcal{X}} \bar{p}_{ij} \cdot D(x_j, c) \right| \leq (\varepsilon/2) \cdot \sum_{x_j \in \mathcal{X}} \bar{p}_{ij} \cdot D(x_j, c).$$

Recall that $\bar{p}_{ij} \leq p_{ij} \leq (1+\varepsilon/2) \cdot \bar{p}_{ij}$ by definition. Thus, we get that

$$\begin{aligned}
&\left| \sum_{x_j \in \mathcal{Z}_i} p'_{ij} \cdot D(x_j, c) - \sum_{x_j \in \mathcal{X}} p_{ij} \cdot D(x_j, c) \right| \\
\leq &\left| \sum_{x_j \in \mathcal{Z}_i} p'_{ij} \cdot D(x_j, c) - \sum_{x_j \in \mathcal{X}} \bar{p}_{ij} \cdot D(x_j, c) \right| + (\varepsilon/2) \cdot \sum_{x_j \in \mathcal{X}} \bar{p}_{ij} \cdot D(x_j, c) \\
\leq &\varepsilon \cdot \sum_{x_j \in \mathcal{X}} \bar{p}_{ij} \cdot D(x_j, c).
\end{aligned}$$

Thus $\mathcal{Z}_i$ is a $(1+\varepsilon/2)$-coreset for the probabilistic k-median clustering for the point set with rounded realization probabilities, and a $(1+\varepsilon)$-coreset for the original support of v_i. □

We combine Lemma 7.2.3 with the coreset results for the metric and Euclidean k-median problem by Chen in Theorem 7.2.2. Chen's construction is probabilistic, so we need to make sure that all coreset computations succeed simultaneously with constant probability. The number of groups is bounded by m, so to ensure a failure probability of at most δ/n for each node, it suffices to demand a failure probability of at most δ/nm in each coreset computation by the Union Bound. By applying the Union Bound again, we get that the probability that a coreset computation fails for any node is bounded by δ.

Figure 7.2: An example for morphing weighted point sets. A number at a node is its weight. A number at an edge is the amount that is moved along this edge. The overall movement cost is slightly smaller in the right example. The earth mover distance is the cost of the cheapest way to morph one point set into the other.

In the metric case, the time to compute a $(1+\varepsilon)$-coreset for one of the groups of a node takes $\mathcal{O}(m \log \frac{nm}{\delta})$ time by Theorem 7.2.2. Computing the coresets for all groups of all n nodes with a failure probability of $\delta/(nm)$ takes $\mathcal{O}(\varepsilon^{-1}nm \cdot (\log \frac{nm}{\delta}) \log(p_{\min}^{-1}))$ time because there are at most $\mathcal{O}(\varepsilon^{-1} \log(p_{\min}^{-1}))$ groups for each node. The size of each coreset is $\mathcal{O}(\varepsilon^{-2}(\log \kappa \frac{nm}{\delta}) \log m)$, so the size of the union of all groups for one node has size $\mathcal{O}(\varepsilon^{-3}(\log \kappa \frac{nm}{\delta})(\log m)(\log p_{\min}^{-1}))$.

In the Euclidean case, computing a $(1+\varepsilon)$ coreset for one group takes $\mathcal{O}(md \log \frac{nm}{\delta})$ time, so computing all groups of all nodes takes can be done in $\mathcal{O}(nmd\varepsilon^{-1}(\log \frac{nm}{\delta})(\log(p_{\min}^{-1})))$ time. The size of one coreset is $\mathcal{O}(\varepsilon^{-2} \log m \cdot (d \log \varepsilon^{-1} + \log\log m + \log \frac{nm}{\delta}))$, so the size of the union of all coresets for one node has size $\mathcal{O}(\varepsilon^{-3} \log m \cdot (d \log \varepsilon^{-1} + \log\log m + \log \frac{nm}{\delta})(\log(p_{\min}^{-1})))$.

The reduction can be used as a preprocessing step before applying an approximation algorithm or computing a coreset in order to reduce in order to reduce the input size that needs to be processed.

7.3 The assigned metric k-median problem

We show that it is possible to reduce the assigned metric k-median problem to a deterministic metric k-median problem. Above, we considered several reductions from probabilistic clustering problems to their deterministic counterparts that were observed by Cormode and McGregor [CM08]. Their work includes a reduction of the assigned metric k-median problem to a deterministic metric k-median problem. However, the reduction does not preserve the approximation guarantee. A deterministic α-approximation yields a $(2\alpha+1)$-approximation for the assigned metric k-median problem. We look at a different reduction that preserves the approximation guarantee. The key idea is to change the metric space, i. e., in order to solve the assigned metric k-median problem we define a deterministic metric k-median problem in a different metric space.

The metric of our choice is the *earth mover distance (EMD)*. Imagine that a weighted point set is morphed into a different point set of the same weight. This is done by shifting

the weight of the points to the second point set. It is allowed to split the weights, as long as every bit of the weight of a point is moved to some point in the target point set, and as long as the target points are not overloaded. The cost of moving a part of the weight of a point is proportional to the distance that it is moved. An illustration from [RTG00] fitting the name 'earth mover' distance is the following. A mass of earth is spread according to the weights of the first point set, and there are holes which have a capacity according to the second weighted point set. Then the earth mover distance is the amount of work to fill the holes with the earth. Figure 7.2 shows an example in the Euclidean plane, and the following definition defines the earth mover distance formally.

Definition 7.3.1. *Let $\mathcal{X}$ be a finite set and $D : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^+$ be a metric. Let $f, g : \mathcal{X} \to [0,1]$ be two mappings that satisfy $\sum_{x\in\mathcal{X}} f(x) = \sum_{x\in\mathcal{X}} g(x)$. We say that a mapping $\rho : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\geq 0}$* morphs *$f$ into g if it satisfies*

$$\sum_{y\in\mathcal{X}} \rho(x,y) = f(x) \text{ for all } x \in \mathcal{X} \text{ and } \sum_{x\in\mathcal{X}} \rho(x,y) = g(y) \text{ for all } y \in \mathcal{X}.$$

The cost of ρ is defined as $\mathfrak{M}(f,g,\rho) := \sum_{x\in\mathcal{X}} \sum_{y\in\mathcal{X}} \rho(x,y) \cdot D(x,y)$. The earth mover distance *$EMD(f,g)$ is the minimum cost of a mapping that morphs f into g.*

Recall that we can assume (or enforce by scaling the weights and probabilities) that all nodes have the same total probability. This ensures that we can calculate the earth mover distance between all pairs of nodes. For any c, the function set $\{f : \mathcal{X} \to [0,1] \mid \sum_{x\in\mathcal{X}} f(x) = c\}$ forms a metric space with the earth mover distance (see for example [RTG00], Appendix A). We can interpret the nodes to be elements of this metric space.

The elements of this metric space are mappings, but we want to compute centers that are points from the original Euclidean space. This is why we need the flexibility to specify a set of center candidates in the definition of the k-median problem. If we could not differentiate between the elements of the metric space that are to be clustered and the elements that are centers, then the resulting centers would be distributions. With the possibility to specify a candidate set, we do not have this problem.

Recall that $\mathcal{L}$ is the original candidate set for the centers, and let c be a center from $\mathcal{X}$. If we define an artificial node v_c with $v_c(c) = 1$ and $v_c(x_j) = 0$ otherwise, then the earth mover distance between any node and v_c is just the expected clustering cost of the node with center c. So we want to compute k centers from the set of candidates $\{v_c \mid c \in \mathcal{L}\}$ where $\mathcal{L} \subset \mathcal{X}$ is the set of candidates given in our input.

With this in mind, we can define an instance for the deterministic metric k-median problem such that an α-approximate solution for the instance yields an α-approximate solution for the assigned k-median problem. In order to approximately solve this instance, we need to apply an approximation algorithm for the deterministic problem that is able to handle candidate sets for the center sets that are not equal to the input point set. Additionally, we need to be able to actually compute the earth mover distance.

The latter can be done efficiently since the task to compute the earth mover distance can be interpreted as a *Hitchcock transportation problem*, which is a special case of the

minimum cost flow problem. The minimum cost flow problem can for example be solved by an algorithm due to Orlin [Orl93] which has a running time of $\mathcal{O}(m^3 \log m)$ for an input with m points (recall that there are m possible realization points in $\mathcal{X}$). For an introduction to minimum cost flow problems and different algorithms, see the introduction of [Orl93] or Chapter 10 in [AMO93]. Recall that we assume that evaluating the distance between two points in the metric space in the input can be done in constant time, otherwise, this would be an additional factor.

Theorem 7.3.2. *Let a set of nodes $\mathcal{V}$, a number $k \in \mathbb{N}$, a finite set of κ possible center locations $\mathcal{L} \subseteq \boldsymbol{X}$ and a precision parameter ε, $0 < \varepsilon \leq 1$, be given. An α-approximation for the assigned metric k-median problem on this instance can be computed by applying an α-approximation algorithm for the unweighted deterministic metric k-median problem to an instance with n nodes and κ center locations in a different metric space. The running time is bounded by $T(n, \kappa, k) \cdot M$ where $T(n, \kappa, k)$ is the running time of the deterministic algorithm and $M \in \mathcal{O}(m^3 \log m)$.*

If the input is weighted with $w : \mathcal{V} \to [1, \infty)$ with $w(v_i) \geq 1$ for all $i \in [n]$ and total weight W, then the running time increases to $T(W, \kappa, k) \cdot M$ or $T(W/(p_{\min} \cdot \varepsilon), \kappa, k) \cdot M$ depending on whether the deterministic algorithm can handle weighted inputs or not. If it cannot, then the approximation guarantee worsens to $\alpha(1 + \varepsilon)$.

Proof. We construct a deterministic k-median instance. The metric space is the function set $\{f : \mathcal{X} \to [0, 1] \mid \sum_{x \in \mathcal{X}} f(x) = 1\}$ with the earth mover distance as the metric distance measure. The set of center candidates is $\{v_c \mid c \in \mathcal{L}\}$ where v_c is defined for all $c \in \mathcal{L}$ by $v_c(c) = 1$ and $v_c(x_j) = 0$ for $x_j \neq c$. The input point set consists of all nodes in $\mathcal{V}$, interpreted as elements of the metric space. The number of centers and the weights are as in the probabilistic input. We compute an α-approximate solution for the instance with the deterministic metric k-median algorithm. During the algorithm, the EMD is computed by the minimum cost flow algorithm due to Orlin [Orl93].

For a node $v_i \in \mathcal{V}$ and a center v_c, the earth mover distance is given by the only mapping that morphs v_i into v_c, which is given by $\rho(x_j, c) = p_{ij}$ for $j \in [m]$ and $\rho(x_j, x_{j'}) = 0$ for $x_{j'} \neq c$. The cost of this mapping is

$$\sum_{j=1}^{m} p_{ij} \cdot D(x_j, c)$$

which is the expected clustering of v_i with c. Thus, the computed α-approximate solution induces an α-approximate solution for the assigned metric k-median problem.

If the input is weighted and the deterministic algorithm can handle weights, we include them as part of the input. If the deterministic algorithm cannot handle weights, then we replace the instance by an unweighted instance with $W/(p_{\min} \cdot \varepsilon)$ nodes by Lemma 7.1.2. □

Notice that the minimum cost flow computation can be sped up by first reducing the support of all nodes as described in Section 7.2. We discuss this in more detail below when stating the coreset result.

Theorem 7.3.2 assumes that the deterministic α-approximation is able to handle candidate sets for the centers that are not the input point set. Luckily, this assumption is not uncommon. For example, the approximation algorithms due to Arya et al. [AGK+04] and Li and Svensson [LS13] both work on instances with separate center candidate sets and input point sets. We use the result by Li and Svensson, which gives the currently best known approximation guarantee for the metric k-median problem.

Theorem 7.3.3 ([LS13])**.** *Let $\boldsymbol{X}$ with $D : \boldsymbol{X} \times \boldsymbol{X} \to \mathbb{R}^+$ be a metric space, let $\varepsilon > 0$ be a constant. There exists an algorithm which computes a $(1+\sqrt{3}+\varepsilon)$-approximate solution to the metric k-median problem for any instance consisting of an integer k, a set of possible centers $\mathcal{L} \subseteq \boldsymbol{X}$ and a set of points $\mathcal{P} \subseteq \boldsymbol{X}$. The running time of the algorithm is polynomial in $|\mathcal{P}|$, $|\mathcal{L}|$ and k.*

The result is stated for the unweighted k-median problem, so we use Lemma 7.1.2 for weighted inputs. Notice that an instance is implicitly a weighted instance if the total probabilities of the points are different because scaling them such that they become equal requires weights.

Corollary 7.3.4. *For instances where $p_i = 1$ for all nodes v_i, there exists a $(1 + \sqrt{3} + \varepsilon)$-approximation for the unweighted or weighted assigned metric k-median problem. The running time is polynomial in $|\mathcal{X}|$, $|\mathcal{L}|$, k and also in $|\mathcal{V}|$ or $W/(p_{\min} \cdot \varepsilon)$, respectively (where W is the total weight and we assume $w(v_i) \geq 1$ for all $i \in [n]$).*

We can also use the reduction in order to compute a coreset for the assigned metric k-median problem. For this, we use the coreset construction by Chen. It works for weighted inputs, too. In the following theorem, we elaborate a bit more on the running time reduction when preprocessing the nodes first.

Theorem 7.3.5. *Given a set of nodes $\mathcal{V}$, a weight function $w : \mathcal{V} \to \mathbb{R}^+$ with $w(v_i) \geq 1$ for all $i \in [n]$ and total weight W, a number $k \in \mathbb{N}$, a finite set of κ possible center locations $\mathcal{L} \subseteq \boldsymbol{X}$ and a precision parameter ε, $0 < \varepsilon \leq 1$, a $(1+\varepsilon)$-coreset for the assigned metric k-median problem given by the set of nodes $\mathcal{U}$ and a weight function $w' : \mathcal{U} \to \mathbb{R}^+$ can be computed in time $\mathcal{O}\left(n(\epsilon^{-1}m + \varepsilon^{-10}k)polylog(\frac{nm\kappa}{\delta}, W, p_{\min}^{-1})\right)$ with error probability δ.*
$\mathcal{U}$ contains $\mathcal{O}\left(\epsilon^{-3}k^2 \cdot polylog(n, \frac{\kappa}{\delta}, W)\right)$ nodes and the size of the support of each node contains at most $\mathcal{O}(\varepsilon^{-3}polylog(m, \frac{\kappa}{\delta}, W))$points.

Proof. First assume that the input is unweighted. By Lemma 7.2.3 and Theorem 7.2.2, we can reduce the support of each node to $\mathcal{O}(\varepsilon^{-3}(\log \kappa \frac{nm}{\delta})(\log m)(\log p_{\min}^{-1}))$ points in $\mathcal{O}(\varepsilon^{-1}nm \cdot (\log \frac{nm}{\delta}) \log(p_{\min}^{-1}))$ time. We run the reduction step with a failure probability of $1 - \delta/2$ and precision parameter $\varepsilon/3$.

We then construct the instance to the deterministic metric k-median problem in the same way as in the proof of Theorem 7.3.2. The computation of a $(1+\varepsilon)$-coreset for the probabilistic metric k-median problem is then equivalent to the computation of a $(1+\varepsilon)$-coreset for the deterministic metric k-median problem. For the deterministic construction, we use Theorem 7.2.2.

During the application of Chen's algorithm, we use a minimum cost flow algorithm. Because of the reduced support sizes, the running time for one computation of the EMD is bounded by $\tilde{\mathcal{O}}(\varepsilon^{-9}((\log \kappa\frac{nm}{\delta})(\log m)(\log p_{\min}^{-1}))^3)$. Using Theorem 7.2.2, we can compute a coreset with $\mathcal{O}(k\varepsilon^{-2}(k\log\kappa + \log\delta^{-1})\log n)$ nodes with failure probability δ in time $\mathcal{O}(nk\log\delta^{-1}\varepsilon^{-9}(\log \kappa nm/\delta \log m \log p_{\min}^{-1})^3)$.

To solve the weighted case, we use a grouping technique as in Lemma 7.2.3 and round all weights down to the closest value $(1+\varepsilon)^\ell$. This changes the cost function only by an ϵ-fraction (notice that all weights are at least one). Let $w_{\max}$ be the maximum weight. We get at most $\lceil \log_{1+\varepsilon}(w_{\max})\rceil + 1 = \mathcal{O}(\varepsilon^{-1}\log(W))$ groups of points with the same weight. For each group, we compute a coreset individually with the algorithm described above while setting the failure probability to $\delta/2n$ (because the number of groups is also bounded by n, so that the overall failure probability is $\delta/2$) and the precision parameter to $\varepsilon/3$. This gives us an algorithm that needs time $\mathcal{O}(nk\varepsilon^{-10}(\log n/\delta(\log \kappa nm/\delta \cdot \log m \log p_{\min}^{-1})^3 \log(W))$ when given reduced nodes and computes a $(k,\varepsilon/3)$-coreset with $\mathcal{O}(k\varepsilon^{-3}(k\log\kappa + \log\delta^{-1})\log n \cdot \log W)$ nodes and succeeds with probability $1-\delta/2$. The support of the probability distributions of all nodes has size at most $\mathcal{O}(\varepsilon^{-3}(\log \kappa\frac{nm}{\delta})(\log m)(\log p_{\min}^{-1}))$.

The overall algorithm succeeds with probability $1-\delta$ and the precision is ε. The overall running time is $\mathcal{O}(\varepsilon^{-1}nm\log\frac{nm}{\delta}\log p_{\min}^{-1} + \varepsilon^{-10}nk\log\frac{n}{\delta}\log W(\log\frac{\kappa nm}{\delta}\log m\log p_{\min}^{-1})^3)$. □

7.4 The assigned Euclidean k-median problem

Even though the Euclidean k-median problem and the metric k-median problem are defined very similarly, none of them is a special case of the other. That is because on the one hand, assuming a metric is less restrictive than assuming the metric induced by the Euclidean norm, but on the other hand, allowing centers from $\mathbb{R}^d$ is less restrictive than allowing centers from a finite set of points. The same is true for the probabilistic counterparts of the problems.

We can still use the reduction from the last section for the assigned Euclidean k-median problem because the Euclidean norm induces a metric. However, we need to equip the deterministic algorithm with a set of possible centers, and additionally, we cannot use Euclidean deterministic algorithms since the metric in the input for the deterministic algorithm is the earth mover distance. Thus we get an additional constant factor even though there exist $(1+\varepsilon)$-approximations for the *Euclidean* deterministic version.

Recall that if the centers can only be chosen from a finite set instead of $\mathbb{R}^d$, we call a problem *discrete*. By the triangle inequality, choosing centers from X instead of $\mathbb{R}^d$ is a 2-approximation. This is true for the deterministic version and holds for our assigned probabilistic problem as well, as we see in the following lemma.

Lemma 7.4.1. *An optimal solution $C \subset X$ for the discrete assigned Euclidean k-median problem is a 2-approximate solution for the assigned Euclidean k-median problem.*

Proof. Let $\mathcal{V}$ be a set of mappings $v_i : X \to [0,1]$, let $w : \mathcal{V} \to \mathbb{R}^+$ be a weight function and let k be a natural number. Let $C^* \subset \mathbb{R}^d$ and $\rho : \mathcal{V} \to C^*$ form an optimal solution for

the assigned Euclidean k-median problem on $\mathcal{V}$, i. e.,

$$\operatorname{ecost}_D(\mathcal{V}, C^*, \rho) = \sum_{i=1}^{n} w(v_i) \sum_{j=1}^{m} p_{ij} \cdot ||x_j - \rho(v_i)||$$

is minimal. We define C' by selecting the closest center from X for every center in C^*, so $C' := \arg\min_{x \in X}\{||x - c|| \mid c \in C^*\}$. We define ρ' accordingly by $\rho'(v_i) = \arg\min_{c' \in C'} ||c' - \rho(v_i)||$. Now let v_i be any node. By the definition of ρ', it holds that $||x_j - \rho(v_i)|| \geq ||\rho'(v_i) - \rho(v_i)||$ for all $j \in [m]$. By the triangle inequality in $\mathbb{R}^d$, this implies that

$$\begin{aligned}
\operatorname{ecost}_D(\mathcal{V}, C', \rho') = & \sum_{i=1}^{n} w(v_i) \sum_{j=1}^{m} p_{ij} \cdot ||x_j - \rho'(v_i)|| \\
\leq & \sum_{i=1}^{n} w(v_i) \sum_{j=1}^{m} p_{ij} \cdot ||x_j - \rho(v_i)|| + ||\rho(v_i) - \rho'(v_i)|| \\
\leq & \; 2 \cdot \sum_{i=1}^{n} w(v_i) \sum_{j=1}^{m} p_{ij} \cdot ||x_j - \rho(v_i)|| \leq 2 \cdot \operatorname{ecost}_D(\mathcal{V}, C^*, \rho).
\end{aligned}$$

An optimal choice of centers from X can only be better than C', and that completes the proof. □

Together with Corollary 7.3.4, this implies a constant approximation for the assigned Euclidean k-median problem. In the deterministic Euclidean case, $(1+\varepsilon)$-approximations are typical, but they assume that either k or d is constant. The following corollary states a constant approximation, but for arbitrary k and d.

Corollary 7.4.2. *For instances where $p_i = 1$ for all nodes v_i, there exists a $(2+2\sqrt{3}+\varepsilon)$-approximation for the unweighted or weighted assigned Euclidean k-median problem. The running time is polynomial in $|X|$ and k, and also in $|\mathcal{V}|$ or $W/(p_{\min} \cdot \varepsilon)$, respectively (where W is the total weight and we assume $w(v_i) \geq 1$ for all $i \in [n]$).*

Proof. We apply Corollary 7.3.4 to the assigned metric k-median problem (with precision parameter $\varepsilon/2$). The input consists of the nodes $\mathcal{V}$, the number of centers k, the finite set of possible locations X and the center candidate set X. The metric space is $\mathbb{R}^d$ with the metric induced by the Euclidean norm. We get a $(1 + \sqrt{3} + \varepsilon/2)$ approximation. By Lemma 7.4.1, the solution is a $(2 + 2\sqrt{3} + \varepsilon)$-approximation for the Euclidean k-median problem on $\mathcal{V}$. □

7.4.1 Superpolynomial algorithms for the assigned Euclidean case

In Section 7.4.2, we develop a coreset construction for the assigned Euclidean k-median problem. It can be used to speed up approximation algorithms. However, no polynomial $(1+\varepsilon)$-approximation is available in the probabilistic case, so we cannot run any on the coreset. We shortly describe an exponential algorithm and the speed-up (which leads to an algorithm that is not exponential, but superpolynomial) after quoting the main result of Section 7.4.2.

Theorem 7.4.21. *Let $X \subset \mathbb{R}^d$ be a finite set and let $\mathcal{V}$ be a set of n nodes $v_i : X \to [0,1]$, $i \in [n]$, let k be a positive integer and let $w : \mathcal{V} \to \mathbb{R}^+$ be a weight function, and let $W := \sum_{i=1}^{n} w(v_i)$ be the total weight. Let $0 < \delta, \varepsilon < 1$ be given constants.*

A $(1+\varepsilon)$-coreset $\mathcal{U}$ for the assigned Euclidean k-median problem consisting of nodes $u_i : X \to [0,1]$ can be computed in time $\mathcal{O}(nd \cdot (k+m) \cdot (d \log(W/(\delta\varepsilon)))$ with error probability δ. The coreset $\mathcal{U}$ consists of $\mathcal{O}(\epsilon^{-2}k^2 d \cdot \log^4(W/(\varepsilon\delta)))$ nodes. In additional running time $\mathcal{O}(nmd\varepsilon^{-1}(\log(nm/\delta) \cdot (\log(p_{\min}^{-1}))))$, the supports of all nodes can be reduced such that each support contains at most $\mathcal{O}(\varepsilon^{-3} d \log^2(m/(p_{\min}\delta\varepsilon)))$ points from X.

Notice that in any solution consisting of a center set $C \subset \mathbb{R}^d$ and an assignment $\rho : \mathcal{V} \to C$ for the assigned Euclidean k-median problem, the input is partitioned into subsets V_c which are assigned to the same cluster c by ρ. In order for the solution to be optimal, it has to hold for each V_c that c is the optimal solution for the assigned Euclidean 1-median problem on V_c. If that was not true, the solution could be improved by replacing the center by this optimal 1-median solution.

Furthermore, for $k = 1$, assigned clustering is directly reducible to unassigned clustering, and thus to a weighted deterministic problem. As all points in V_c are assigned to the same center, the assignment is irrelevant when looking for the best center, and the problem can be interpreted as a deterministic weighted Euclidean 1-median problem. For this problem, we can use an algorithm due to Kumar, Sabharwal and Sen.

Lemma 7.4.3 ([KSS10])**.** *Let P be a set of points in $\mathbb{R}^d$, and let ε, $0 < \varepsilon < 1$, be a precision parameter. There exists an algorithm with running time $\mathcal{O}(2^{(1/\varepsilon)^{\mathcal{O}(1)}} d)$ that finds a point that is a $(1+\varepsilon)$-approximation to the deterministic Euclidean 1-median of P with constant probability and which only needs to access the point set via uniform random sampling.*

That leads to a simple algorithm to approximate the assigned Euclidean k-median problem. It has exponential running time.

Lemma 7.4.4. *A $(1+\varepsilon)$-approximation for the assigned Euclidean k-median problem can be computed in time $\mathcal{O}(k^{n+1} n(m + d \cdot 2^{(1/\varepsilon)^{\mathcal{O}(1)}} \cdot \ln k/\delta))$ with probability $1 - \delta$.*

Proof. We iterate through all ways to partition $\mathcal{V}$ into k subsets. The number of possible partitionings is bounded by k^n. For each partitioning and each of the k subsets $\mathcal{V}'$ in the partitioning we have to compute an approximate deterministic 1-median of the instance where the locations in $\mathcal{X}$ are the input points and the weights are defined by $w'(x_j) = \sum_{v_i \in \mathcal{V}'} w(v_i) \cdot p_{ij}$ for all $j \in [m]$. The total weight of this instance is bounded by W and we can transform it into an unweighted instance with more nodes by Lemma 7.1.2 (interpret it as a probabilistic 1-median problem with m nodes that have all probability concentrated at x_j such that it matches the formulation of the Observation). The total number of weighted 1-median instances is $k \cdot k^n = k^{n+1}$.

The running time of the algorithm by Kumar, Sabharwal and Sen does not depend on the number of input points for the unweighted 1-median problem. Thus, the actual number of points does not matter as long as we avoid to actually compute the unweighted instance

which is possible since the algorithm does not need direct access to the points, only via uniform random samples. We provide these samples via appropriate weighted sampling. The time to set up the weights and probabilities is bounded by $\mathcal{O}(nm)$ for each 1-median instance.

The algorithm is randomized and has a constant failure probability, and we want to compute a significant amount of 1-medians. Thus, we have to decrease the failure probability by performing several runs for each 1-median. Let $1 - \delta'$ be the failure probability of the algorithm by Kumar et al. We run the algorithm $\delta'^{-1} \ln(k^{n+1}/\delta)$ times. Then the probability that it fails in all runs is

$$\left(1 - \frac{1}{1/\delta'}\right)^{(1/\delta')\ln(k^{n+1}/\delta)} \leq \left(\frac{1}{e}\right)^{\ln(k^{n+1}/\delta)} = \frac{\delta}{k^{n+1}}.$$

By the union bound, this implies that the probability that we compute $(1+\varepsilon)$-approximate 1-median for every subset in every partitioning is at least $1-\delta$. As one of the partitionings is optimal, we can find a $(1+\varepsilon)$-approximation by returning the best solution that was found, and this succeeds with probability $1-\delta$, too.

The probability δ' is a constant, so the number of runs is $\mathcal{O}(n \ln k/\delta)$, and each run takes time $\mathcal{O}(2^{(1/\varepsilon)^{\mathcal{O}(1)}} d)$. This implies the bound on the running time. □

The coreset thus yields a superpolynomial approximation algorithm for the assigned Euclidean k-median problem.

Corollary 7.4.5. *A $(1+\varepsilon)$-approximation for the assigned Euclidean k-median problem can be computed in time $\tilde{\mathcal{O}}(nd^2m \log W + sk^{s+1}md)$ for $s \in \mathcal{O}(d \log^4 W/p_{\min})$ with constant probability if ε and k are also constant.*

Proof. We can compute a coreset in time $\mathcal{O}(nd \cdot (k+m) \cdot (d \log(W/(\delta\varepsilon)))$ with error probability $\delta/2$. The coreset is of size $s \in \mathcal{O}(\epsilon^{-2}k^2d \cdot \log^4(W/(p_{\min} \cdot \varepsilon\delta))) \subseteq \mathcal{O}(d \log^4 W/p_{\min})$ for constant k, ε and δ. Applying the algorithm in Lemma 7.4.4 with failure probabilty $\delta/2$ to the coreset takes time $\mathcal{O}(sk^{s+1}md \cdot 2^{(1/\varepsilon)^{\mathcal{O}(1)}} \cdot \ln k/\delta) \subset \tilde{\mathcal{O}}(sk^{s+1}md)$ for constant k, ε and δ. The total running time is $\mathcal{O}(nd^2m \log W) + \tilde{\mathcal{O}}(sk^{s+1}md)$ under these assumptions. □

7.4.2 A coreset construction for the assigned Euclidean case

This section proposes a coreset construction which is an extension of the construction by Chen [Che09]. We mentioned the work by Chen before, e. g., in the related work section on coresets for the k-means problem in Section 4.2.3, specifically on page 74. Chen constructs coresets of a size that is polynomial in d, k and ε and polylogarithmic in the number of points. He adapts his construction for the Euclidean and metric k-median and k-means problem. Chen also shows how to maintain the coresets in data streams.

Since Chen's work, constructions for smaller coresets have been proposed, including the coreset constructions by Langberg and Schulman [LS10] and Feldman and Langberg [FL11a]. The coresets computed by the latter two constructions have a size that

Figure 7.3: An illustration of the partitioning due to Chen. The radii of the rings are $R, 2R, \ldots, 2^{\lceil \log \beta |P| \rceil} R$ where $R := \text{cost}_{\ell_2}(P, A)/(\beta|P|)$ is a lower bound on the optimum clustering cost of P, and $2^{\lceil \log \beta |P| \rceil} R \geq \beta |P| R \geq \text{cost}^*_{\ell_2}(P)$ is an upper bound on the optimum clustering cost.

is independent of the number of input points. It is an open question whether one of these constructions can be carried over to the assigned Euclidean k-median problem. We extend the construction by Chen to work for the assigned Euclidean k-median problem.

In Section 4.2.2, we discussed how uniform sampling can be used to construct coresets for the (discrete unweighted deterministic) k-median clustering problem. We saw that the number of samples that are needed to ensure a small variance can by bounded by

$$\frac{(\text{diam}(P))^2}{2(\varepsilon' \cdot \text{cost}^*_{\ell_2}(P)/|P|)^2} \cdot (k \ln n + \ln(2/\delta)).$$

Thus, the important idea is to partition the input point set into subsets where the diameter is related to the average optimum cost. Chen's construction [Che09] thus works in two steps. First, it partitions the nodes into disjoint subsets. Second, it draws a random sample of points from each subset.

The partitioning is based on a bicriteria approximation that computes a set A of αk centers with a cost of $\beta \cdot \text{cost}^*_{\ell_2}(P)$. The points are first partitioned according to the center that they are closest to. Then, each of the k subsets is subdivided further. The first subset consists of all points that are within distance $\text{cost}_{\ell_2}(P, A)/(\beta|P|) \leq \text{cost}^*_{\ell_2}(P)/|P|$ of the center. Chen then places exponentially growing spheres around the center, subdividing the remaining points by rings of increasing width like Figure 7.3 illustrates. The clustering costs of all points in a ring satisfy a lower bound that is connected to the diameter of the ring, which is the key to the analysis. We see this in more detail below when we extend the method to our application. We need to refine the partitioning because our nodes are probabilistic, which means that nodes can have a significantly different clustering cost even if they are in the same ring.

In addition to partitioning and sampling, we add a third step, where we thin out the support of the the probability distributions of the sampled nodes in order to obtain a

coreset with a small overall representation size. We now discuss the realization of the three steps by comparing them with the steps in [Che09].

Recall that the input to the assigned Euclidean k-median problem contains a set of n nodes $\mathcal{V} = \{v_i : X \to [0,1]\}$, where node v_i for $i \in [n]$ appears at location x_j with probability $p_{ij} = x_j$ and satisfies $p_i = \sum_{j=1}^{m} p_{ij} = 1$. Additionally, the input contains a positive integer k and possibly a weight function $w : \mathcal{V} \to \mathbb{R}^{\geq 1}$. The dimension d is a positive integer, too, and the set of locations $X \subset \mathbb{R}^d$ is finite. In the following description and in the remainder of this section, we will often denote the assigned Euclidean k-median problem as the *probabilistic k-median problem* in order to differentiate it better from the deterministic k-median problem.

Step (1) partitions the nodes into groups that are similar in terms of their locations and their contributions to the total clustering cost. As mentioned, Chen starts the partitioning by computing a bicriteria approximation and assigning each point to its closest center. In the probabilistic case, no bicriteria approximation was known, so we first show how to find such a bicriteria approximation for the assigned probabilistic problem. We start with computing point representatives for each node.

- For every node $v_i \in \mathcal{V}$, compute a point $y_i \in \mathbb{R}^d$ that satisfies $\sum_{j=1}^{m} p_{ij} \cdot ||x_j - y_i|| \leq 2 \cdot \min_{x_{j'} \in X} \sum_{j=1}^{m} p_{ij} \cdot ||x_j - x_{j'}||$, i. e., y_i is a 2-approximation of the probabilistic 1-median for v_i. Let $Y := \{y_1, y_2, \ldots, y_n\} \subset \mathbb{R}^d$ be the set the resulting n points and define the weight of y_i by $w(y_i) := w(v_i)$.

Now, we compute a deterministic bicriteria approximation for the set of 1-medians interpreted as a deterministic clustering problem (whereas Chen computes a bicriteria approximation for the input point set in the deterministic case). A similar step is part of the constant factor approximation for the assigned k-median problem due to Cormode and McGregor [CM08].

- Compute a set $A \subseteq Y$ which is a center set of an $[\alpha, \beta]$-bicriteria approximation to $\text{cost}^*_{\ell_2}(Y)$. That is, $A := \{a_1, \ldots, a_\tau\}$ satisfies

$$\text{cost}_{\ell_2,w}(Y, A) \leq \beta \cdot \text{cost}^*_{\ell_2,w}(Y),$$

  where $\beta \geq 1$ is a constant, $\tau \leq \alpha k$ is the number of centers with $\alpha = \mathcal{O}(\log(W/\varepsilon)))$, $W := \sum_{v_i \in V} w(v_i)$ is the total weight of the nodes (and of Y). Let $\sigma_Y : Y \to A$ be an assignment that assigns every point in Y to a closest center in A. Define $\sigma_{\mathcal{V}} : \mathcal{V} \to A$ to be the corresponding assignment for $\mathcal{V}$ such that $\sigma_{\mathcal{V}}(v_i) = \sigma_Y(y_i)$ for all $i \in [n]$.

We show that the computed set A is actually a $[3\beta + 2, \alpha]$-bicriteria approximation to the probabilistic optimum $\text{ecost}^{*,k}_D(\mathcal{V})$. This enables us to transfer the partitioning step by Chen. We show that $R := \text{ecost}_{D,w}(\mathcal{V}, A, \sigma_{\mathcal{V}})/((3\beta + 2)W)$ is a lower bound on the average radius of the optimal cost of a probabilistic k-median clustering for $\mathcal{V}$ and an upper bound $2^\nu R$ on the distance between an arbitrary point in Y and its closest center in A. The value of ν is $\lceil \log((9\beta + 6)W) \rceil$. This ensures that the following partitioning is indeed a partitioning into disjoint subsets:

- For all $\ell \in [\tau]$, define $Y_\ell \subseteq Y$ as the subset of points in Y that σ_Y assigns to a_ℓ, i. e., Y_ℓ is the set of all points whose closest point in A is a_ℓ (ties broken arbitrarily). Set $\mathcal{V}_\ell := \{v_i | y_i \in Y_\ell\}$. Furthermore, for each $\ell \in [\tau]$ and each $h \in \{0, 1, \ldots, \nu\}$, let

$$Y_{\ell,h} := \begin{cases} Y_\ell \cap B(a_\ell, R) & h = 0 \\ Y_\ell \cap [B(a_\ell, 2^h R) \backslash B(a_\ell, 2^{h-1} R)] & h \geq 1 \end{cases}$$

  be the *h-th ring set* for the center a_ℓ and the corresponding set Y_ℓ. Recall that $B(a_\ell, R) = \{y \in \mathbb{R}^d \mid ||a_\ell - y|| \leq R\}$ is the set of all points in $\mathbb{R}^d$ that are within distance R of a_ℓ. Set $\mathcal{V}_{\ell,h} := \{v_i | y_i \in Y_{\ell,h}\}$.

This gives us subsets of nodes which have relatively close 1-medians. For our purposes this partitioning is not sufficient, as the probability distributions of the nodes can have different variances and may not behave similarly according to the cost function. Thus, we further subdivide the ring sets according to $\sum_{x_j \in X} p_{ij} D(x_j, y_i)$. Notice that this term is not the variance because the distances are not squared. It is the expected distance of the realization of v_i to y_i. We will see that $2^\mu R$ is an upper bound on this expected distance for all v_i if μ is chosen as $\lceil \log((6\beta + 4)W) \rceil$. For each $\ell \in [\tau]$, each $h \in \{0, 1, \ldots, \nu\}$, and each $a \in \{0, 1, \ldots, \mu\}$, let

$$Y_{\ell,h,a} := \begin{cases} \{y_i \in Y_{\ell,h} \mid w(y_i) \cdot \sum_{x_j \in X} p_{ij} \cdot ||x_j - y_i|| \leq R\} & a = 0 \\ \{y_i \in Y_{\ell,h} \mid 2^{a-1} R < w(y_i) \cdot \sum_{x_j \in X} p_{ij} \cdot ||x_j - y_i|| \leq 2^a R\} & a \geq 1 \end{cases}$$

and set $\mathcal{V}_{\ell,h,a} := \{v_i \mid y_i \in Y_{\ell,h,a}\}$. The set of all $\mathcal{V}_{\ell,h,a}$ is the desired partitioning.

Step (2) consists of sampling from each subset. We describe the sampling for weighted inputs, since even if the input is originally unweighted, we need weights for our assumption that all realization probabilities are one. From each $\mathcal{V}_{\ell,h,a}$, we sample a multiset $\mathcal{U}_{\ell,h,a}$ with $s' := \lceil c \cdot \varepsilon^{-2} \cdot [\log(1/\delta) + k(\log(k) + d\log(1/\varepsilon) + \log(\log(W/\varepsilon)))] \rceil$ nodes, where c is a sufficiently large constant. The nodes are sampled with replacement, and, in each sample step, a node $v_i \in \mathcal{V}_{\ell,h,a}$ is picked with probability $w(v_i) / \sum_{v_{i'} \in \mathcal{V}_{\ell,h,a}} w(v_{i'})$. We set the weight of the sample node v_i to $w'(v_i) = \sum_{v_{i'} \in \mathbf{V}_{\ell,h,a}} w(v_{i'})/s'$. We store all sampled nodes in the multiset

$$\mathcal{U} := \{u_1, \ldots, u_s\} := \bigcup_{\ell=1}^{\tau} \bigcup_{h=0}^{\nu} \bigcup_{a=0}^{\mu} \mathcal{U}_{\ell,h,a}.$$

The size of $\mathcal{U}$ is $s = s' \cdot \tau \cdot (\nu + 1) \cdot (\mu + 1) \in \mathcal{O}(k^2 d \varepsilon^{-2} \log^4(W/(\varepsilon\delta)))$.

Step (3) computes a node $z_i : X \to [0, 1]$ with a sparse support for each $v_i \in \mathcal{V}$ by using the reduction from Lemma 7.2.3 combinded with Theorem 7.2.2. The node z_i assigns a positive probability to at most $\mathcal{O}(\varepsilon^{-3} d \log^2(m/(p_{\min}\delta\varepsilon)))$ points from X.

Realization and running time

We use known results to implement the steps. For Step (1), we need to compute the sets Y and A. Computing Y works similar to the proof of Lemma 7.4.4 where we noticed that for $k = 1$, the assigned Euclidean k-median problem directly reduces to a deterministic problem. This applies to computing y_i as well, since y_i is an approximate probabilistic 1-median of v_i. Recall the following result by Kumar, Sabharwal and Sen stated as Lemma 7.4.3 on page 173.

Lemma 7.4.3 ([KSS10]). *Let P be a set of points in $\mathbb{R}^d$, and let ε, $0 < \varepsilon < 1$, be a precision parameter. There exists an algorithm with running time $\mathcal{O}(2^{(1/\varepsilon)^{\mathcal{O}(1)}} d)$ that finds a point that is a $(1+\varepsilon)$-approximation to the deterministic Euclidean 1-median of P with constant probability and which only needs to access the point set via uniform random sampling.*

As in Lemma 7.4.4, we notice that we can theoretically convert v_i into a set of unweighted nodes by Lemma 7.1.2, but do not actually do the conversion. Instead, we use our knowledge of this transformation to simulate uniform sampling on the implicitly given unweighted point set. The (implicit) deterministic k-median instance consists of the union of the supports of the unweighted nodes. We compute a 2-approximate 1-median for this instance. That is done by running multiple runs of the algorthm by Kumar, Sabharwal and Sen with a constant $\varepsilon < 1$ by using simulated uniform sampling, and then returning the best found 1-median. If the algorithm succeeds in at least one run, then the computed 1-median is at least a 2-approximation since $\varepsilon < 1$. Let the failure probability of one run of the algorithm be $1 - \delta'$. We set the number of runs to $(1/\delta') \log(n/\delta)$, so the probability that it fails in all runs is bounded by $(1 - \delta')^{(1/\delta') \log(n/\delta)} \leq 1 - \delta/n$. Thus, we find a 2-approximate 1-median with a probability of at least $1 - \delta/n$, and the probability that we compute one for all nodes without a failure is $1 - \delta$ by the union bound. Since we run the algorithm with constant precision, each run takes $\mathcal{O}(d)$ time. We can thus compute Y in time $\mathcal{O}(nd(1/\delta') \log(n/\delta))$. Setting up the probabilities before the simulated uniform sampling takes $\mathcal{O}(nmd)$ time for all nodes together.

Computing A means that we have to compute a bicriteria approximation to the deterministic weighted k-median problem for Y. For this, we use an algorithm by Indyk [Ind99] in a version presented by Chen [Che09]. It computes a bicriteria approximation for the metric k-median problem. Since the best solution consisting of input points for the Euclidean k-median problem is a 2-approximation for the Eucliden k-median problem, the bicriteria approximation also works for the Euclidean k-median problem. Computing a distance takes time $\mathcal{O}(d)$, as a worst case bound we can assume that the time multiplies by $\mathcal{O}(d)$.

The following theorem needs $k \in \mathcal{O}(\sqrt{n})$. Notice that our coreset size (as well as the coreset size of Chen's construction) is $\Omega(k^2)$, so $k \in \mathcal{O}(\sqrt{n})$ does not pose an additional restriction. If k is larger, then using n input points is better than actually computing the coreset.

Theorem 7.4.6 ([Ind99] and appendix A in [Che09]). *Given a set P of n points from a metric space, let $k \in \mathcal{O}(\sqrt{n})$ be a given parameter. It is possible to compute $\mathcal{O}(k)$ centers*

in $\mathcal{O}(nk \log 1/\delta)$ time such that clustering P with them has a k-median cost that is within a constant factor of the cost of an optimal k-median solution for P with probability $1-\delta$. For weighted input (with integer weights), the number of centers is $\mathcal{O}(k \log W)$ and the running time is $\mathcal{O}(nk \log(1/\delta) \log\log W)$.

We apply Theorem 7.4.6 to Y. As we need integer weights, we weight y_i by $\lfloor w(v_i)/\varepsilon \rfloor$ for $i \in [n]$. Notice that $\lfloor w(v_i)/\varepsilon \rfloor \cdot \varepsilon \geq (w(v_i)/\varepsilon - 1)\varepsilon = w(v_i) - \varepsilon \geq w(v_i) - \varepsilon w(v_i) = (1-\varepsilon)w(v_i)$ since the weights are at least one. So our instance corresponds to an instance which has a cost that is at most an ε-fraction off the true cost. By adjusting this ε and the precision of the coreset appropriately, this does not affect our coreset result. We get $\mathcal{O}(k \log W/\varepsilon)$ centers in $\mathcal{O}(nkd \log(1/\delta) \log(\log W/\varepsilon))$ time.

For each y_i, we can compute $\sigma_Y(y_i)$ and thus the index ℓ with $v_i \in \mathcal{V}_\ell$ by computing the distances to the $\mathcal{O}(k \log W/\varepsilon)$ centers in time $\mathcal{O}(dk \log W/\varepsilon)$. As this also gives us the distance to a_ℓ, we can obtain the index h of the ring set in constant time by computing $\lceil \log ||y_i - a_\ell|| \rceil - \log R$. Then we compute the expected distance of v_i to y_i in time $\mathcal{O}(md)$ in order to compute the index a such that $v_i \in \mathcal{V}_{\ell,h,a}$, again in constant time. Overall, determining ℓ, h and a takes $\mathcal{O}(dk \log W/\varepsilon + md)$ for each node.

Determining the sample probabilities for all nodes can be done in time $\mathcal{O}(n)$, and then the sampling takes time $s = s' \cdot \tau \cdot (\nu+1) \cdot (\mu+1) \leq n$ under the assumption that it is possible to draw a random sample in constant time.

The sum of these running times lies in $\mathcal{O}(nd \cdot (k+m) \cdot (d \log(W/(\delta\varepsilon)))$. Finally, computing sparse representations of all sample nodes takes additional time $\mathcal{O}(nmd\varepsilon^{-1}(\log(nm/\delta) \cdot \log(p_{\min}^{-1})))$ when using Lemma 7.2.3 with Theorem 7.2.2 as described in Section 7.2.

Overview on the analysis

In order to analyze the sampling, we use Theorem 4.2.3 on page 68 which is due to Haussler [Hau90, Hau92]. After transforming the input to an unweighted version, we can show that sampling $s'' = \lceil \xi^{-2} \ln 1/\delta' \rceil$ nodes from each $V_{\ell,a,h}$ for a suitable δ' and a constant $\xi > 0$ is sufficient to bound the error for a given center set C by

$$\xi \sum_{\text{all } V_{\ell,a,h}} \sum_{v_{i'} \in V_{\ell,a,h}} w(v_{i'}) \Bigg(d(Y_{\ell,a,h}, C) + \operatorname{diam}(Y_{\ell,a,h}) + \max_{y_i \in Y_{\ell,a,h}} \sum_{x_j \in X} p_{ij} \cdot ||x_j - y_i|| \Bigg)$$

with high probability, where we abbreviate $\operatorname{dist}(Y_{\ell,a,h}, C) = \{||y_i - c|| \mid y_i \in Y_{\ell,a,h}, c \in C\}$ by $d(Y_{\ell,a,h}, C)$. The abbreviation $d(P,Q) := \operatorname{dist}(P,Q) = \min\{||x-y|| \mid x \in P, y \in Q\}$ will be frequently used, as well as $d(x,P) := \operatorname{dist}(x,P) = \min\{||x-y|| \mid y \in P\}$.

We show that the first of the three summands is bounded because we use 1-medians to calculate A. Then, we prove that the second term is bounded because we partitioned into groups of nodes with closely located approximated 1-medians. The third term is bounded because the nodes also have similar expected distances to their y_i.

The final step is to show that the coreset property holds for *all* center sets. In order to use the union bound, we need to discretize the input space in order to show that it is sufficient to prove the coreset property for a small enough family of center sets.

Similar to Chen, we define huge spheres restricting the position of possible centers and subdivide these by a suitable grid. Finding suitable huge spheres turns out to be challenging, as the probability distributions can be widespread.

Simulating Weighted Sampling

We use Lemma 7.1.2 (see page 162) to transform the input point set into an unweighted point set. We get an unweighted set $\bar{\mathcal{V}}$ containing $\lfloor w(v_i)/(p_{\min}\varepsilon)\rfloor$ nodes $\bar{v}_{iz}$ for each $v_i \in \mathcal{V}$ with $\bar{p}_{ij} := \bar{v}_{iz}(x_j) := p_{ij} \cdot p_{\min} \cdot \varepsilon$, so each copy has a total probability of $p_{\min} \cdot \varepsilon$. The following holds. For each subset $\mathcal{T} \subseteq \mathcal{V}$ and the corresponding subset $\bar{\mathcal{T}} := \{\bar{v}_{iz} \mid v_i \in \mathcal{T}, z = 1, \ldots, \lfloor w(v_i)/(p_{\min}\varepsilon)\rfloor\}$ of $\bar{\mathcal{V}}$, and for each $C \subset \mathcal{L}$ with $|C| = k$, it holds that

$$\left|\mathrm{ecost}_{D,w}(\mathcal{T}, C, \rho) - \mathrm{ecost}_D(\bar{\mathcal{T}}, C, \bar{\rho})\right| \leq \varepsilon \cdot \mathrm{ecost}_{D,w}(\mathcal{T}, C, \rho)$$

for any mapping $\rho : \mathcal{V} \to C$ and $\bar{\rho}(\bar{v}_{iz}) := \rho(v_i)$ for all $i \in [n]$ and $z = 1, \ldots, \lfloor w(v_i)/(p_{\min}\varepsilon)\rfloor$.

Notice that $\bar{\mathcal{V}}$ corresponds directly to $\mathcal{V}$ where the weight of v_i is rounded down to $\lfloor w(v_i)/(p_{\min}\varepsilon)\rfloor \cdot p_{\min} \cdot \varepsilon$. Thus, changing the weights in this way causes the clustering cost to change by at most an ε-fraction by Lemma 7.1.2. Without loss of generality we can assume that this step has been performed already, which means that $w(v_i)/(p_{\min}\varepsilon) \in \mathbb{N}$ and $\mathrm{ecost}_{D,w}(\mathcal{T}, C, \rho) = \mathrm{ecost}_D(\bar{\mathcal{T}}, C, \bar{\rho})$ holds for all $\mathcal{T} \subseteq \mathcal{V}$.

Let $\mathcal{S}$ be a set of sample nodes where each $u = v_i \in \mathcal{S}$ is picked independently and non-uniformly at random from $\mathcal{T}$ according to the weights and realization probabilities, i. e., v_i is picked with probability $w(v_i)/\sum_{v_{i'} \in \mathcal{T}} w(v_{i'})$. Construct $\bar{\mathcal{S}}$ by adding one of the copies $v_{i,z} \in \bar{\mathcal{T}}$ of v_i chosen uniformly at random for each $u = v_i \in \mathcal{S}$. This means that $\bar{\mathcal{S}}$ contains the same number of nodes as $\mathcal{S}$, while $\bar{\mathcal{T}}$ contains $\left(\sum_{v'_i \in \mathcal{T}} w(v'_i)\right)/(p_{\min} \cdot \varepsilon)$ nodes.

$\bar{\mathcal{S}}$ is a set of sample nodes from $\bar{\mathcal{T}}$ where each sample node is picked independently and uniformly at random from $\bar{\mathcal{T}}$ since each $v_{i,z}$ is picked with probability

$$\frac{w(v_i)}{\sum_{v_{i'} \in \mathcal{T}} w(v_{i'})} \cdot \frac{p_{\min} \cdot \varepsilon}{w(v_i)} = \frac{p_{\min} \cdot \varepsilon}{\sum_{v_{i'} \in \mathcal{T}} w(v_{i'})} = \frac{1}{|\bar{\mathcal{T}}|}.$$

We set the weight of any sample node $v_i \in \mathcal{S}$ to

$$w'(v_i) := \frac{1}{|\mathcal{S}|} \sum_{v_{i'} \in T} w(v_{i'}) = \frac{|\bar{\mathcal{T}}|}{|\bar{\mathcal{S}}|}(p_{\min} \cdot \varepsilon)$$

which implies the following relationship between the clustering cost of $\mathcal{S}$ and $\bar{\mathcal{S}}$

$$\begin{aligned}
&\operatorname{ecost}_{D,w'}(\mathcal{S}, C, \rho) \\
=&\sum_{v_i \in \mathcal{S}} w'(v_i) \sum_{x_j \in X} p_{ij} \cdot ||x_j - \rho(v_i)|| = \sum_{v_{i,z} \in \bar{\mathcal{S}}} w'(v_i) \sum_{x_j \in X} p_{ij} \cdot ||x_j - \rho(v_i)|| \\
=&\sum_{v_{i,z} \in \bar{\mathcal{S}}} \frac{|\bar{\mathcal{T}}|}{|\bar{\mathcal{S}}|} p_{\min} \cdot \varepsilon \sum_{x_j \in X} p_{ij} \cdot ||x_j - \rho(v_i)|| = \frac{|\bar{\mathcal{T}}|}{|\bar{\mathcal{S}}|} \sum_{v_{i,z} \in \bar{\mathcal{S}}} \sum_{x_j \in X} \bar{p}_{ij} \cdot ||x_j - \bar{\rho}(v_{iz})|| \\
=&\frac{|\bar{\mathcal{T}}|}{|\bar{\mathcal{S}}|} \cdot \operatorname{ecost}_D(\bar{\mathcal{S}}, C, \bar{\rho}) \qquad (7.1)
\end{aligned}$$

where $\rho : \mathcal{S} \to C$ is a mapping and $\rho : \bar{\mathcal{S}} \to C$ is defined by $\bar{\rho}(v_{iz}) = \rho(v_i)$ for all $v_{iz} \in \bar{\mathcal{S}}$.

Now we can apply Theorem 4.2.3 to find a way to bound the number of samples $|\mathcal{S}|$ while keeping the error bounded. The following lemma corresponds to Lemma 3.3 in [Che09], but we have to work with the approximated 1-medians instead of the input points, and we get two additional summands.

Lemma 7.4.7. *Let $\mathcal{T}$ be any subset of $\mathcal{V}$, let $Y(\mathcal{T}) := \{y_i \in Y \mid v_i \in \mathcal{T}\}$ be the subset of the approximated probabilistic 1-medians of all the nodes contained in $\mathcal{T}$, and let $\delta', \xi > 0$ be given parameters. Let $\mathcal{S}$ be a sample of $s'' := \lceil \xi^{-2} \ln(2/\delta') \rceil$ nodes picked independently at random from $\mathcal{T}$ where each $v_i \in \mathcal{T}$ is picked with probability $w(v_i)/\sum_{v_{i'} \in \mathcal{T}} w(v_{i'})$. Assign the weight $w'(v_i) := \sum_{v_{i'} \in \mathcal{T}} w(v_{i'})/s''$ to $v_i \in \mathcal{S}$. Then it holds for any fixed set $C \subseteq X$ with $|C| = k$ that*

$$\begin{aligned}
&\left| \min_{\rho:\mathcal{T}\to C} \operatorname{ecost}_{D,w}(\mathcal{T}, C, \rho) - \min_{\rho:\mathcal{S}\to C} \operatorname{ecost}_{D,w'}(\mathcal{S}, C, \rho) \right| \\
\leq\ & \xi \left(\sum_{v_{i'} \in \mathcal{T}} w(v_{i'}) \right) \left(d(Y(\mathcal{T}), C) + \operatorname{diam}(Y(\mathcal{T})) + \max_{y_i \in Y(\mathcal{T})} \sum_{x_j \in X} p_{ij} ||x_j - y_i|| \right)
\end{aligned}$$

with probability at least $1 - \delta'$, where we use $d(Y(\mathcal{T}), C) := \operatorname{dist}(Y(\mathcal{T}), C)$.

Proof. Consider $\bar{\mathcal{T}}$ and $\bar{\mathcal{V}}$ as defined above. Define the function $f_C : \bar{\mathcal{V}} \to \mathbb{R}$ by

$$f_C(v_{i,z}) := \min_{c \in C} \sum_{x_j \in X} \bar{p}_{ij} \cdot ||x_j - c||$$

for the given set C. For $c \in C$, set $y^c := \arg\min_{y' \in Y(\mathcal{T})} ||y' - c||$. Then, by the triangle

inequality, for every node $v_{i,z} \in \bar{\mathcal{T}} \subseteq \bar{\mathcal{V}}$, it holds that

$$\begin{aligned}
0 \;\leq\; & f_C(v_{i,z}) = \min_{c\in C} \sum_{x_j\in X} \bar{p}_{ij}||x_j - c|| \leq \min_{c\in C} \sum_{x_j\in X} \bar{p}_{ij}\left[||x_j - y_i|| + ||y_i - y^c|| + ||y^c - c||\right] \\
\leq\; & \sum_{x_j\in X} \bar{p}_{ij}||x_j - y_i|| + \min_{c\in C} \sum_{x_j\in X} p_{ij}p_{\min}\varepsilon||y_i - y^c|| + \min_{c\in C} \sum_{x_j\in X} p_{ij}p_{\min}\varepsilon||y^c - c|| \\
\leq\; & \sum_{x_j\in X} \bar{p}_{ij}||x_j - y_i|| + \min_{c\in C} ||y_i - y^c||p_{\min}\varepsilon + \min_{c\in C} ||y^c - c||p_{\min}\varepsilon \\
\leq\; & \sum_{x_j\in X} \bar{p}_{ij}||x_j - y_i|| + \operatorname{diam}(Y(\mathcal{T}))\cdot p_{\min}\varepsilon + d(Y(\mathcal{T}), C)\cdot p_{\min}\varepsilon \\
\leq\; & \max_{y_{i'}\in Y(\mathcal{T})} \sum_{x_j\in X} \bar{p}_{i'j}||x_j - y_{i'}|| + d(Y(\mathcal{T}), C)p_{\min}\varepsilon + \operatorname{diam}(Y(\mathcal{T}))p_{\min}\varepsilon
\end{aligned}$$

where for the last inequality, we replace the first term by the maximum over the possible terms for all nodes. Due to Theorem 4.2.3, setting $\mathcal{F} := \{f_C\}$ for the given C, $M := d(Y(\mathcal{T}), C)\cdot p_{\min}\cdot\varepsilon + \operatorname{diam}(Y(\mathcal{T}))\cdot p_{\min}\cdot\varepsilon + \max_{y_i\in Y(\mathcal{T})}\sum_{x_j\in X}\bar{p}_{ij}\cdot||x_j - y_i||$ and $N := \xi M$, we have that, for a sample $\widetilde{S}$ of size

$$s'' = \left\lceil \xi^{-2}\cdot\ln\left(\frac{2}{\delta'}\right)\right\rceil \geq \frac{M^2}{2N^2}\cdot\ln\left(\frac{2}{\delta'}\right)$$

from $\bar{\mathcal{T}}$, it holds that

$$\begin{aligned}
& \mathbf{Pr}\left[\left|\frac{1}{|\bar{\mathcal{T}}|}\sum_{v_{i,z}\in\bar{\mathcal{T}}}\min_{c\in C}\sum_{x_j\in X}\bar{p}_{ij}\cdot||x_j - c|| - \frac{1}{|\widetilde{S}|}\sum_{v_{i,z}\in\widetilde{S}}\min_{c\in C}\sum_{x_j\in X}\bar{p}_{ij}\cdot||x_j - c||\right| \geq \xi M\right] \\
=\; & \mathbf{Pr}\left[\left|\frac{f(\bar{\mathcal{T}})}{|\bar{\mathcal{T}}|} - \frac{f(\widetilde{S})}{|\widetilde{S}|}\right| \geq N\right] \leq \delta'.
\end{aligned}$$

Due to Equality (7.1), this implies that

$$\begin{aligned}
& \left|\min_{\rho:\mathcal{T}\to C}\operatorname{ecost}_{D,w}(\mathcal{T}, C, \rho) - \min_{\rho:\mathcal{S}\to C}\operatorname{ecost}_{D,w'}(\mathcal{S}, C, \rho)\right| \\
=\; & \left|\min_{\rho:\mathcal{T}\to C}\operatorname{ecost}_D(\bar{\mathcal{T}}, C, \rho) - \min_{\rho:\mathcal{S}\to C}\frac{|\bar{\mathcal{T}}|}{|\bar{\mathcal{S}}|}\cdot\operatorname{ecost}_D(\bar{\mathcal{S}}, C, \rho)\right| \\
=\; & |\bar{\mathcal{T}}|\cdot\left|\frac{1}{|\bar{\mathcal{T}}|}\min_{\rho:\mathcal{T}\to C}\operatorname{ecost}_D(\bar{\mathcal{T}}, C, \rho) - \frac{1}{|\bar{\mathcal{S}}|}\min_{\rho:\mathcal{S}\to C}\operatorname{ecost}_D(\bar{\mathcal{S}}, C, \rho)\right| \\
\leq\; & \xi|\bar{\mathcal{T}}|\cdot M \\
=\; & \xi|\bar{\mathcal{T}}|\left(d(Y(\mathcal{T}), C)p_{\min}\varepsilon + \operatorname{diam}(Y(\mathcal{T}))p_{\min}\varepsilon + \max_{y_i\in Y(\mathcal{T})}\sum_{x_j\in X}\bar{p}_{ij}||x_j - y_i||\right) \\
=\; & \xi\left(\sum_{v_{i'}\in\mathcal{T}} w(v_{i'})\right)\left(d(Y(\mathcal{T}), C) + \operatorname{diam}(Y(\mathcal{T})) + \max_{y_i\in Y(\mathcal{T})}\sum_{x_j\in X} p_{ij}||x_j - y_i||\right)
\end{aligned}$$

with probability at least $1 - \delta'$. □

Further analysis plan and first upper bound

Now assume that we are given a partitioning $\mathcal{V} = \bigcup_{r=1}^{\lambda} \mathcal{V}_r$ of disjoint subsets of $\mathcal{V}$ and sample sufficiently many points from each $\mathcal{V}_r$ to apply Lemma 7.4.7. Then, with probability $1 - \delta'$, the total error induced is

$$\xi\Bigg(\sum_{r=1}^{\lambda} \sum_{v_{i'} \in \mathcal{V}_r} w(v_{i'}) d(Y(\mathcal{V}_r), C) + \sum_{r=1}^{\lambda} \sum_{v_{i'} \in \mathcal{V}_r} w(v_{i'}) \operatorname{diam}(Y(\mathcal{V}_r)) + \sum_{r=1}^{\lambda} \sum_{v_{i'} \in \mathcal{V}_r} w(v_{i'}) \max_{y_i \in Y(\mathcal{V}_r)} \sum_{x_j \in X} p_{ij} \cdot ||x_j - y_i|| \Bigg).$$

Thus, we need to show that our partitioning allows for a good upper bound on these three error terms while keeping λ small. More precisely, we have to relate each of the terms to the optimal cost. By setting ξ appropriately, that will then yield to a coreset.

We start with a useful lemma that shows why it is helpful to work with approximate 1-medians as the representatives of the nodes. Interestingly, it provides a bound on the last term of our error term for the case that we partition $\mathcal{V}$ into $|\mathcal{V}|$ partitions each containing one node (which we will not do because it would induce $|\mathcal{V}|$ subsets).

Lemma 7.4.8. *If each $y_i \in Y$ is a 2-approximation of the probabilistic 1-median of v_i, then $\sum_{i=1}^{n} \sum_{j=1}^{m} w(v_i) p_{ij} \cdot ||x_j - y_i|| \leq 2 \cdot \operatorname{ecost}^{*,n}_{D,w}(\mathcal{V}) \leq 2 \cdot \operatorname{ecost}^{*}_{D,w}(\mathcal{V})$.*

Proof. The assigned k-median problem demands that a node is clustered with one center. Thus, we can assume that in an optimal solution with n centers, there is a one-to-one correspondence between centers and nodes. Then, the center corresponding to a node v_i is a 1-median for the deterministic problem on X where x_j is weighted with $w(v_i)p_{ij}$. Since the y_i are a 2-approximation for the cheapest such 1-median, we get that the cost $\sum_{j=1}^{m} w(v_i) p_{ij} \cdot ||x_j - y_i||$ cannot be more than twice the cost of v_i in an optimal assigned n-median solution. In other words, the set Y provides a 2-approximation to $\operatorname{ecost}^{*,n}_{D,w}(\mathcal{V})$. Furthermore, $2 \cdot \operatorname{ecost}^{*,n}_{D,w}(\mathcal{V}) \leq 2 \cdot \operatorname{ecost}^{*}_{D,w}(\mathcal{V})$ because $k \leq n$. □

Now we show that the first of the three error terms is always bounded as long as we use 2-approximate 1-medians to form Y.

Lemma 7.4.9. *Let $C \subseteq X$ be any set of k cluster centers, let $\mathcal{V} = \bigcup_{r=1}^{\lambda} \mathcal{V}_r$ be any partitioning of $\mathcal{V}$ into disjoint subsets and set $Y(\mathcal{V}_r) := \{y_i \in Y \mid v_i \in \mathcal{V}_r\}$ for each set in this partitioning. If each $y_i \in Y$ is a 2-approximation of the probabilistic 1-median of v_i, then we have*

$$\sum_{r=1}^{\lambda} \sum_{v_{i'} \in \mathcal{V}_r} w(v_{i'}) \operatorname{dist}(Y(\mathcal{V}_r), C) \leq \sum_{y_i \in Y} w(y_i) \cdot \operatorname{dist}(y_i, C)$$

and

$$\sum_{y_i \in Y} w(y_i) \operatorname{dist}(y_i, C) \leq 3 \cdot \min_{\rho : \mathcal{V} \to C} \operatorname{ecost}_{D,w}(\mathcal{V}, C, \rho).$$

Proof. Let $C \subseteq X$ be any set of k centers for the probabilistic k-median clustering problem for $\mathcal{V}$, and let $\rho : \mathcal{V} \to C$ be an assignment from nodes in $\mathcal{V}$ to centers in C that achieves the minimum cost. Notice that

$$\begin{aligned}\sum_{r=1}^{\lambda} \sum_{v_{i'} \in \mathcal{V}_r} w(v_{i'}) \operatorname{dist}(Y(\mathcal{V}_r), C) &= \sum_{r=1}^{\lambda} \sum_{v_{i'} \in \mathcal{V}_r} w(v_{i'}) \min_{y_i \in Y(\mathcal{V}_r)} \operatorname{dist}(y_i, C) \\ &\leq \sum_{r=1}^{\lambda} \sum_{v_{i'} \in \mathcal{V}_r} w(v_{i'}) \operatorname{dist}(y_{i'}, C) = \sum_{y_i \in Y} w(y_i) \operatorname{dist}(y_i, C)\end{aligned}$$

and thus the first part of the claim holds.

By Lemma 7.4.8 we have $\sum_{i=1}^{n} \sum_{j=1}^{m} w(v_i) p_{ij} \cdot ||x_j - y_i|| \leq 2 \cdot \operatorname{ecost}^*_{D,w}(\mathcal{V})$. Combining this with the triangle inequality, we gain

$$\begin{aligned}&\sum_{y_i \in Y} w(y_i) d(y_i, C) \leq \sum_{i=1}^{n} w(v_i) ||y_i - \rho(v_i)|| = \sum_{i=1}^{n} \sum_{x_j \in X} w(v_i) p_{ij} ||y_i - \rho(v_i)|| \\ &\leq \sum_{i=1}^{n} \sum_{x_j \in X} w(v_i) p_{ij} \left(||y_i - x_j|| + ||x_j - \rho(v_i)|| \right) \leq 2 \operatorname{ecost}^*_{D,w}(\mathcal{V}) + \operatorname{ecost}_{D,w}(\mathcal{V}, C, \rho) \\ &\leq 3 \cdot \operatorname{ecost}_{D,w}(\mathcal{V}, C, \rho).\end{aligned}$$

□

Bicriteria Approximation

Let $A = \{a_1, \ldots, a_\tau\} \subseteq Y$ be the center set of the computed $[\alpha, \beta]$-bicriteria approximation to $\operatorname{cost}^*_{\ell_2,w}(Y)$. Recall that $\beta \geq 1$ is some constant, $\tau \leq \alpha k$, $\alpha = \mathcal{O}(\log(W/\varepsilon))$. We show that A is not only a $[\alpha, \beta]$-bicriteria approximation to $\operatorname{cost}^*_{\ell_2,w}(Y)$, but also a $[3\beta + 2, \alpha]$-bicriteria approximation to $\operatorname{ecost}^*_{D,w}(\mathcal{V})$. Recall that $\sigma_Y : Y \to A$ assigns every point in Y to its closest center in A, and that $\sigma_{\mathcal{V}} : \mathcal{V} \to A$ is the corresponding assignment for $\mathcal{V}$. By the triangle inequality, we can write $\operatorname{ecost}_{D,w}(\mathcal{V}, A, \sigma_{\mathcal{V}})$ as the sum of two terms: The cost of the probabilistic input when using the y_i as centers (which we already bounded in Lemma 7.4.8) and the (weighted) distance between each y_i and its closest a_ℓ in A. The latter is bounded in the following lemma.

Lemma 7.4.10. *If each $y_i \in Y$ is a 2-approximation of the probabilistic 1-median of v_i and A is an $[\alpha, \beta]$-bicriteria approximation to $\operatorname{cost}^*_{\ell_2,w}(Y)$, then it holds that*

$$\sum_{\ell=1}^{\tau} \sum_{y_i \in Y_\ell} w(y_i) \cdot ||y_i - a_\ell|| \leq 3\beta \cdot \operatorname{ecost}^*_{D,w}(\mathcal{V}).$$

Also, $\sum_{\ell=1}^{\tau} \sum_{y_i \in Y_\ell} w(y_i) \cdot ||y_i - a_\ell|| \leq 3 \operatorname{ecost}_{D,w}(\mathcal{V}, A, \sigma_{\mathcal{V}})$.

Proof. Let $C \subseteq X$ be any set of k optimal centers for the probabilistic k-median clustering problem for $\mathcal{V}$, and let $\rho : \mathcal{V} \to C$ be an optimal assignment from nodes in V to centers

in C. Note that ρ provides an assignment for Y as well (assign y_i to $\rho(v_i)$), but for Y this is not necessarily optimal. By triangle inequality, we have

$$\begin{aligned}\mathrm{cost}^*_{\ell_2,w}(Y) &\leq \sum_{i=1}^{n} w(y_i)||y_i - \rho(v_i)|| = \sum_{i=1}^{n}\sum_{x_j\in\mathcal{X}} w(v_i)p_{ij}||y_i - \rho(v_i)|| \\ &\leq \sum_{i=1}^{n}\sum_{x_j\in\mathcal{X}} w(v_i)p_{ij}\left(||y_i - x_j|| + ||x_j - \rho(v_i)||\right) \leq 3\cdot \mathrm{ecost}^*_{D,w}(\mathcal{V}),\end{aligned}$$

where the last inequality follows from Lemma 7.4.8 and the definition of ρ. Since

$$\sum_{\ell=1}^{\tau}\sum_{y_i\in Y_\ell} w(y_i)||y_i - a_\ell|| = \min_{\rho:Y\to A} \mathrm{cost}_{\ell_2,w}(Y, A, \rho) \leq \beta \mathrm{cost}^*_{\ell_2,w}(Y),$$

the claim follows. Similarly, we get that

$$\begin{aligned}&\sum_{\ell=1}^{\tau}\sum_{y_i\in Y_\ell} w(y_i)||y_i - a_\ell|| = \sum_{\ell=1}^{\tau}\sum_{y_i\in Y_\ell}\sum_{j=1}^{m} w(v_i)p_{ij}||y_i - a_\ell|| \\ \leq &\sum_{\ell=1}^{\tau}\sum_{y_i\in Y_\ell}\sum_{j=1}^{m} w(v_i)p_{ij}||y_i - x_j|| + \sum_{\ell=1}^{\tau}\sum_{y_i\in Y_\ell}\sum_{j=1}^{m} w(v_i)p_{ij}||x_j - a_\ell|| \\ \leq &\, 2\cdot \mathrm{ecost}^*_{D,w}(\mathcal{V}) + \mathrm{ecost}_{D,w}(\mathcal{V}, A, \sigma_{\mathcal{V}}) \leq 3\,\mathrm{ecost}_{D,w}(\mathcal{V}, A, \sigma_{\mathcal{V}})\end{aligned}$$

where the second inequality again follows from Lemma 7.4.8. □

Now, we are prepared to bound $\mathrm{ecost}_{D,w}(\mathcal{V}, A, \sigma_{\mathcal{V}})$ and thus prove that A is a bicriteria approximation to $\mathrm{ecost}^*_{D,w}(\mathcal{V})$.

Lemma 7.4.11. *If each $y_i \in Y$ is a 2-approximation of the probabilistic 1-median of v_i and A is an $[\alpha, \beta]$-bicriteria approximation to $\mathrm{cost}^*_{\ell_2,w}(Y)$, then*

$$\mathrm{ecost}_{D,w}(\mathcal{V}, A, \sigma_{\mathcal{V}}) \leq (3\beta + 2)\cdot \mathrm{ecost}^*_{D,w}(\mathcal{V}).$$

Proof. It follows from the triangle inequality and Lemma 7.4.8 and 7.4.10 that

$$\begin{aligned}&\mathrm{ecost}_{D,w}(\mathcal{V}, A, \sigma_{\mathcal{V}}) \leq \sum_{\ell=1}^{\tau}\sum_{y_i\in Y_\ell}\sum_{x_j\in X} w(v_i)p_{ij}\cdot\left(||x_j - y_i|| + ||y_i - a_\ell||\right) \\ = &\sum_{y_i\in Y}\sum_{x_j\in X} w(v_i)p_{ij}||x_j - y_i|| + \sum_{\ell=1}^{\tau}\sum_{y_i\in Y_\ell} w(y_i)||y_i - a_\ell|| \leq (3\beta+2)\cdot \mathrm{ecost}^*_{D,w}(\mathcal{V}).\end{aligned}$$

□

Ring Sets and Their Subsets

In this part, we gain a bound on the diameter. Recall from Lemma 7.4.7 and the plan on page 183 that we want to bound three error terms, and that the second one is

$$\xi \sum_{r=1}^{\lambda} \sum_{v_{i'} \in P_r} w(v_{i'}) \operatorname{diam}(Y(P_r)).$$

We show that the definition of the ring sets ensures that this term is small. This is similar to Claim 3.4 in [Che09] where Chen analyzes the sum of the diameters of the sets in his partitioning. The latter also includes ring sets and is defined in Definition 3.1.1 of [Che09].

We partition Y like Chen partitions the input point set. First, the y_i are partitioned according to their closest center in the bicriteria approximation for Y, i. e., we get a subset for every a_ℓ. Then, we further subdivide each subset into ring sets: We place exponentially growing balls around every a_ℓ. These subdivide the set of points belonging to a_ℓ. Then, the diameter of such a ring set is similar to the cost of each y_i in the ring set, because this cost is lower bounded by the radius of the inner ball.

To show that this works, we have to make sure that our choice of the radius for the smallest ball, R, and the radius of the largest ball, $2^\nu R$, are chosen appropriately.

Recall that we defined $R = (\operatorname{ecost}_{D,w}(\mathcal{V}, A, \sigma_{\mathcal{V}}))/((3\beta+2)W)$ above. By Lemma 7.4.11, we have

$$R = \frac{\operatorname{ecost}_{D,w}(\mathcal{V}, A, \sigma_{\mathcal{V}})}{(3\beta+2)W} \leq \frac{(3\beta+2)\operatorname{ecost}^*_{D,w}(\mathcal{V})}{(3\beta+2)W} = \operatorname{ecost}^*_{D,w}(\mathcal{V})/W.$$

This means that the inner sphere satisfies that the diameter is related to the average optimum cost. In particular, the contribution of the subsets of points lying within the inner spheres to the second error term is bounded by

$$\sum_{\ell=1}^{\tau} \sum_{v_{i'} \in \mathcal{V}_{\ell,1}} w(v_{i'}) \operatorname{diam}(Y(\mathcal{V}_{\ell,1})) \leq \sum_{v_i \in \mathcal{V}} w(v_i) 2R \leq 2\operatorname{ecost}^*_{D,w}(\mathcal{V})$$

which is what we need (we look into the other subsets below).

Now we want to find an upper bound (depending on R) for the distance between an arbitrary point in Y and its closest center in A. This distance is bounded by $\operatorname{cost}_{\ell_2,w}(Y, A, \sigma_Y) = \sum_{i=1}^{n} w(y_i)||y_i - \sigma_Y(y_i)||$. By Lemma 7.4.10, $\operatorname{cost}_{\ell_2,w}(Y, A, \sigma_Y) \leq 3\operatorname{ecost}_{D,w}(\mathcal{V}, A, \sigma_{\mathcal{V}}) = (9\beta+6)RW$. Recall that we set ν to $\lceil \log((9\beta+6)W) \rceil$. That means it holds that

$$2^\nu R = 2^{\lceil \log((9\beta+6)W) \rceil} R \geq (9\beta+6)RW \geq \operatorname{cost}_{\ell_2,w}(Y, A, \sigma_Y),$$

so $2^\nu R$ provides the desired bound.

Due to the definition of $Y_{\ell,h}$, the diameter $\operatorname{diam}(Y_{\ell,h})$ is bounded by $2(2^h R)$ for each $\ell \in [\tau]$ and each $h \in \{1, \ldots, \nu\}$, and we saw above that this also holds for $h = 0$. In the following lemma, we use this to conclude proving the bound on the sum of the weighted diameters of all subsets. The following lemma resembles Claim 3.4 in [Che09].

Lemma 7.4.12. *If each $y_i \in Y$ is a 2-approximation of the probabilistic 1-median of v_i and A is an $[\alpha, \beta]$-bicriteria approximation to $\operatorname{cost}^*_{\ell_2,w}(Y)$, then*

$$\sum_{\ell=1}^{\tau}\sum_{h=0}^{\nu}\sum_{y_i\in Y_{\ell,h}} w(y_i)2^h R \leq (6\beta+1)\cdot \operatorname{ecost}^*_{D,w}(\mathcal{V})$$

and

$$\sum_{\ell=1}^{\tau}\sum_{h=0}^{\nu}\sum_{y_i\in Y_{\ell,h}} w(y_i)\operatorname{diam}(Y_{\ell,h}) \leq (12\beta+2)\cdot \operatorname{ecost}^*_{D,w}(\mathcal{V}).$$

Proof. Let $v_i \in \mathcal{V}$ be an arbitrary node, and let $y_i \in Y_{\ell,h}$ for some $\ell \in [t]$ and $h \in \{0,\dots,\nu\}$. Notice that $2^h R = R$ if $h = 0$, and that $||y_i - a_\ell|| \geq 2^{h-1}R$ holds for all $y_i \in Y_{\ell,h}$ if $h \geq 1$. Hence, we have $2^h R \leq 2||y_i - a_\ell|| + R$. It follows that

$$\begin{aligned}
&\sum_{\ell=1}^{t}\sum_{h=0}^{\nu}\sum_{y_i\in Y_{\ell,h}} w(y_i)2^h R \leq \sum_{\ell=1}^{t}\sum_{h=0}^{\nu}\sum_{y_i\in Y_{\ell,h}} w(y_i)\left(2||y_i - a_\ell|| + R\right)\\
&= \sum_{\ell=1}^{t}\sum_{y_i\in Y_\ell} w(y_i)\left(2||y_i - a_\ell|| + R\right) \leq 6\beta \operatorname{ecost}^*_{D,w}(\mathcal{V}) + \frac{\operatorname{ecost}_{D,w}(\mathcal{V}, A, \sigma_{\mathcal{V}})}{3\beta+2}\\
&\leq (6\beta+1)\operatorname{ecost}^*_{D,w}(\mathcal{V}),
\end{aligned}$$

where the second inequality follows from Lemma 7.4.10 and the last inequality follows from Lemma 7.4.11. Now, since $\operatorname{diam}(Y_{\ell,h}) \leq 2(2^h R)$, the above inequality also implies the second part of the claim. □

Up to now, we have seen that the partitioning of our nodes consists of subsets with small diameter and small distance to an arbitrary C. It remains to bound the maximal value of $\sum_{x_j\in\mathcal{X}} w(y_i)p_{ij}||x_j - y_i||$ for each partition. The problem is that the different realizations of v_i do not necessarily lie near y_i but can be arbitrarily far away. However, if they are, then *every* assignment of v_i to a cluster center will have to pay this distance (weighted with the nodes weight). This is the reason why we subdivide the ring sets into subsets with the same behavior regarding $\sum_{x_j\in X} w(v_i)p_{ij}||x_j - y_i||$.

Recall that $\mu = \lceil \log((6\beta+4)W)\rceil$, so $R\cdot 2^\mu \geq R(6\beta+4)W$. That implies for all nodes $v_i \in \mathcal{V}$ that

$$\begin{aligned}
2^\mu R &\geq 2^{\lceil \log((6\beta+4)W)\rceil} R \geq 2(3\beta+2)RW\\
&= 2\cdot \operatorname{ecost}_{D,w}(\mathcal{V}, A, \sigma_{\mathcal{V}}) \geq 2\cdot \operatorname{ecost}^{*,n}_{D,w}(\mathcal{V})\\
&\geq \min_{\rho: V\to Y} \operatorname{ecost}_{D,w}(\mathcal{V}, Y, \rho).
\end{aligned}$$

This means that $w(v_i)\sum_{x_j\in X} p_{ij}\cdot||x_j - y_i|| \leq \min_{\rho:V\to Y}\operatorname{ecost}_{D,w}(\mathcal{V}, Y, \rho) \leq 2^\mu R$, so our partitioning is well-defined and $Y_{\ell,h} = \bigcup_{a\in\{0,\dots,\mu\}} Y_{\ell,h,a}$. The final partitioning ensures that the third error term is bounded.

Lemma 7.4.13. *If each $y_i \in Y$ is a 2-approximation of the probabilistic 1-median of v_i and A is an $[\alpha, \beta]$-bicriteria approximation to $\mathrm{cost}^*_{\ell_2,w}(Y)$, then it holds that*

$$\sum_{\ell=1}^{\tau}\sum_{h=0}^{\nu}\sum_{a=0}^{\mu}\sum_{v_i\in\mathcal{V}_{\ell,h,a}} w(v_i)\max_{y_{i'}\in Y_{\ell,h,a}}\sum_{x_j\in X} p_{i'j}||x_j - y_{i'}|| \leq 5\,\mathrm{ecost}^*_{D,w}(\mathcal{V}).$$

Proof. For $a = 0$, $w(v_i)\cdot\sum_{x_j\in X} p_{ij}\cdot||x_j - y_i|| \leq R$ holds by the definition of $Y_{\ell,h,0}$. For $a \geq 1$, the maximum in each group can only be twice the term for v_i. Thus, we can conclude that

$$\begin{aligned}
&\sum_{\ell=1}^{\tau}\sum_{h=0}^{\nu}\sum_{a=0}^{\mu}\sum_{v_i\in\mathcal{V}_{\ell,h,a}} w(v_i)\max_{y_{i'}\in Y_{\ell,h,a}}\sum_{x_j\in X} p_{i'j}\cdot||x_j - y_{i'}|| \\
\leq\ &\sum_{\ell=1}^{\tau}\sum_{h=0}^{\nu}\sum_{a=0}^{\mu}\sum_{v_i\in\mathcal{V}_{\ell,h,a}} w(v_i)\left(R + 2\sum_{x_j\in X} p_{ij}\cdot||x_j - y_i||\right) \\
\leq\ &\left(\sum_{v_i\in V} w(v_i)\right)\cdot R\ +\ 2\sum_{y_i\in Y} w(y_i)\sum_{x_j\in X} p_{ij}\cdot||x_j - y_i|| \\
\leq\ &\mathrm{ecost}^*_{D,w}(\mathcal{V})\ +\ 4\,\mathrm{ecost}^*_{D,w}(\mathcal{V}),
\end{aligned}$$

where the last inequality follows from the definition of R and Lemma 7.4.8. □

Bounding the Number of Possible Center Locations

So far, we have seen statements that hold for fixed center sets. Now we want to achieve statements that hold for all possible center sets. The problem is that the set of center candidates is $\mathbb{R}^d$, so there are infinitely many possible center sets. Now and up to page 196 we follow Section 4 of [Che09] to resolve this problem. The approach is very similar to Chen's work, but the individual statements and lemmata turn out to be more technical.

The idea is to distinguish between two cases. Either, there is a center very far away from the input node set (more specifically, far away from the centers of our bicriteria approximation for the y_i), or all nodes (or at least their approximate 1-medians) lie within a bounded area. In the first case, we get a sufficiently high lower bound on the clustering cost which allows us to show that the error of U is small in comparison. In the second case, we can discretize the bounded area by a grid and show that shifting centers to the grid does not change the cost too much. The grid then has a finite and small enough number of centers such that we can use the union bound to bound the failure probability for all possible center sets. We use the same grid as Chen, which is defined as follows.

Definition 7.4.14 (Definition 4.2 in [Che09])**.** *Let $\Phi \in \mathbb{N}^+$ and $b \in \mathbb{N}^+$ be parameters. Let $\mathfrak{U} := \bigcup_{\ell=1}^{\tau} B(a_\ell, 2^\Phi R)$ be the union of large enough spheres around all centers in A. For $\ell \in [\tau]$ and $h \in \{0, \ldots, \Phi\}$, define*

$$L_{\ell,h} := \begin{cases} B(a_\ell, R) & h = 0 \\ B(a_\ell, 2^h R)\backslash B(a_\ell, 2^{h-1}R) & h \geq 1 \end{cases}.$$

For $h \in \{0, \dots, \Phi\}$, set $r_h := 2^h \varepsilon R/(b\beta\sqrt{d})$. For $\ell \in [\tau]$ and $h \in \{0, \dots, \Phi\}$, consider an axis-parallel grid with side length r_h to partition $L_{\ell,h}$ into cells. Pick an arbitrary point from each grid cell in $L_{\ell,h}$ and store these representative points in $\mathfrak{G}_{\ell,h}$. Finally, set $\mathfrak{G} := \bigcup_{\ell=1}^{\tau} \bigcup_{h=0}^{\Phi} \mathfrak{G}_{\ell,h}$.

Chen bounds the number of cells in this grid by a volume argument, noticing that each cell that has an overlap with $B(a_\ell, 2^h)$ does lie completely within $B(a_\ell, 2^{h+1})$. Thus the number of cells can be bounded by dividing the volume of $B(a_\ell, 2^{h+1})$ by the volume of a cell. By simplifying the resulting term, Chen obtains the following.

Lemma 7.4.15 (Claim 4.3 in [Che09])**.** *Let $\ell \in [\tau]$ and $h \in \{0, \dots, \Phi\}$ be given. The number of cells that overlap with $L_{\ell,h}$ is bounded by $((4\sqrt{\pi e}b\beta)/(\varepsilon))^d$. This implies that the numer of points in $\mathfrak{G}$ satisfies that $|\mathfrak{G}| \leq \tau(\Phi+1)\left((4\sqrt{\pi e}b\beta)/(\varepsilon)\right)^d$.*

We set the parameters Φ and b differently from Chen, so we get a grid of a different size.

Corollary 7.4.16. *For $\Phi := \lceil \log(18(3\beta+2)W/\varepsilon) \rceil$, and $b := 390$, we have $\ln(|\mathfrak{G}|) = \mathcal{O}(\log(k) + \log(\log(W/\varepsilon)) + d\log(1/\varepsilon))$.*

Proof. Recall that $\tau = \alpha k$ with $\alpha = \mathcal{O}(\log(W/\varepsilon))$ and that β is a constant. Then it follows by Lemma 7.4.15 and the definiton of Φ and b that

$$\begin{aligned}
\ln(|\mathfrak{G}|) &\leq \ln\left(\alpha k \left(\lceil \log(18(3\beta+2)W/\varepsilon)\rceil + 1\right)\left(\frac{4\sqrt{\pi e}390\beta}{\varepsilon}\right)^d\right) \\
&= \ln\left(k\mathcal{O}(\log[W/\varepsilon]) \cdot \left(\lceil \log[18(3\beta+2)W/\varepsilon]\rceil + 1\right)\left(\frac{4\sqrt{\pi e}390\beta}{\varepsilon}\right)^d\right) \\
&= \mathcal{O}(\log(k) + \log(\log(W/\varepsilon)) + d\log(1/\varepsilon)).
\end{aligned}$$

□

Now we proceed in the following way. First, we show that we can ensure that the coreset property holds when the centers lie on arbitrary points in $\mathfrak{G}$. Then we show that the coreset property holds for points outside $\mathfrak{U}$ anyway because they are so far away that the cost is so high that our error is always small in comparison. Finally, we see that coreset property for all center sets from $\mathfrak{U}$, even if the centers do not lie on grid points.

We start by ensuring the coreset property for center sets that consist of grid points. We do this by using our results from Section 7.4.2 that hold for one center set. The bound on $\ln(\mathfrak{G})$ means that we can set the failure probabilities in a way that ensures no failure occurs for any of the possible choices of k centers from $\mathfrak{G}$ with high probability.

Lemma 7.4.17. *Let $\mathcal{V}$ be a set of nodes, let k be an integer and let $\mathcal{U}$ be the sample set weighted by w' which is obtained by Step (2) of our coreset construction (see page 177). For all sets C of at most k centers chosen from $\mathfrak{G}$, it holds that*

$$\left|\min_{\rho:V\to C} \operatorname{ecost}_{D,w}(\mathcal{V}, C, \rho) - \min_{\rho:\mathcal{U}\to C} \operatorname{ecost}_{D,w'}(\mathcal{U}, C, \rho)\right| \leq \varepsilon/5 \cdot \min_{\rho:\mathcal{V}\to C} \operatorname{ecost}_{D,w}(\mathcal{V}, C, \rho)$$

with probability at least $1 - \delta/2$.

Proof. Fix an arbitrary set $C \subseteq X$ of size at most k. Due to Lemma 7.4.7, setting $\xi := \varepsilon/(5(12\beta+10))$ and $\delta' := |\mathfrak{G}|^{-k}\delta/(2\tau(\nu+1)(\mu+1))$, it holds that

$$\begin{aligned}&\left|\min_{\rho:\mathcal{V}_{\ell,h,a}\to C} \operatorname{ecost}_{D,w}(\mathcal{V}_{\ell,h,a}, C, \rho) - \min_{\rho:\mathcal{U}_{\ell,h,a}\to C} \operatorname{ecost}_{D,w'}(\mathcal{U}_{\ell,h,a}, C, \rho)\right| \\ \leq &\frac{\varepsilon}{5(12\beta+10)} \sum_{v_i\in\mathcal{V}_{\ell,h,a}} w(v_i)(\operatorname{dist}(Y_{\ell,h,a}, C) + \operatorname{diam}(Y_{\ell,h,a}) + \max_{y_i\in Y_{\ell,h,a}} \sum_{x_j\in\mathcal{X}} p_{ij}\cdot||x_j - y_i||)\end{aligned}$$

with probability at least $1-\delta'$ for each choice of $\ell \in [\tau]$, $h \in \{0,\dots,\nu\}$, and $a \in \{0,\dots,\mu\}$. Due to the Union Bound, the above inequality is true with probability $1-\tau(\nu+1)(\mu+1)\delta' = 1-|\mathfrak{G}|^{-k}\delta/2$ for all $\mathcal{V}_{\ell,h,a}$ simultaneously. Since there are at most $|\mathfrak{G}|^k$ different ways to select a set C of size at most k from $\mathfrak{G}$, the above inequality holds for every such set C with probability at least $1-\delta/2$.

The required sample size is

$$\begin{aligned}s'' = \left\lceil \xi^{-2}\ln(2/\delta')\right\rceil &= \left\lceil \left(\frac{12\beta+10}{\varepsilon}\right)^2 \ln\left(2\frac{\tau(\nu+1)(\mu+1)|\mathfrak{G}|^k}{\delta}\right)\right\rceil \\ &\leq \left\lceil c'\varepsilon^{-2}\left[\log(1/\delta) + k\log(|\mathfrak{G}|) + \log k\log W/\varepsilon\right]\right\rceil \\ &\leq \left\lceil c\varepsilon^{-2}\left[\log(1/\delta) + k\left(\log(k) + \log(\log(W/\varepsilon)) + d\log(1/\varepsilon)\right)\right]\right\rceil = s'\end{aligned}$$

for some sufficiently large constants c' and c. To complete the proof, we calculate the sum of the errors of all subsets $\mathcal{V}_{\ell,h,a}$ for all $\ell \in [\tau]$, $h \in \{0,\dots,\nu\}$, and $a \in \{0,\dots,\mu\}$. For the following calculation, we use $\varepsilon' := \varepsilon/(5(12\beta+10))$.

$$\begin{aligned}&\left|\min_{\rho:V\to C} \operatorname{ecost}_{D,w}(\mathcal{V}, C, \rho) - \min_{\rho:\mathcal{U}\to C} \operatorname{ecost}_{D,w'}(\mathcal{U}, C, \rho)\right| \\ \leq\ &\sum_{\ell=1}^{\tau}\sum_{h=0}^{\nu}\sum_{a=0}^{\mu}\left|\min_{\rho:V_{\ell,h,a}\to C} \operatorname{ecost}_{D,w}(\mathcal{V}_{\ell,h,a}, C, \rho) - \min_{\rho:U_{\ell,h,a}\to C} \operatorname{ecost}_{D,w'}(\mathcal{U}_{\ell,h,a}, C, \rho)\right| \\ \leq\ &\sum_{\ell=1}^{\tau}\sum_{h=0}^{\nu}\sum_{a=0}^{\mu}\varepsilon'\cdot\sum_{v_i\in V_{\ell,h,a}} w(v_i)(\operatorname{dist}(Y_{\ell,h,a}, C) + \operatorname{diam}(Y_{\ell,h,a}) + \max_{y_{i'}\in Y_{\ell,h,a}}\sum_{x_j\in\mathcal{X}} p_{i'j}||x_j - y_{i'}||) \\ \leq\ &\varepsilon'\left(3\min_{\rho:V\to C}\operatorname{ecost}_{D,w}(\mathcal{V}, C, \rho) + \sum_{\ell=1}^{\tau}\sum_{h=0}^{\nu}\sum_{y_i\in Y_{\ell,h}} w(y_i)\operatorname{diam}(Y_{\ell,h}) \ +\ 5\operatorname{ecost}^*_{D,w}(\mathcal{V})\right) \\ \leq\ &\frac{\varepsilon}{5(12\beta+10)}\left(3\min_{\rho:V\to C}\operatorname{ecost}_{D,w}(\mathcal{V}, C, \rho) \ +\ (12\beta+2)\operatorname{ecost}^*_{D,w}(\mathcal{V}) \ +\ 5\operatorname{ecost}^*_{D,w}(\mathcal{V})\right) \\ \leq\ &\frac{\varepsilon}{5}\cdot\min_{\rho:V\to C}\operatorname{ecost}_{D,w}(\mathcal{V}, C, \rho),\end{aligned}$$

where the third inequality follows from Lemma 7.4.9 and Lemma 7.4.13, and the penultimate inequality follows from Lemma 7.4.12. □

Now we deal with the case that the centers can be chosen arbitrarily. First, we look at center sets that contain at least one center which is far away, i. e., lies outside $\mathfrak{U}$. We do

this in three steps. First, we show that a center outside $\mathfrak{U}$ implies a sufficiently high lower bound on the optimal cost, more precisely, it implies that $\min_{\rho:\mathcal{V}\to C} \operatorname{ecost}_{D,w}(\mathcal{V}, C, \rho) > 17/\varepsilon \operatorname{ecost}_{D,w}(\mathcal{V}, A, \sigma_{\mathcal{V}})$. Second, we rewrite the coreset cost into a more convenient form by shifting between realization probabilities and weights. Third, we show that (in any case, not only if there is a center outside of $\mathfrak{U}$), the error of the coreset is bounded by $17 \cdot \operatorname{ecost}_{D,w}(\mathcal{V}, A, \sigma_{\mathcal{V}})$. Together, we get the coreset result for center sets with at least one center outside of $\mathfrak{U}$.

The main technical step is to bound the error of $\mathcal{U}$ for all possible center sets by using the geometric properties of the partitioning from which we sample. We get the following high upper bound.

Lemma 7.4.18. *Let $C \subset \mathbb{R}^d$ be an arbitrary center set with k centers. It holds that*

$$\Big| \min_{\rho:\mathcal{V}\to C} \operatorname{ecost}_{D,w}(\mathcal{V}, C, \rho) - \min_{\rho:U\to C} \operatorname{ecost}_{D,w'}(\mathcal{U}, C, \rho) \Big| \leq 17 \cdot \operatorname{ecost}_{D,w}(\mathcal{V}, A, \sigma_{\mathcal{V}}).$$

Additionally, it holds that $\operatorname{ecost}_{D,w'}(\mathcal{U}, A, \sigma_{\mathcal{V}}) \leq (6\beta + 6) \operatorname{ecost}^*_{D,w}(\mathcal{V})$.

Proof. Recall that the total weight of all nodes in $\mathcal{U}_{\ell,h,a}$ is exactly the total weight $W_{\ell,h,a} := \sum_{v_i \in \mathcal{V}_{\ell,h,a}} w(v_i)$ of all nodes in $\mathcal{V}_{\ell,h,a}$. We like to map each node in $\mathcal{V}_{\ell,h,a}$ to the node in $\mathcal{U}_{\ell,h,a}$ that represents it. So, we want to split $V_{\ell,h,a}$ into s' disjoint subsets of nodes $\mathcal{V}_{\ell,h,a,1}, \dots, V_{\ell,h,a,s'}$ such that each $v_{i'} \in U_{\ell,h,a}$ covers a subset $\mathcal{V}_{\ell,h,a,z}$ with $z \in [s']$. To do so, we start with $z = 1$ and put arbitrary nodes from $\mathcal{V}_{\ell,h,a}$ into $V_{\ell,h,a,z}$ until the chosen nodes have a weight of at least $W_{\ell,h,a}/s'$. If the last added node exceeded $W_{\ell,h,a}/s'$, then we reduce its weight such that the sum of all nodes in $\mathcal{V}_{\ell,h,a,z}$ is exactly $W_{\ell,h,a}/s'$, create a copy with the remaining weight and start $\mathcal{V}_{\ell,h,a,z+1}$ with it.

During this process, we may split nodes into two or or more copies, but we do not change the clustering cost. We now assume that $\mathcal{V}$ is replaced by the set of this splitted nodes. We use the mapping $\pi : [n] \to [n]$ which is defined such that the coreset node $v_{\pi(i)}$ covers v_i. Using this notation, we can rewrite the cost of $\mathcal{U}$ when clustering with a mapping ρ by

$$\begin{aligned}
\operatorname{ecost}_{D,w'}(\mathcal{U}, C, \rho) &= \sum_{\ell=1}^{\tau} \sum_{h=0}^{\nu} \sum_{a=0}^{\mu} \sum_{v_{i'} \in \mathcal{U}_{\ell,h,a}} w'(v_{i'}) \left[\sum_{x_j \in X} p_{i'j} ||x_j - \rho(v_{i'})|| \right] \\
&= \sum_{\ell=1}^{\tau} \sum_{h=0}^{\nu} \sum_{a=0}^{\mu} \sum_{z} \sum_{v_i \in V_{\ell,h,a,z}} w(v_i) \left[\sum_{x_j \in X} p_{\pi(i)j} ||x_j - \rho(v_{\pi(i)})|| \right] \\
&= \sum_{\ell=1}^{\tau} \sum_{h=0}^{\nu} \sum_{a=0}^{\mu} \sum_{v_i \in V_{\ell,h,a}} w(v_i) \left[\sum_{x_j \in X} p_{\pi(i)j} ||x_j - \rho(v_{\pi(i)})|| \right]. \qquad (7.2)
\end{aligned}$$

This clustering cost looks very much like the clustering cost of $\mathcal{V}$, except for the two occurences of $\pi(i)$. The error is

$$\begin{aligned}
&\Big| \min_{\rho:\mathcal{V}\to C} \operatorname{ecost}_{D,w}(\mathcal{V}, C, \rho) - \min_{\rho:\mathcal{U}\to C} \operatorname{ecost}_{D,w'}(\mathcal{U}, C, \rho) \Big| \\
=&\Big| \sum_{\ell=1}^{\tau} \sum_{h=0}^{\nu} \sum_{a=0}^{\mu} \sum_{v_i \in \mathcal{V}_{\ell,h,a}} w(v_i) \sum_{x_j \in X} \Big[p_{ij} ||x_j - \sigma(v_i)|| - p_{\pi(i)j} ||x_j - \sigma(v_{\pi(i)})|| \Big] \Big| \qquad (7.3)
\end{aligned}$$

where $\sigma : \mathcal{V} \to C$ is an assignment that minimizes the clustering cost for C, which also minimizes the clustering cost of $\mathcal{U}$ with C (because it just assigns every node in $\mathcal{V}$ to the center that minimizes the expected distance). We can replace $p_{ij}||x_j - \sigma(v_i)|| - p_{\pi(i)j}||x_j - \sigma(v_{\pi(i)})|$ by the maximum of the two distances since both are positive. To denote the resulting term, let $\pi'(i) \in \{i, \pi(i)\}$ be such that $p_{\pi'(i)j}||x_j - \sigma(v_{\pi'(i)})||$ is the larger term.

Now we use a repeated application of the triangle inequality which reduces the error to terms that we know. For each ℓ, h, a and $v_i \in \mathcal{V}_{\ell,h,a}$ notice that

$$\begin{aligned}
&\sum_{x_j \in X} p_{\pi'(i)j}||x_j - \sigma(v_{\pi'(i)})|| \leq \sum_{x_j \in X} p_{\pi'(i)j}||x_j - \sigma(v_i)|| \\
\leq &\sum_{x_j \in X} p_{\pi'(i)j}\Big(||x_j - y_{\pi'(v_i)}|| + ||y_{\pi'(v_i)} - y_i|| + ||y_i - \sigma(v_i)||\Big) \\
\leq &||y_{\pi'(v_i)} - y_i|| + ||y_i - \sigma(v_i)|| + \sum_{x_j \in X} p_{\pi'(i)j}||x_j - y_{\pi'(v_i)}|| \\
\leq &||y_{\pi'(v_i)} - y_i|| + \sum_{x_j \in X} p_{ij}||y_i - \sigma(v_i)|| + \sum_{x_j \in X} p_{\pi'(i)j}||x_j - y_{\pi'(v_i)}|| \\
\leq &||y_{\pi'(v_i)} - y_i|| + \sum_{x_j \in X} p_{ij}\Big(||y_i - x_j|| + ||x_j - \sigma(v_i)||\Big) + \sum_{x_j \in X} p_{\pi'(i)j}||x_j - y_{\pi'(v_i)}|| \\
\leq &||y_{\pi'(v_i)} - y_i|| + \sum_{x_j \in X} p_{ij}||x_j - \sigma(v_i)|| + 3 \cdot \sum_{x_j \in X} p_{ij}||x_j - y_i|| + R. \qquad (7.4)
\end{aligned}$$

where the last inequality follows since v_i and $v_{\pi'(i)}$ are in the same subset, and all nodes $v_{i'}$ and $v_{i''}$ from the same subset $\mathcal{V}_{\ell,h,a}$ satisfy

$$-R + \frac{1}{2} \sum_{x_j \in X} p_{i''j}||x_j - y_{i''}|| < \sum_{x_j \in X} p_{i'j}||x_j - y_{i'}|| < R + 2 \sum_{x_j \in X} p_{i''j}||x_j - y_{i''}|| \qquad (7.5)$$

by the definition of the subsets. As v_i and $v_{\pi'(v_i)}$ are also in the same ring set $\mathbf{V}_{\ell,h}$, we can bound $||y_{\pi'(v_i)} - y_i||$ in a similar fashion: In the inner sphere, $||y_{\pi'(v_i)} - y_i|| \leq R$. Otherwise, the distance of both $y_{\pi'(v_i)}$ and y_i to a_ℓ is at least $2^{h-1}R$ and at most $2^h R$. Thus, $||y_{\pi'(v_i)} - y_i|| \leq ||y_{\pi'(v_i)} - a_\ell|| + ||y_i - a_\ell|| \leq 2^h R + ||y_i - a_\ell|| \leq 3||y_i - a_\ell||$. So, for all h, we have $||y_{\pi'(v_i)} - y_i|| \leq R + 3||y_i - a_\ell||$. We combine this with the error term (7.3) and the bounds in (7.4) to get

$$\begin{aligned}
&\Big|\min_{\rho:\mathcal{V}\to C} \operatorname{ecost}_{D,w}(\mathcal{V}, C, \rho) - \min_{\rho:\mathcal{U}\to C} \operatorname{ecost}_{D,w'}(\mathcal{U}, C, \rho)\Big| \\
\leq &\sum_{\ell=1}^{\tau} \sum_{v_i \in \mathcal{V}_\ell} w(v_i)\Big(3||y_i - a_\ell|| + \sum_{x_j \in X} p_{ij}||x_j - \sigma(v_i)|| + 3 \cdot \sum_{x_j \in X} p_{ij}||x_j - y_i|| + 2R\Big) \\
\leq &3 \cdot 3\operatorname{ecost}_{D,w}(\mathcal{V}, A, \sigma_{\mathcal{V}}) + \sum_{\ell=1}^{\tau} \sum_{v_i \in \mathcal{V}_\ell} w(v_i)\Big(\sum_{x_j \in X} p_{ij}||x_j - \sigma(v_i)|| + 3\sum_{x_j \in X} p_{ij}||x_j - y_i|| + 2R\Big) \\
\leq &10 \cdot \operatorname{ecost}_{D,w}(\mathcal{V}, A, \sigma_{\mathcal{V}}) + 6\operatorname{ecost}^*_{D,w}(\mathcal{V}) + \sum_{\ell=1}^{\tau} \sum_{v_i \in \mathcal{V}_\ell} w(v_i)2R \leq 17 \cdot \operatorname{ecost}_{D,w}(\mathcal{V}, A, \sigma_{\mathcal{V}})
\end{aligned}$$

where the second inequality follows because Lemma 7.4.10 says that $\sum_{\ell=1}^{\tau}\sum_{y_i\in Y_\ell} w(y_i)\cdot ||y_i - a_\ell|| \leq 3\,\mathrm{ecost}_{D,w}(\mathcal{V}, A, \sigma_{\mathcal{V}})$, and the third inequality follows because Lemma 7.4.8 says that $\sum_{i=1}^{n}\sum_{j=1}^{m} w(v_i)p_{ij}\cdot||x_j - y_i|| \leq 2\cdot \mathrm{ecost}^{*,n}_{D,w}(\mathcal{V}) \leq 2\cdot\mathrm{ecost}^{*}_{D,w}(\mathcal{V})$ and because the middle term is just the clustering cost of $\mathcal{V}$ with C since σ is chosen optimally.

In order to prove the second statement, notice that similar to (7.2), we can rewrite the clustering cost of $\mathcal{U}$ with A by

$$\mathrm{ecost}_{D,w'}(\mathcal{U}, A, \sigma_{\mathcal{V}}) = \sum_{\ell=1}\sum_{h=0}^{\nu}\sum_{a=0}^{\mu}\sum_{v_i\in\mathcal{V}_{\ell,h,a}} w(v_i)\Big[\sum_{x_j\in X} p_{\pi(i)j}||x_j - \sigma_{\mathcal{V}}(v_{\pi(i)})||\Big]. \tag{7.6}$$

Then we observe that it holds forall $v_i \in \mathcal{V}_{\ell,h,a}$ for $\ell\in[\tau], h\in\{0,\dots,\nu\}$ and $a\in\{0,\dots,\mu\}$ by the triangle inequality and (7.5) that

$$\begin{aligned}\sum_{x_j\in X} p_{\pi(i)j}||x_j - \sigma_{\mathcal{V}}(v_{\pi(i)})|| &\leq \sum_{x_j\in X} p_{\pi(i)j}\Big(||x_j - y_{\pi(i)}|| + ||y_{\pi(i)} - \sigma_{\mathcal{V}}(v_{\pi(i)})||\Big)\\ &\leq R + 2\sum_{x_j\in X} p_{ij}||x_j - y_i|| + \sum_{x_j\in X} p_{\pi(i)j}||y_{\pi(i)} - \sigma_{\mathcal{V}}(v_{\pi(i)})||\\ &\leq R + 2\sum_{x_j\in X} p_{ij}||x_j - y_i|| + \sum_{x_j\in X} p_{\pi(i)j}2^h R\end{aligned}$$

Now we combine this with (7.6) and then apply Lemma 7.4.8, Lemma 7.4.12 and the fact that $WR \leq \mathrm{ecost}^{*}_{D,w}(\mathcal{V})$ by the definition of R to obtain that

$$\begin{aligned}&\mathrm{ecost}_{D,w'}(\mathcal{U}, A, \sigma_{\mathcal{V}})\\ &\leq WR + 2\sum_{\ell=1}\sum_{h=0}^{\nu}\sum_{a=0}^{\mu}\sum_{v_i\in\mathcal{V}_{\ell,h,a}} w(v_i)\sum_{x_j\in X} p_{ij}||x_j - y_i|| + \sum_{\ell=1}\sum_{h=0}^{\nu}\sum_{a=0}^{\mu}\sum_{v_i\in\mathcal{V}_{\ell,h,a}} w(v_i)2^h R\\ &\leq \mathrm{ecost}^{*}_{D,w}(\mathcal{V}) + 4\cdot\mathrm{ecost}^{*}_{D,w}(\mathcal{V}) + (6\beta+1)\cdot\mathrm{ecost}^{*}_{D,w}(\mathcal{V}) \leq (6\beta+6)\,\mathrm{ecost}^{*}_{D,w}(\mathcal{V}).\end{aligned}$$

That completes the proof for the second statement. □

Lemma 7.4.19. *Let $\mathcal{V}$ be a set of nodes, let k be an integer and let $\mathcal{U}$ be the sample set weighted by w' which is obtained by Step (2) of our coreset construction (see page 177). Let $C\subseteq\mathbb{R}^d$ with a center $c\in C$ that is outside $\mathfrak{U}$. Then*

$$\left|\min_{\rho:\mathcal{V}\to C}\mathrm{ecost}_{D,w}(\mathcal{V}, C, \rho) - \min_{\rho:\mathcal{U}\to C}\mathrm{ecost}_{D,w'}(\mathcal{U}, C, \rho)\right| \leq \varepsilon\cdot\min_{\rho:\mathcal{V}\to C}\mathrm{ecost}_{D,w}(\mathcal{V}, C, \rho).$$

Proof. If there is no $v_i\in\mathcal{V}$ which is assigned to c in the optimal clustering of $\mathcal{V}$, then c can be ignored. Thus, assume that there exists a node v_i that is assigned to c, i. e., $c\in\arg\min_{c'\in C}\sum_{j=1}^{m} w(v_i)p_{ij}||x_j - c'||$. Let a_ℓ be the center in A to which v_i is assigned by

$\sigma_{\mathcal{V}}$. Since $c \notin \mathfrak{U}$, we have $||a_\ell - c|| > 2^\Phi R$. Hence, we get the following lower bound on the optimal clustering cost:

$$\min_{\rho:V\to C} \operatorname{ecost}_{D,w}(\mathcal{V}, C, \rho) \geq \sum_{j=1}^{m} w(v_i)p_{ij}||x_j - c|| \geq \sum_{j=1}^{m} w(v_i)p_{ij}\left(||a_\ell - c|| - ||x_j - a_\ell||\right)$$
$$> w(v_i)2^\Phi R - \sum_{j=1}^{m} w(v_i)p_{ij}||x_j - a_\ell|| \geq (18(3\beta+2)W/\varepsilon)\cdot R - \operatorname{ecost}_{D,w}(\mathcal{V}, A, \sigma_{\mathcal{V}})$$
$$\geq \frac{18}{\varepsilon}\cdot \operatorname{ecost}_{D,w}(\mathcal{V}, A, \sigma_{\mathcal{V}}) - \operatorname{ecost}_{D,w}(\mathcal{V}, A, \sigma_{\mathcal{V}}) \geq \frac{17}{\varepsilon}\cdot \operatorname{ecost}_{D,w}(\mathcal{V}, A, \sigma_{\mathcal{V}}).$$

We get

$$\operatorname{ecost}_{D,w}(\mathcal{V}, A, \sigma_{\mathcal{V}}) < \frac{\varepsilon}{17} \min_{\rho:V\to C} \operatorname{ecost}_{D,w}(\mathcal{V}, C, \rho). \tag{7.7}$$

On the other hand, we can bound the error of our coreset by $\min_{\rho:V\to C} \operatorname{ecost}_{D,w}(\mathcal{V}, C, \rho)$ by Lemma 7.4.18. We conclude that if a center lies outside of $\mathcal{U}$, it holds that

$$\left|\min_{\rho:V\to C} \operatorname{ecost}_{D,w}(\mathcal{V}, C, \rho) - \min_{\rho:U\to C} \operatorname{ecost}_D w'(\mathcal{U}, C, \rho)\right|$$
$$< 17\cdot \operatorname{ecost}_D w(\mathcal{V}, A, \sigma_{\mathcal{V}}) \leq \varepsilon \min_{\rho:V\to C} \operatorname{ecost}_{D,w}(\mathcal{V}, C, \rho)$$

which concludes the proof. □

Lemma 7.4.19 implies that for all C with a center c outside $\mathfrak{U}$, our sampling set is already a coreset. Thus, we only have to deal with the case that $C \subseteq \mathfrak{U}$. Therefore, suppose that $C = \{c_1, \ldots, c_k\} \subseteq \mathfrak{U}$. Let $C' = \{c'_1, \ldots, c'_k\}$, where $c'_t \in \mathfrak{G}$ is the representative point of the cell containing c_t for $t \in [k]$. Now, we want to show that clustering with C' is not much different from clustering with C. In other words, we want to show that the difference between the optimal clustering costs with C and C' is small. We start with showing this statement for one fixed node v_i. The main fact to show the following lemma is that for the optimal center for v_i in C, we can find a slightly shifted center in C', and the distance between these two can be bounded by the diameter of a cell in our grid.

Lemma 7.4.20. *Let $\mathcal{V}$ be a set of nodes, let k be an integer and let $\mathcal{U}$ be the sample set weighted by w' which is obtained by Step (2) of our coreset construction (see page 177). For every center set $C \subset \mathfrak{U}$ with $|C| = k$ there exists a center set $C_G \subset \mathfrak{G}$ with $|C_G| \leq k$ such that*

$$\left|\min_{\rho:\mathcal{V}\to C} \operatorname{ecost}_{D,w}(\mathcal{V}, C, \rho) - \min_{\rho:\mathcal{V}\to C} \operatorname{ecost}_{D,w}(\mathcal{V}, C_G, \rho)\right| \leq (\varepsilon/10)\cdot \min_{\rho:\mathcal{V}\to C} \operatorname{ecost}_{D,w}(\mathcal{V}, C, \rho)$$

and

$$\left|\min_{\rho:\mathcal{U}\to C} \operatorname{ecost}_{D,w'}(\mathcal{U}, C, \rho) - \min_{\rho:\mathcal{U}\to C} \operatorname{ecost}_{D,w'}(\mathcal{U}, C_G, \rho)\right| \leq (\varepsilon/2)\cdot \min_{\rho:\mathcal{V}\to C} \operatorname{ecost}_{D,w}(\mathcal{V}, C, \rho).$$

Proof. Each $c \in C$ lies in a cell, and this cell has a representative in $\mathfrak{G}$. We define a mapping $g : C \to \mathfrak{G}$ that sets $g(c)$ to this representative. The center set $C_G := \{g(c) \mid c \in C\}$ is the set of these representatives. For each $c_g \in C_G$, we set $u(c_g)$ to a center with $g(c) = c_g$. Now we show that C and C_G have approximately the same clustering cost with respect to $\mathcal{V}$ and also with respect to $\mathcal{U}$.

Let $v_i \in \mathcal{V}_{\ell,h}$ for $l \in [\tau]$, $h \in \{0, \dots, \nu\}$ be a node let $c^* \in C$ and $c_g^* \in C_G$ be centers that minimize the clustering cost of v_i. Then the clustering cost of $\mathcal{V}$ with C differs from the clustering cost of C_G by at most $w(v_i) \cdot \max\{||c^* - g(c^*)||, ||c_g^* - u(c_g^*)||\}$. Assume c^* has the higher clustering cost. Since $u(c_g^*)$ is in C, clustering with it can only be more expensive than clustering with c^*. Thus, we have by the triangle inequality that

$$\begin{aligned}
&\Big|\min_{c\in C}\sum_{x_j\in X} w(v_i)p_{ij}||x_j - c|| - \min_{c\in C_g}\sum_{x_j\in X} w(v_i)p_{ij}||x_j - c_g||\Big| \\
\le &\sum_{x_j\in X} w(v_i)p_{ij}||x_j - u(c_g^*)|| - \sum_{x_j\in X} w(v_i)p_{ij}||x_j - c_g^*|| \\
\le &\sum_{x_j\in X} w(v_i)p_{ij}||x_j - c_g^*|| + \sum_{x_j\in X} w(v_i)p_{ij}||c_g^* - u(c_g^*)|| - \sum_{x_j\in X} w(v_i)p_{ij}||x_j - c_g^*|| \\
= &w(v_i) \cdot ||c_g^* - u(c_g^*)||.
\end{aligned}$$

If c_g^* has the same clustering cost, we can make the same calculations with the difference that we replace c_g^* by $u(c^*)$. The rest then follows analogously.

So it suffices to bound $w(v_i)$ times the distance between c and its representative $g(c)$ for any $c \in C$. This distance is bounded by $\sqrt{d}$ times the cell width of the cell that c lies in, and this cell width depends on the distance between c and its closest center in A.

Let h' be such that $c \in L_{\ell,h'}$ (recall that $v_i \in \mathcal{V}_{\ell,h}$). Then the cell width is defined as $r_{h'} = 2^{h'}\varepsilon R/(390\beta\sqrt{d})$, so $r_{h'}\sqrt{d} = 2^{h'}\varepsilon R/(390\beta)$.

If $h' = 0$, then $r_{h'}\sqrt{d} = \varepsilon R/(390\beta)$. Otherwise, c is at least $2^{h'-1}R$ away from a_ℓ, so

$$r_{h'}\sqrt{d} \le 2 \cdot 2^{h'-1}R\varepsilon/(390\beta) \le 2\varepsilon||c - a_\ell||/(390\beta) \le 2\varepsilon/(196\beta)(||c - x_j|| + ||x_j - a_\ell||)$$

for any $x_j \in X$. It follows that

$$\begin{aligned}
&|\min_{\rho:\mathcal{V}\to C} \operatorname{ecost}_{D,w}(\mathcal{V}, C, \rho) - \min_{\rho:\mathcal{V}\to C} \operatorname{ecost}_{D,w}(\mathcal{V}, C_G, \rho)| \\
\le &\sum_{i=1}^{n} w(v_i)\varepsilon R/(390\beta) + \min_{\rho:\mathbf{V}\to C}\sum_{j=1}^{m} p_{ij}w(v_i)2\varepsilon/(196\beta)(||c - x_j|| + ||x_j - a_\ell||)\Big) \\
\le &\frac{\varepsilon}{390\beta}\Big(WR + 2\min_{\rho:\mathcal{V}\to C} \operatorname{ecost}_{D,w}(\mathcal{V}, C, \rho) + 2\operatorname{ecost}_{D,w}(\mathcal{V}, A, \sigma_{\mathcal{V}})\Big) \\
\le &(\varepsilon/10)\min_{\rho:\mathcal{V}\to C} \operatorname{ecost}_{D,w}(\mathcal{V}, C, \rho)
\end{aligned}$$

by Lemma 7.4.11 and by the fact that it holds $R = \operatorname{ecost}_{D,w}(\mathcal{V}, A, \sigma_{\mathcal{V}})/((3\beta + 2)W) \le \operatorname{ecost}^*_{D,w}(\mathcal{V})/W$. This proves the first statement of the lemma. For the coreset, we get

$$|\min_{\rho:\mathcal{U}\to C} \operatorname{ecost}_{D,w'}(\mathcal{U}, C, \rho) - \min_{\rho:\mathcal{U}\to C} \operatorname{ecost}_{D,w'}(\mathcal{U}, C_G, \rho)|$$

$$\leq \sum_{v_{i'} \in \mathcal{U}} w'(v_{i'}) \Big(\varepsilon R/(390\beta) + \min_{\rho:\mathcal{U}\to C} \sum_{j=1}^{m} p_{i'j} w'(v_{i'}) 2\varepsilon/(196\beta)(||c - x_j|| + ||x_j - a_\ell||) \Big)$$

$$\leq \frac{\varepsilon}{390\beta} \Big(RW + 2 \min_{\rho\mathcal{U}\to C} \operatorname{ecost}_{D,w'}(\mathcal{U}, C, \rho) + 2 \min_{\rho:\mathcal{U}\to C} \sum_{v_{i'} \in \mathcal{U}} \sum_{j=1}^{m} p_{i'j} w'(v_{i'}) ||x_j - a_\ell|| \Big)$$

since the coreset nodes are input nodes (with different weight) and since the sum of the weights is the same for $\mathcal{V}$ and $\mathcal{U}$. We can now use Lemma 7.4.18. The first statement of the lemma applies to the middle term, and the second statement of the lemma applies to the last term. For the first term, we use that $RW = \operatorname{ecost}_{D,w}(\mathcal{V}, A, \sigma_V)/(3\beta+2) \leq \operatorname{ecost}^*_{D,w}(\mathcal{V})$ by definition and by Lemma 7.4.11. We get

$$\begin{aligned}
&| \min_{\rho:\mathcal{U}\to C} \operatorname{ecost}_{D,w'}(\mathcal{U}, C, \rho) - \min_{\rho:\mathcal{U}\to C} \operatorname{ecost}_{D,w'}(\mathcal{U}, C_G, \rho)| \\
\leq &\frac{\varepsilon}{390\beta} \Big(RW + 2 \cdot 17 \cdot \operatorname{ecost}_{D,w}(\mathcal{V}, A, \sigma_{\mathcal{V}}) + 2 \cdot (6\beta + 6) \operatorname{ecost}^*_{D,w}(\mathcal{V}) \Big) \\
\leq &\frac{\varepsilon}{390\beta} \Big(\operatorname{ecost}^*_{D,w}(\mathcal{V}) + 34(3\beta + 2) \operatorname{ecost}^*_{D,w}(\mathcal{V}) + (12\beta + 12) \operatorname{ecost}^*_{D,w}(\mathcal{V}) \Big) \\
\leq &\frac{\varepsilon}{390\beta} (114\beta + 81) \operatorname{ecost}^*_{D,w}(\mathcal{V}) \leq (\varepsilon/2) \cdot \operatorname{ecost}^*_{D,w}(\mathcal{V}).
\end{aligned}$$

by Lemma 7.4.11 again. That completes the proof. □

The main theorem

We conclude with collecting our results and proving the main theorem.

Theorem 7.4.21. *Let $X \subset \mathbb{R}^d$ be a finite set and let $\mathcal{V}$ be a set of n nodes $v_i : X \to [0,1]$, $i \in [n]$, let k be a positive integer and let $w : \mathcal{V} \to \mathbb{R}^+$ be a weight function, and let $W := \sum_{i=1}^{n} w(v_i)$ be the total weight. Let $0 < \delta, \varepsilon < 1$ be given constants.*

A $(1+\varepsilon)$-coreset $\mathcal{U}$ for the assigned Euclidean k-median problem consisting of nodes $u_i : X \to [0,1]$ can be computed in time $\mathcal{O}(nd \cdot (k+m) \cdot (d \log(W/(\delta\varepsilon)))$ with error probability δ. The coreset $\mathcal{U}$ consists of $\mathcal{O}(\epsilon^{-2} k^2 d \cdot \log^4(W/(\varepsilon\delta)))$ nodes. In additional running time $\mathcal{O}(nmd\varepsilon^{-1}(\log(nm/\delta) \cdot (\log(p_{\min}^{-1}))))$, the supports of all nodes can be reduced such that each support contains at most $\mathcal{O}(\varepsilon^{-3} d \log^2(m/(p_{\min}\delta\varepsilon)))$ points from X.

Proof. The coreset size is a direct result of the description of the algorithm. This description and the discussion of the running times can be found from page 176 to page 179. Let $C \subset \mathbb{R}^d$ be a set of centers. If there is at least one center outside of $\mathfrak{U}$, then Lemma 7.4.19 guarantees that

$$\left| \min_{\rho:\mathcal{V}\to C} \operatorname{ecost}_{D,w}(\mathcal{V}, C, \rho) - \min_{\rho:\mathcal{U}\to C} \operatorname{ecost}_{D,w'}(\mathcal{U}, C, \rho) \right| \leq (\varepsilon/2) \cdot \min_{\rho:\mathcal{V}\to C} \operatorname{ecost}_{D,w}(\mathcal{V}, C, \rho)$$

holds. Otherwise, all centers in C lie within $\mathfrak{U}$. Now C either lies on grid points and we define $C_G := C$, or Lemma 7.4.20 gives a center set C_G with

$$| \min_{\rho:\mathcal{V}\to C} \operatorname{ecost}_{D,w}(\mathcal{V}, C, \rho) - \min_{\rho:\mathcal{V}\to C} \operatorname{ecost}_{D,w}(\mathcal{V}, C_G, \rho)| \leq (\varepsilon/10) \cdot \min_{\rho:\mathcal{V}\to C} \operatorname{ecost}_{D,w}(\mathcal{V}, C, \rho)$$

and

$$|\min_{\rho:\mathcal{U}\to C} \operatorname{ecost}_{D,w'}(\mathcal{U},C,\rho) - \min_{\rho:\mathcal{U}\to C} \operatorname{ecost}_{D,w'}(\mathcal{U},C_G,\rho)| \leq (\varepsilon/2)\cdot \min_{\rho:\mathcal{V}\to C} \operatorname{ecost}_{D,w}(\mathcal{V},C,\rho).$$

Lemma 7.4.17 then ensures that C_G satisfies that

$$\begin{aligned}&\left|\min_{\rho:V\to C_G} \operatorname{ecost}_{D,w}(\mathcal{V},C_G,\rho) - \min_{\rho:\mathcal{U}\to C_G} \operatorname{ecost}_{D,w'}(\mathcal{U},C_G,\rho)\right| \\ \leq (\varepsilon/5)\cdot &\min_{\rho:\mathcal{V}\to C_G} \operatorname{ecost}_{D,w}(\mathcal{V},C_G,\rho) \leq \ (1+\varepsilon)\cdot(\varepsilon/5)\cdot \min_{\rho:\mathcal{V}\to C} \operatorname{ecost}_{D,w}(\mathcal{V},C,\rho).\end{aligned}$$

Together, this implies that

$$\begin{aligned}&\left|\min_{\rho:V\to C} \operatorname{ecost}_{D,w}(\mathcal{V},C,\rho) - \min_{\rho:\mathcal{U}\to C} \operatorname{ecost}_{D,w'}(\mathcal{U},C,\rho)\right| \\ \leq &\left|\min_{\rho:V\to C} \operatorname{ecost}_{D,w}(\mathcal{V},C,\rho) - \min_{\rho:\mathcal{V}\to C_G} \operatorname{ecost}_{D,w}(\mathcal{V},C_G,\rho)\right| \\ &+\left|\min_{\rho:\mathcal{V}\to C_G} \operatorname{ecost}_{D,w}(\mathcal{V},C_G,\rho) - \min_{\rho:\mathcal{U}\to C_G} \operatorname{ecost}_{D,w'}(\mathcal{U},C_G,\rho)\right| \\ &+\left|\min_{\rho:\mathcal{U}\to C_g} \operatorname{ecost}_{D,w'}(\mathcal{U},C_G,\rho) - \min_{\rho:\mathcal{U}\to C} \operatorname{ecost}_{D,w'}(\mathcal{U},C,\rho)\right| \\ \leq &(1/10+2/5+1/2)\varepsilon\cdot \operatorname{ecost}_{D,w}(\mathcal{V},C,\rho) = \varepsilon\cdot \operatorname{ecost}_{D,w}(\mathcal{V},C,\rho).\end{aligned}$$

Performing Step (3) increases the error, but the result is still a $(1+3\varepsilon)$-coreset. Thus, running all steps with precision parameter $\varepsilon/3$ yields a $(1+\varepsilon)$-coreset for this case as well. □

8 Projective clustering problems

A *shape fitting problem* is the task to determine which of a set of predefined shapes fits best for a given input point set. There is a huge variety of different problems that fall into this category, since there are many ways to choose the candidate shapes and also to define how well a shape fits to a set of points. For example, we can fit lines or circles to a point set, or hyperplanes, subspaces of another fixed dimension, or just points. The fit of a shape can be the sum of the distances of all points to the shape, or the maximum distance between any point and the shape. And then, the distance can also be defined in different ways, for example as the Euclidean distance or as the squared Euclidean distance, or as any qth power of a distance based on a p-norm.

Notice that the k-means and the k-median problem are special cases of shape fitting problems where the shapes are sets of k points and the measure to evaluate a shape is the sum of the (squared) distances. So, we have been studying shape fitting problems in this thesis at several places. In this chapter, we discuss two other specific shape fitting problems. They are both based on the the sum of the squared distances as a measure of the fit of shapes (which are subsets of $\mathbb{R}^d$).

The first problem is the subspace approximation problem, where we want to find a subspace of $\mathbb{R}^d$ of dimension j that minimizes the sum of the squared distances to an input point set. We encountered this problem in Section 3.1.2 in the context of the singular value decomposition. In fact, we already know that the best fit subspace of dimension m, named V_m, can be computed by the singular value decomposition. It is the optimal solution to the linear subspace approximation problem. We see how to compute coresets for the linear and affine subspace approximation problem in Section 8.2.

The second problem that we consider is the integer projective clustering problem where each shape is a set of k subspaces that are j-dimensional. The study of inputs with integer coordinates was initiated by Edwards and Varadarajan [EV05] as a reaction to a result by Har-Peled [HP04]. The latter showed that there are no coresets of sublinear size for the problem to fit two strips to a point set in $\mathbb{R}^2$. Edwards and Varadarajan showed how to compute coresets for the problem to fit a set of k hyperplanes to a point set under the assumption that the input points have integer coordinates. Both results are for the version where the fit is measured by the maximum distance of any input point to the shape. We consider the integer projective clustering problem with the sum of the squared Euclidean distances as the measure in Section 8.4. This variant has also been studied before as we will see below in the related work section.

The results presented in this chapter are joint work with Dan Feldman and Christian Sohler and have partially appeared in [FSS13].

8.1 Introduction to projective clustering problems

Since we build upon the results from Chapter 3, we use the matrix notation for input points, i. e., the input consists of a matrix $A \in \mathbb{R}^{n \times d}$ which represents n points of dimension d. The ith row of A is referred to as A_{i*}.

Recall that a (linear) subspace of $\mathbb{R}^d$ is a subset $V \subseteq \mathbb{R}^d$ which is the span of a finite set of vectors from $\mathbb{R}^d$. If the subspace is j-dimensional, it is spanned by j linearly independent vectors $a_1, \ldots, a_j$, and such a set of spanning vectors is called a basis of V. Notice that linear subspaces always contain the origin. An j-dimensional *affine subspace* for $j < d$ is defined by a linear j-dimensional subspace V spanned by a basis $\{a_1, \ldots, a_j\}$ plus a vector t that is orthogonal to all a_ℓ for $\ell \in [j]$. The affine subspace then is $T := T_{V,t} := \{x + t \mid x \in V\}$. If t is nonzero, then T does not contain the origin, and the length of t is the distance of $T_{V,t}$ to the origin. Projective clustering can be defined for both linear and affine subspaces.

Definition 8.1.1. *Let $A \in \mathbb{R}^{n \times d}$ be a matrix with n points, let $k \geq 1$ and $0 \leq j \leq d-1$ be given integer parameters. The linear (affine) projective clustering problem with squared Euclidean distances is the task to choose a set $\mathcal{V}$ of k linear (affine) subspaces of $\mathbb{R}^d$ of dimension j that minimizes*

$$\sum_{i=1}^{n} \min_{V \in \mathcal{V}} \operatorname{dist}^2(A_{i*}, V).$$

The subspace approximation problem is the special case where $k = 1$. The integer projective clustering problem has the additional assumption that the input points have integer coordinates.

We omit the addition 'with squared Euclidean distances' from now on since squared Euclidean distances are the focus of this chapter. Notice that the affine projective clustering problem for $j = 0$ is the k-means problem. For $j = 1$, the problem is called the k-line problem. The special case of $k = 1$ and $j \in \{0, d-1\}$, which is the subspace approximation problem, is also known as the low-rank approximation problem if we are interested in the projection of the points to the best fit subspace (not in the subspace itself). The optimal solution to this problem is the matrix $A^{(j)}$, which consists of the projections of the points in A to the best fit subspace V_j.

Related work on projective clustering problems. Since projective clustering comprises a lot of interesting problems as special cases, the amount of research on projective clustering problems is vast. We have a short look at some results. Earlier work often studies projective clustering problems with the maximum distance as the measure of fit. That means that the maximum distance between any input point and the shape shall be minimized. Usually, the shape consists of k linear or affine subspaces. The problem can then also be interpreted as covering the input point set with the geometric forms that arise from considering all points that are of a specific distance to a subspace.

Zero dimensional subspaces are only meaningful when they are affine. In the easiest case, we have one affine subspace of dimension zero, and then we get the *minimum enclosing*

ball problem. It is the task to compute a sphere of minimum radius that covers a point set. It is a well-studied problem that is solvable in polynomial time. For a comprehensive list of references on the minimum enclosing ball problem, see for example [Yıl08]. Its generalization is the k-center problem where a point set is supposed to be covered by k spheres, and the maximum radius is to be minimized. In terms of projective clustering, it is the special case of arbitrary k and $j = 1$. We shortly discuss the k-center problem and some references on page 159.

For $j = 1$, i. e., if the subspace is of dimension one, the task is to fit a set of lines to the input point set. These can be linear or affine subspaces. Following the term k-center, the problem is also called the k-line center problem. Since the set of points that is within a given distance of a one-dimensional subspace is an (infinite) cylinder, it is also sometimes referred to as a covering problem with cylinders. Megiddo and Tamir show that it is NP-complete to decide whether a set of n points in the plane can be covered by k affine subspaces of dimension $j = 1$. This implies that multiplicative approximation is NP-hard since any multiplicative approximation would have to decide whether optimal fit actually covers the points. This holds when the fit is measured by the maximum distance as well as when the fit is measured by the sum of the distances or squared distances. In both cases, the cost is zero if there are k lines such that each input point lies on one of them. For constant k, d and ε, Agarwal, Procopiuc and Varadarajan [APV05] give an algorithm with running time $\mathcal{O}(n \log n)$ that computes a set of k lines such that the maximum distance is at most $(1 + \varepsilon)$ times the optimal maximum distance. Har-Peled [HP04] shows that coresets do not exist for the k-line center problem even if the dimension d and the number of lines k are both two[1].

The problem of $j > 1$ and $k = 1$ is for example considered in [AHPV04, HPV04, SV12, YZ03]. In particular, Agarwal, Har-Peled and Varadarajan [AHPV04] show how to compute coresets for this case. Edwards and Varadarajan [EV05] build upon this result and develop a coreset construction for $j = d - 1$ and $k > 1$ under the assumption that the input points have integer coordinates (this is necessary for the existence of coresets due to the result by Har-Peled [HP04]). The projective clustering problem for $j > 1$ and $k > 1$ is also considered in [AP03, HPV02].

When the fit is measured by the sum of the distances or the sum of the squared distances, then the affine case $j = 0$ is the k-median problem or k-means problem, respectively. We discuss related work on these problems in Section 2.1 and in Section 2.2. Both problems are NP-hard. Feldman, Fiat and Sharir [FFS06] denote the case of $j = 1$ as the k-line median or k-line mean problem for Euclidean and squared Euclidean distances, respectively. They develop a $(1 + \varepsilon)$-approximation algorithm for these problems.

For squared Euclidean distances, the case $k = 1$ is special since it can be solved exactly by the singular value decomposition. It is also known as low-rank approximation or the subspace approximation problem. The research focus then lies on developing faster

[1] Notice that there are two types of coresets when the maximum distance is used for shape fitting, named additive and multiplicative coresets by Har-Peled. Additive coresets exist and are used by Agarwal et al., but multiplicative coresets do not exist.

algorithm. Work on computing one j-dimensional subspace that minimizes the sum of the distances, the sum of the squared distances or the sum of the qth power of the distances in a p-norm includes [DRVW06, DTV11, FFS06, FMSW10, SV12].

Research that considers the general case of $j > 1$ and $k > 1$ includes [DRVW06, DV07, FL11a, VX12a, VX12b]. The paper by Feldman and Langberg [FL11a] proposes a general framework based on sampling according to sensitivity which was developed by Langberg and Schulman [LS10]. Varadarajan and Xiao improve their results for the projective clustering problem in [VX12a, VX12b]. We discuss both the technique from [FL11a] and the method from [VX12a] in more detail in the following sections.

8.2 Small coresets for subspace approximation

In this section we study the linear subspace approximation problem. It is the special case of the linear projective clustering problem where k is set to one, and it is the task to compute the linear subspace of dimension j that minimizes the sum of the squared distances to a point set. The optimal solution is the best fit subspace V_j. Here, we are interested in computing a coreset for this problem. That means that we want to be able to approximate the sum of the squared distances for all possible j-dimensional subspaces, not only for V_j. We compute a coreset by using the following result which we proved in Section 3.2.

Corollary 3.2.2. *Let $A \in \mathbb{R}^{n\times d}$ and let V be a j-dimensional subspace. Let $\varepsilon \in (0,1)$ and $m \in \mathbb{N}$ with $m \geq \lceil j/\varepsilon \rceil$ and $n, d \geq m + j$. Let $A^{(m)} \in \mathbb{R}^{n\times m}$ be the projection of A to V_m, the best fit subspace of dimension m. Then it holds that*

$$\operatorname{dist}^2(A,V) \leq \operatorname{dist}^2(A^{(m)},V) + \sum_{i=m+1}^{\min\{n,d\}} \sigma_i^2 \leq \operatorname{dist}^2(A,V) + \varepsilon \cdot \sum_{i=j+1}^{m+j} \sigma_i^2 \leq (1+\varepsilon)\operatorname{dist}^2(A,V).$$

Corollary 3.2.2 holds for any j-dimensional subspace, so $A^{(m)}$ and the constant form a $(1+\varepsilon)$-coreset for the subspace approximation problem. However, $A^{(m)}$ still contains n points, and even though they are projected to an m-dimensional subspace, the matrix $A^{(m)}$ still stores them with d coordinates. Instead, we could store the right singular vectors that span V_m and the representation of the points in this basis. That would require m points of dimension d and n points of dimension m. Alternatively, we could also apply a coreset construction for the subspace approximation problem to $A^{(m)}$ as we did for the k-means problem in Section 4.4. However, there is an easier way to obtain a small coreset.

The sum of the squared distances of all points in $A^{(m)}$ to V satisfies $\operatorname{dist}^2(A^{(m)},V) = ||A^{(m)}||^2 - ||\pi_V(A^{(m)})||^2$ by the Pythagorean theorem (see Observation 3.1.3) where $\pi_V(A^{(m)})$ is the matrix that arises from projecting all points in $A^{(m)}$ to V. Thus, a point set is a coreset for the subspace approximation problem if the sum of the lengths of the projected points is similar for $A^{(m)}$ and A for any subspace V.

Recall from Section 3.1.2 that we can represent $A^{(m)}$ as $A^{(m)} = \sum_{i=1}^{m} \sigma_i u_i v_i^T$ where u_i and v_i are left and right singular vectors to the singular value σ_i. Additionally, $u_1, \ldots, u_n \in \mathbb{R}^n$

and $v_1, \ldots, v_d \in \mathbb{R}^d$ are orthonormal bases of $\mathbb{R}^n$ and $\mathbb{R}^d$, respectively. Let V be any j-dimensional subspace and assume that it is spanned by the orthonormal basis $\{a_1, \ldots, a_j\}$. Then there are unique coefficients $\alpha_{\ell,s}$ for $\ell \in [j]$ and $s \in d$ such that $a_\ell = \sum_{s=1}^{d} \alpha_{\ell,s} v_s$. It holds for any $\ell \in [j]$ that

$$||A^{(m)} a_\ell||^2 = \Big|\Big| \sum_{i=1}^{m} \sigma_i u_i v_i^T \sum_{j=1}^{d} \alpha_{\ell j} v_j \Big|\Big|^2 = \Big|\Big| \sum_{i=1}^{m} \sum_{j=1}^{d} \sigma_i \alpha_{\ell j} u_i v_i^T v_j \Big|\Big|^2 = \Big|\Big| \sum_{i=1}^{m} \sigma_i \alpha_{\ell i} u_i \Big|\Big|^2 = \sum_{i=1}^{m} \sigma_i^2 \alpha_{\ell i}^2.$$

We define the coreset to be the matrix $S^{(m)} \in \{m \times d\}$ where the ith row is the point $\sigma_i v_i$. Then we have that

$$||S^{(m)} a_\ell||^2 = \Big|\Big| S^{(m)} \sum_{j=1}^{d} \alpha_{\ell j} v_j \Big|\Big|^2 = \Big|\Big| \sum_{i=1}^{m} \sigma_i v_i \sum_{j=1}^{d} \alpha_{\ell j} v_j \Big|\Big|^2 = \sum_{i=1}^{m} \sigma_i^2 \alpha_{\ell i}^2.$$

Thus, $S^{(m)}$ does indeed form a coreset together with the constant. Notice that $S^{(m)}$ contains only $m \in \mathcal{O}(j/\varepsilon)$ points.

Corollary 8.2.1. *Let $A \in \mathbb{R}^{n \times d}$. Let $\varepsilon \in (0,1)$ and $m \in \mathbb{N}$ with $m \geq \lceil j/\varepsilon \rceil$ and $n, d \geq m + j$. It holds for every j-dimensional subspace V that*

$$\operatorname{dist}^2(A, V) \leq \operatorname{dist}^2(S^{(m)}, V) + \sum_{i=m+1}^{d} \sigma_i^2 \leq (1+\varepsilon) \operatorname{dist}^2(A, V)$$

where the matrix $S^{(m)} \in \mathbb{R}^{m \times d}$ is defined by setting the ith row to $\sigma_i v_i$ for $i = 1, \ldots, m$.

8.2.1 Affine subspaces

The affine subspace approximation problem consists of finding a linear subspace V and a translation t such that the sum of the squared distances of all points in A to their closest point in $V + t$ is minimized. Figure 8.1 illustrates that the affine best fit subspace can have a better fit to a point set than the best linear subspace.

We start with a lemma that relates the sum of the squared distances to an affine subspace to the sum of the squared distances to a linear subspace plus additional terms. We need some notation. Let $\pi_V : \mathbb{R}^d \to V$ be the mapping that projects every point x to its projection in V, so $\operatorname{dist}^2(x, V) = \operatorname{dist}^2(x, \pi_V(x))$. For any subspace, we denote the mapping that projects every point x to its closest point in T by $\pi_T(x)$. Notice that $\pi_T(x) = \pi_V(x - t) + t = \sum_{\ell=1}^{j} (x - t)^T a_\ell = \sum_{\ell=1}^{j} x^T a_\ell = \pi_V(x) + t$ since t is orthogonal to all a_ℓ.

Lemma 8.2.2. *Let $A \in \mathbb{R}^{n \times d}$ contain a set of n points from $\mathbb{R}^d$. It holds for any j-dimensional affine subspace $T = T_{V,t}$ that*

$$\sum_{i=1}^{n} \operatorname{dist}^2(A_{i*}, T_{V,t}) = \operatorname{dist}^2(A, V) + n \cdot (||\mu - \pi_T(\mu)||^2 - ||\mu - \pi_V(\mu)||^2)$$

where $\mu = \frac{1}{n} \sum_{i=1}^{n} A_{i*}$. If μ is the null vector, then the distance to $T_{V,t}$ is given by

$$\sum_{i=1}^{n} \operatorname{dist}^2(A_{i*}, T_{V,t}) = \operatorname{dist}^2(A, V) + n \cdot ||t||^2.$$

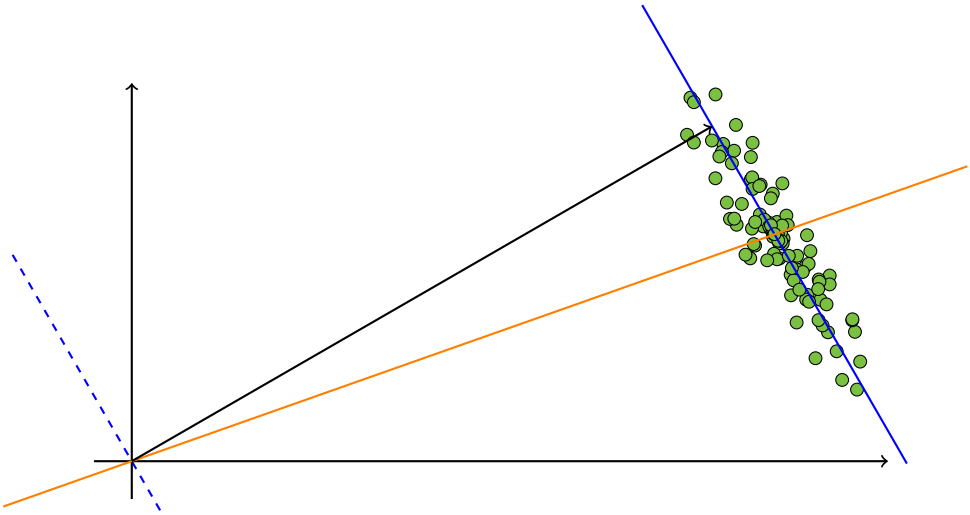


Figure 8.1: An example of a point set, the linear best fit subspace (orange) and the affine best fit subspace (blue). The affine subspace is given by the translation depicted by the black arrow and the linear subspace indicated by the dashed line.

Proof. Translating $T_{V,t}$ and A by the same vector does not change the distances between the points in A and their closest point in $T_{V,t}$. For the translation $-t$ we get $\sum_{i=1}^{n} \text{dist}^2(A_{i*}, T_{V,t})) = \sum_{i=1}^{n} \text{dist}^2(A_{i*} - t, V)$ since $\{x - t \mid x \in T_{V,t}\} = V$. By the Pythagorean theorem (see Observation 3.1.3), we get that

$$\sum_{i=1}^{n} \text{dist}^2(A_{i*} - t, \pi_V(A_{i*} - t)) = \sum_{i=1}^{n} ||A_{i*} - t||^2 - \sum_{i=1}^{n} ||\pi_V(A_{i*} - t)||^2$$

for all $i \in [n]$. We use Observation 2.4.1 that allows to calculate the sum of the squared distances by using the centroid of a point set. We interpret t as the center and get

$$\sum_{i=1}^{n} ||A_{i*} - t||^2 = \Big(\sum_{i=1}^{n} ||A_{i*} - \mu||^2\Big) + n \cdot ||\mu - t||^2.$$

Since $t \in T_{V,t}$ we get by the Pythagorean theorem that

$$\begin{aligned} &||\mu - t||^2 = ||\pi_T(\mu) - t||^2 + \text{dist}^2(\mu, \pi_T(\mu)) \\ =&||\pi_V(\mu) + t - t||^2 + \text{dist}^2(\mu, \pi_T(\mu)) = ||\pi_V(\mu)||^2 + \text{dist}^2(\mu, \pi_T(\mu)). \end{aligned}$$

For the subtrahend, recall that $\pi_V(x) = \sum_{\ell=1}^{j} x^T a_\ell a_\ell$ and that t is orthogonal to all a_ℓ. We thus get that

$$\begin{aligned} ||\pi_V(A_{i*} - t)||^2 = \Big|\Big| \sum_{\ell=1}^{j} (A_{i*} - t)^T a_\ell a_\ell \Big|\Big|^2 = \Big|\Big| \sum_{\ell=1}^{j} (A_{i*}^T a_\ell - t^T a_\ell) a_\ell \Big|\Big|^2 =& \Big|\Big| \sum_{\ell=1}^{j} A_{i*}^T a_\ell a_\ell \Big|\Big|^2 \\ =&||\pi_V(A_{i*})||^2 \end{aligned}$$

for all $i \in [n]$. We conclude that

$$\begin{aligned}\sum_{i=1}^{n} \operatorname{dist}^2(A_{i*}, T_{V,t}) &= \sum_{i=1}^{n} ||A_{i*} - t||^2 - \sum_{i=1}^{n} ||\pi_V(A_{i*} - t)||^2 \\ &= \Big(\sum_{i=1}^{n} ||A_{i*} - \mu||^2 - ||\pi_V(A_{i*})||^2\Big) + n \cdot (||\pi_V(\mu)||^2 + ||\mu - \pi_T(\mu)||^2).\end{aligned}$$

We can rewrite $\operatorname{dist}^2(A, V)$ in a similar way to obtain

$$\operatorname{dist}^2(A, V) = \sum_{i=1}^{n} ||A_{i*}||^2 - ||\pi_V(A_{i*})||^2 = \sum_{i=1}^{n} \Big(||A_{i*} - \mu||^2 - ||\pi_V(A_{i*})||^2\Big) + n||\mu||^2.$$

The first statement then follows because $||\mu||^2 = ||\pi_V(\mu)||^2 + ||\mu - \pi_V(\mu)||^2$. The second statement follows since $||\mu - \pi_T(\mu)||^2 - ||\mu - \pi_V(\mu)||^2 = ||t||^2$ if μ is the null vector. □

By the reformulation of $\operatorname{dist}^2(A, T_{V,t})$ we also see that the optimal translation of a given subspace V is $t = \mu - \pi_V(\mu)$ since $||\mu - \pi_T(\mu)||^2 = ||\mu - \pi_V(\mu) - t||^2$ is then zero and it is the only term that depends on t. If $\mu = 0$, then the optimal affine subspace of dimension j is the linear best fit subspace. In order to compute the solution to the affine subspace approximation problem if the centroid is not zero, we can translate the input matrix by $-\mu$, compute the best fit subspace and translate it back. Computing a coreset for the affine subspace approximation works based on the same idea. We see this in more detail in the following corollary.

Corollary 8.2.3. *Let $A \in \mathbb{R}^{n \times d}$. Let $\varepsilon \in (0, 1)$ and $m \in \mathbb{N}$ with $m \geq \lceil j/\varepsilon \rceil$ and $n, d \geq m + j$. There exist a matrix $S \in \mathbb{R}^{2m \times d}$ and a constant $\Delta \in \mathbb{R}^+$ such that*

$$\operatorname{dist}^2(A, T_{V,t}) \leq \operatorname{dist}^2(S, T_{V,t}) + \Delta \leq (1 + \varepsilon) \operatorname{dist}^2(A, T_{V,t})$$

holds for every j-dimensional affine subspace $T_{V,t}$ consisting of a j-dimensional subspace V and an (orthogonal) translation vector $t \in \mathbb{R}^d$.

Proof. Let A' be the matrix obtained by translating A by $-\mu$ where $\mu := (1/n) \sum_{i=1}^{n} A_{i*}$ is the centroid of the input point set. Let $v_1, \ldots, v_d$ be right singular vectors with corresponding singular values for A'. By Corollary 8.2.1, the matrix $S^{(m)} \in \mathbb{R}^{m \times d}$ which has $\sigma_i v_i$ as its ith row for $i \in [m]$ satisfies

$$\operatorname{dist}^2(A', V) \leq \operatorname{dist}^2(S^{(m)}, V) + \sum_{i=m+1}^{d} \sigma_i^2 \leq (1 + \varepsilon) \operatorname{dist}^2(A', V) \tag{8.1}$$

for all j-dimensional subspaces V. We use the matrix S' which contains $(\sigma_i/\sqrt{2})v_i$ and $-(\sigma_i/\sqrt{2})v_i$ as rows $2i - 1$ and $2i$ for each $i \in [m]$, so S' has $2m$ rows. Notice that the centroid of S' is the origin since each pair of rows cancels itself out when summing them up. S' has the same distance to any linear subspace as $S^{(m)}$ since the projection length of a row $(\sigma_i/\sqrt{2})v_i$ to a unit vector a_ℓ is $||(\sigma_i/\sqrt{2})v_i^T a_\ell||^2 = ||\sigma_i v_i^T a_\ell||^2/2$, and the projection length

of $-(\sigma_i/\sqrt{2})v_i$ is $||\sigma_i v_i^T a_\ell||^2/2$ as well. Thus, $||\pi_V(S_i^{(m)})||^2 = ||\pi_V(S'_{2i-1})||^2 + ||\pi_V(S'_{2i})||^2$, and thus

$$\begin{aligned}\operatorname{dist}^2(S', V) &= \sum_{i=1}^{2m} ||S'_{i*}||^2 - ||\pi_V(S'_{i*})||^2 = \sum_{i=1}^{2m} \sigma^2/2 - \sum_{i=1}^{m}(||\pi_V(S'_{2i-1})||^2 + ||\pi_V(S'_{2i})||^2) \\ &= \sum_{i=1}^{m}(||S_i^{(m)}||^2 - ||\pi_V(S_i^{(m)})||^2) = \operatorname{dist}^2(S^{(m)}, V).\end{aligned}$$

Together with (8.1), this implies that

$$\operatorname{dist}^2(A', V) \leq \operatorname{dist}^2(S', V) + \sum_{i=m+1}^{d} \sigma_i^2 \leq (1+\varepsilon)\operatorname{dist}^2(A', V)$$

holds for all j-dimensional linear subspaces. Let $T_{V,t}$ be an j-dimensional affine subspace and consider the translated subspace $T_{V,t'} = \{x - \mu \mid x \in T_{V,t}\}$. Since the centroid of A' and S' is the origin, Lemma 8.2.2 implies that

$$\operatorname{dist}^2(A', T_{V,t'}) = \operatorname{dist}(A', V) + n \cdot ||t'||^2$$

and similarly $\operatorname{dist}^2(S', T_{V,t'}) = \operatorname{dist}^2(S', V) + n \cdot ||t'||^2$. This implies

$$\operatorname{dist}^2(A', T_{V,t'}) \leq \operatorname{dist}^2(S', T_{V,t'}) + \sum_{i=m+1}^{d} \sigma_i^2 \leq (1+\varepsilon)\operatorname{dist}^2(A', T_{V,t'}).$$

The matrix S now arises from adding μ to each row in S'. Then translating A', S' and $T_{V,t'}$ by μ gives A, S and $T_{V,t}$, and the distances between them are not changed. We thus get that

$$\operatorname{dist}^2(A, T_{V,t}) \leq \operatorname{dist}^2(S, T_{V,t}) + \sum_{i=m+1}^{d} \sigma_i^2 \leq (1+\varepsilon)\operatorname{dist}^2(A, T_{V,t}).$$

Thus, the matrix $S \in \mathbb{R}^{2m \times d}$ and the constant $\Delta := \sum_{i=m+1}^{d} \sigma_i^2$ form a $(1+\varepsilon)$-coreset for the affine subspace approximation problem. □

8.3 A coreset framework for projective clustering

Before turning to the integer projective clustering problem, we have a look at a framework that allows to compute coresets for different variants of projective clustering problems. We use it in the next section. The framework is based on a non-uniform sampling technique.

Sampling is a powerful tool to construct coresets. We have seen when discussing the work by Chen [Che09] and applying it to probabilistic k-median clustering in Section 7.4.2 that it has the potential to construct coresets which have a size that is polynomial in the dimension d, at least for the k-means and the k-median problems. These are special projective clustering problems.

Figure 8.2: A point set with an outlier as an input to the 2-means problem.

Sampling is about finding the important points in an input point set with high probability. Figure 8.2 sketches a point set where most points are concentrated around two centers and one point, named z in the example, is far away. Assume that we consider the k-means problem for $k = 2$. If the distance between z and the rest of the points is large enough, then z has to be part of any coreset that wants to approximate the cost function. This pointed example shows why sampling needs to be applied with care. If we sample uniformly at random and want to sample z with constant probability, we need a linear number of samples. There are different ways to handle this problem.

Chen uses a bicriteria approximation to get approximately good centers and then partitions the point set according to the distance to the nearest center. Uniform sampling is then applied to each subset, and the points in a subset have a similar distance to their closest center. In the toy example in Figure 8.2, z would get its own subset (if it is sufficiently far away) and is then sampled with probability one.

A different way is to directly base the sampling probabilities on the distances to the centers from the bicriteria approximation. This idea is used by Arthur and Vassilvitskii [AV07] for computing an approximation for the k-means problem, and it is used for the construction of (weak) coresets by Feldman, Monemizadeh and Sohler [FMS07]. The latter construction uses a set of centers that provides an approximative solution and distinguishes between points that are close to a center and points that are further away from their closest center. Uniform sampling is used for the close points. For the other points, the probability is based on the cost of the points. Again, z is likely to be sampled since its share in the cost of the approximative solution should be high.

Langberg and Schulman [LS10] and Feldman and Langberg [FL11a] define the notion of *sensitivity* of points which is an even more direct way of measuring the importance of a point. Their work is not restricted to the k-means problem but works for a large class of shape fitting problems. We review their technique in the following.

In this context, inputs to a shape fitting problem are viewed as functions. Assume we are given a matrix $A \in \mathbb{R}^{n \times d}$, and that the problem is to find the best shape in a set $\mathcal{Q}$ of closed sets from $\mathbb{R}^d$, i. e., the shape $Q \in \mathcal{Q}$ that minimizes $\text{dist}^2(A, Q)$, the sum of the squared distances between the points in A and the shape. Then we can define a function $f_{A_{i*}} : \mathcal{Q} \to \mathbb{R}^+$ that evaluates the fit of a shape to the point by mapping each Q to $f_{A_{i*}}(Q) := \text{dist}^2(A_{i*}, Q)$. Thus, the matrix A induces a finite set of functions from $\mathcal{Q}$ to $\mathbb{R}^+$. A coreset has to approximate the sum of these function values for all possible shapes.

The sensitivity is now defined as the maximum share that one of the functions can contribute to the sum of the function values for any given shape.

Definition 8.3.1 (Sensitivity, [LS10, FL11a]). *Let F be a finite set of functions from a set $\mathcal{Q}$ to $\mathbb{R}^+$. The sensitivity of a function $f \in F$ is*

$$\sigma(f) = \sup_{Q \in \mathcal{Q}} \frac{f(Q)}{\sum_{f' \in F} f'(Q)}.$$

The total sensitivity of F is $\mathfrak{S}(F) := \sum_{f \in F} \sigma(f)$.

When we speak of the sensitivity of a matrix A, we mean the sensitivity of the function set $F = \{f_{A_{i*}} \mid i = 1, \ldots, n\}$ for $f_{A_{i*}}$ as described above. Thus, we use the abbreviation $\sigma(A_{i*}) := \sigma(f_{A_{i*}})$, and we say that the total sensitivity of A is $\mathfrak{S}(A) := \sum_{i=1}^{n} \sigma(A_{i*})$.

Notice that sensitivity is an accurate measure of the importance of a point for the cost function. If a set of points has a low total sensitivity, then we can be sure that omitting it cannot distort the cost function much for any shape. In the example in Figure 8.2, placing two centers in the centroids of the two point clouds gives z a high share in the cost, so the sensitivity of z is high.

Feldman and Langberg connect projective clustering problems and coresets with the theory evolving around PAC Learning and the concept of the Vapnik-Chervonenkis dimension (VC dimension). They develop a sensitivity based sampling scheme that computes coresets and approximations for different shape fitting problems. We need two more definitions to state the coreset result.

Definition 8.3.2 (Dimension of a range space). *A* range space *is a pair* (F, ranges) *where F is a set, and* ranges *is a set of subsets of F. The* dimension *of the range space* (F, ranges) *is the smallest integer v, such that for every $G \subseteq F$ we have*

$$\Big|\{G \cap \text{range} \mid \text{range} \in \text{ranges}\}\Big| \leq |G|^v.$$

Definition 8.3.3 (Dimension of a set of functions. [LLS01, FL11a]). *Let F be a finite set of functions from a set Q to $\mathbb{R}^+$. For every $Q \in \mathcal{Q}$ and $r \geq 0$, let* $\text{range}(F, Q, r) = \{f \in F \mid f(Q) \leq r\}$. *Let* $\text{ranges}(F) = \{\text{range}(F, Q, r) \mid Q \in \mathcal{Q}, r \geq 0\}$. *The* dimension $\dim(F)$ *of F is the dimension of the range space* $\Big(F, \text{ranges}(F)\Big)$.

Notice that $\text{ranges}(F)$ contains sets of functions from F, so the number of elements in $\text{ranges}(F)$ is bounded by $2^{|F|}$. It will usually be smaller, though. If $\mathcal{Q}$ is the set of all sets of k centers from $\mathbb{R}^d$ and F consists of all functions $f_{A_{i*}} = \text{dist}^2(A_{i*}, Q)$ for $Q \in \mathcal{Q}$, then $\text{ranges}(F)$ contains an element for every possible way to partition the point set with k spheres. If $\mathcal{Q}$ consists of all sets of k j-dimensional subspaces, then $\text{range}(F, Q, r)$ contains all sets of functions that represent points of A that are within distance r of Q, and $\text{ranges}(F)$ contains an element for each partitioning that can be induced by choosing Q and r. We see a bound on $\dim(F)$ for this case in the next section.

We can now state the coreset result by Feldman and Langberg that is applicable to different projective clustering problems. The core of their algorithm is based on sampling points according to (an upper bound on) the sensitivity, and the following theorem states the size

of the computed coreset depending on the dimension of F and its sensitivity. Feldman and Langberg also provide bounds for both terms for different shape fitting problems and obtain coreset results, and the paper also includes approximation and streaming results.

Theorem 8.3.4 (Theorem 4.1 in [FL11a]). *Let F be a finite set of functions from a set $\mathcal{Q}$ to $\mathbb{R}^+$. There is a subset $S \subseteq F$ of size*

$$|S| = O(1) \cdot \dim(F) \cdot \left(\frac{\mathfrak{S}(F)}{\varepsilon}\right)^2,$$

with an appropriate weight $w_f > 0$ for each $f \in S$ such that such that for every $Q \in \mathcal{Q}$,

$$(1-\varepsilon)\sum_{f\in F} f(Q) \leq \sum_{f\in S} w_f f(Q) \leq (1+\varepsilon)\sum_{f\in F} f(Q).$$

8.4 The integer linear projective clustering problem

In this section, we use the framework by Feldman and Langberg to compute coresets for the integer linear projective clustering problem. We do so by combining the dimensionality reduction from Section 3.2 with the work by Varadarajan and Xiao [VX12a] on coresets for the integer linear projective clustering problem. Recall that we proved the following theorem.

Theorem 3.2.3. *Let $A \in \mathbb{R}^{n\times d}$ be a matrix. Let $j \geq 1$ be an integer, let $\varepsilon \in (0,1)$ and let $m \in \mathbb{N}$ with $m \geq \lceil 18j/\varepsilon^2 \rceil$ and $n, d \geq m + j$. Then for any non-empty closed set C which is contained in a j-dimensional subspace, we have*

$$\left|\left(\mathrm{cost}_{\ell_2^2}(A^{(m)}, C) + \sum_{i=m+1}^{d} \sigma_i^2\right) - \mathrm{cost}_{\ell_2^2}(A, C)\right| \leq \varepsilon \cdot \mathrm{cost}_{\ell_2^2}(A, C).$$

We can apply the theorem to the integer projective clustering problem: Every set of k affine subspaces of dimension j is contained in a $k(j+1)$-dimensional linear subspace. Thus, we need $m := \lceil 18k(j+1)/\varepsilon^2 \rceil$.

The problem with combining this dimensionality reduction with known algorithms for the integer projective clustering problem is that the lower dimensional representation of a point set does not necessarily have integer coordinates even if the original points have this property. We discuss the details of this difficulty before we consider the technique by Varadarajan and Xiao to obtain the coreset result.

Lemma 8.4.1 (Variation of Lemma 5.1 in [VX12a]). *Let $d \geq 2$. Let $A \in \{1, \ldots, \Delta\}^{n\times d}$ be a set of n points, let k be an integer and let $0 \leq j \leq d-1$ be an integer. Let $\mathcal{Q}_{jk}$ be the family of all sets of k affine subspaces of $\mathbb{R}^d$ of dimension j. At least one of the following options holds.*

- *It holds $\mathrm{cost}_{\ell_2^2}(A, \mathcal{T}) \geq \frac{1}{(d\Delta)^{f(j)}}$ for all $\mathcal{T} \in \mathcal{Q}_{jk}$ for a function f that depends on j.*

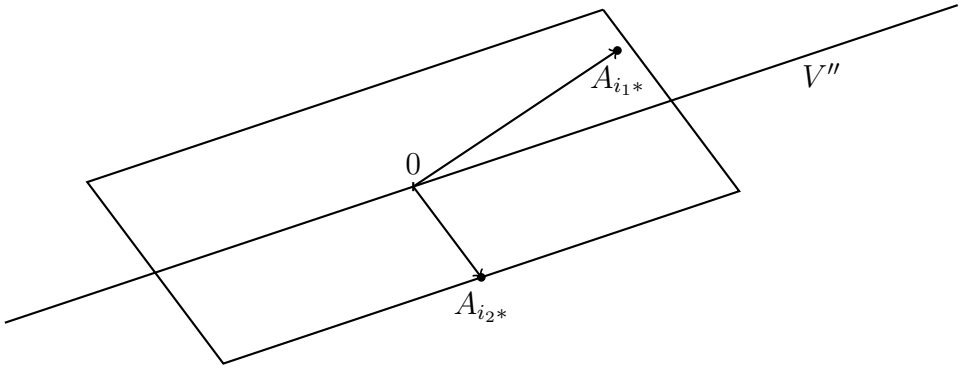


Figure 8.3: An example for $j = 1$. V' is the projection of C to the span of the two linearly independent input points A_{i_1*} and A_{i_2*}, and $V'' = V'$ in this example. The box is constructed by choosing a side length of two times the maximum length, which is $||A_{i_1*}||^2$, for the direction that is spanned by V'', and by chosing two times the maximum distance to V'' as the orthogonal side length.

- *The rank of A is bounded by $k(j+1)$.*

Proof. Let $\mathcal{T} \in \mathcal{Q}_{jk}$ be any set of k affine j-dimensional subspaces. Compute the partitioning of the points in A according to their closest subspace in $\mathcal{T}$. For each subset A' in this partitioning and the corresponding subspace T, consider whether the matrix that arises from moving A' by $\mu(A')$ has a rank of at most j. If it is, then A' lies within an affine j-dimensional subspace and within an $(j+1)$-dimensional linear subspace. If this is true for all k subsets, then A lies within a $k(j+1)$-dimensional subspace and the statement of the lemma is true.

Otherwise, consider a subset A' that does not lie within an affine j-dimensional subspace when translated by $\mu(A')$. Let $T \in \mathcal{T}$ be its closest affine subspace. We want that the subspace spanned by A' contains the origin. Translating A' by $\mu(A')$ might disturb the integrality of the coordinates, thus we translate A' by subtracting an arbitrary point in A' from A' and from T. This does not change the distances between points in A' and T. Name the translated versions A'' and T'. Notice that the coordinates of all points in A'' are integers between $-\Delta$ and 2Δ.

Next, we observe that by Lemma 8.2.2, we know that the cost of T' decreases if we translate it such that it contains the origin. So we move T' to a linear subspace V, where V is of dimension j. A lower bound on $\text{cost}_{\ell_2^2}(A'', V)$ implies a lower bound on $\text{cost}_{\ell_2^2}(A, \mathcal{T})$.

We know that A'' contains at least $j+1$ linearly independent points $A_{i_1*}, \ldots, A_{i_{j+1}*}$. V does not necessarily lie within the subspace spanned by this $j+1$ points, so we consider V', the projection of V to the span of the $A_{i_\ell*}$. Notice that the distance between any of the $j+1$ points and V' is not larger than the distance between V and the corresponding point. Thus, $\text{dist}(A_{i_\ell*}, V') \leq \sqrt{\text{cost}_{\ell_2^2}(A'', V)} \leq \sqrt{\text{cost}_{\ell_2^2}(A, \mathcal{T})}$ for all $\ell \in [j+1]$.

V' is at most j-dimensional. We extend it to a j-dimensional subspace V'' by adding arbitrary linearly independent vectors contained in the span of the $A_{i_\ell*}$. Picture a j-

dimensional box that lies in V'' that contains the projections of all $A_{i_\ell *}$ to V'' and the origin. A box like this can be found with sides of length $2 \cdot \max_{\ell=1,\dots,j+1} ||A_{i_\ell *}||$. It can be extended to a $(j+1)$-dimensional box containing the points themselves by assigning a side length of $2 \cdot \max_{\ell=1,\dots,j+1} \operatorname{dist}(A_{i_\ell *}, V')$ to the remaining orthogonal direction. An example for the construction of the box is depicted in Figure 8.3. The $(j+1)$-dimensional volume of this box is

$$2^{j+1} \cdot (\max_{\ell=1,\dots,j+1} ||A_{i_\ell *}||)^j \cdot \max_{\ell=1,\dots,j+1} \operatorname{dist}(A_{i_\ell *}, V') \leq 2^{j+1}(2 \cdot d \cdot \Delta)^j \cdot \sqrt{\operatorname{cost}_{\ell_2^2}(A, \mathcal{T})}.$$

The inequality follows because the squared ength of all points is bounded by $2d\Delta$ and because $\max_{\ell=1,\dots,j+1} \operatorname{dist}(A_{i_\ell *}, V') \leq \sqrt{\operatorname{cost}_{\ell_2^2}(A, \mathcal{T})}$.

We find a lower bound on this expression by observing that the described box contains all $j+1$ points and thus in particular the simplex that they span together with the origin. The $(j+1)$-volume of this simplex is $1/(j+1)!$ times the square root of $|\det(A'''A'''^T)|$, where A''' is a $(j+1) \times d$-matrix that contains $A_{i1}, \dots, A_{i(j+1)}$ as its rows (see 3.6.2 in [GK94]). A''' only contains integers, so its determinant is an integer, too. Thus, it is at least one, and this also holds for its square root. We conclude that the simplex has a $(j+1)$-dimensional volume of at least $1/((j+1)!)$, so it holds that

$$\begin{aligned} & 1/((j+1)!) \leq 2^{j+1}(2d \cdot \Delta)^j \cdot \sqrt{\operatorname{cost}_{\ell_2^2}(A, \mathcal{T})} \\ \Rightarrow \quad & \operatorname{cost}_{\ell_2^2}(A, V) \geq 1/(2^{j+1}(2d \cdot \Delta)^j (j+1)!)^2 \end{aligned}$$

which gives a lower bound on the cost of $1/(d\Delta)^{f(j)}$ for a function f that only depends on the parameter j. Again, the statement of the lemma is true. □

Lemma 8.4.1 says that we either have a lower bound on the optimal cost of A or A is contained in a lower dimensional subspace. We keep this in mind when going through the coreset construction proposed in [VX12a].

Varadarajan and Xiao develop a way to bound the sensitivity of a point set which we discuss in the next section. By the framework by Feldman and Langberg, this leads to a coreset result if $\dim(F)$ in Theorem 8.3.4 is suitably bounded. The following lemma by Feldman and Langberg settles this part. It is an extension of the corresponding result for the VC dimension.

Lemma 8.4.2 (Lemma 12.6, [FL11a], and personal communication with Dan Feldman)**.** *Let $\mathcal{Q}_{j,k}$ be the family of all sets consisting of k affine subspaces of dimension at most j. Define $F := \{f_{A_{i*}} \mid i = 1, \dots, n\}$ for with $f_{A_{i*}} : \mathcal{Q}_{j,k} \to \mathbb{R}^+$ defined by $f_{A_{i*}}(Q) = \operatorname{dist}^2(A_{i*}, Q)$. Then $\dim(F) \in \mathcal{O}(dk)$. If A is contained in an m-dimensional subspace for $m \geq j$, then it holds that $\dim(F) = \mathcal{O}(km)$.*

8.4.1 Sensitivity bounds from L_∞-coresets

The key idea by Varadarajan and Xiao is a way to bound the sensitivity of a point set A based on a so-called L_∞-coreset of A. An L_∞-coreset S is a coreset approximating the

maximum distance between the point set and any query shape. The name is due to the fact that the maximum distance is the infinity norm of the vector that consists of the distances between each point and its closest subspace. The next definition follows [EV05].

Definition 8.4.3 (L_∞-coresets)**.** *Let A in $\mathbb{R}^{n\times d}$, and $k > 0$ be an integer. Let $\mathcal{Q}$ be a family of closed and non-empty subsets of $\mathbb{R}^d$. A matrix $S \in \mathbb{R}^{r\times d} \subset A$ is an L_∞-coreset if it satisfies for all $\mathcal{U} \in \mathcal{Q}$ that*

$$\max_{i=1,\dots,n} \operatorname{dist}(A_{i*}, \mathcal{U}) \leq (1+\varepsilon) \cdot \max_{i=1,\dots,r} \operatorname{dist}(S_{i*}, \mathcal{U}).$$

If $\mathcal{Q} = \mathcal{Q}_{jk}$ is the family of all sets of k affine subspaces of dimension j, then we call the L_∞-coreset also L_∞-(j,k)-coreset for A.

In [VX12a] and [AHPV04], L_∞-coresets are instead defined to satisfy that

$$(1-\varepsilon) \max_{i=1,\dots,n} \operatorname{dist}(A_{i*}, \mathcal{U}) \leq \max_{i=1,\dots,r} \operatorname{dist}(S_{i*}, \mathcal{U}).$$

Notice that the existence of a L_∞-coreset in the sense of Definition 8.4.3 implies the existence of a L_∞-coreset with $\varepsilon' = \varepsilon/(1+\varepsilon)$ in the sense of [VX12a], and a similar statement holds for the reverse direction. In this sense, the definitions are equivalent.

Notice that for $\mathcal{Q} = \mathcal{Q}_{jk}$, the coreset property is

$$\max_{i=1,\dots,n} \min_{T\in\mathcal{T}} \operatorname{dist}(A_{i*}, T) \leq (1+\varepsilon) \cdot \max_{i=1,\dots,r} \min_{T\in\mathcal{T}} \operatorname{dist}(S_{i*}, T)$$

for all $\mathcal{T} \in \mathcal{Q}_{ij}$. The following lemma states the reduction by Varadarajan and Xiao.

Lemma 8.4.4 (Lemma 3.1 in [VX12a], applied to squared distances)**.** *Let an input matrix $A \in \mathbb{R}^{n\times d}$ and an integer $k > 0$ be given. Let $\mathcal{Q}$ be a family of closed and non-empty subsets of $\mathbb{R}^d$. Suppose that there exists a non-decreasing function $f_\varepsilon(n', d)$ such that, for all submatrices $A' \in \mathbb{R}^{n'\times d}$ of A, an L_∞-coreset $S \subseteq A'$ for $\mathcal{Q}$ of A' of size $f_\varepsilon(n', d)$ can be computed in time $T(n', d)$. Then it is possible to compute an upper bound $s(A_{i*})$ on the sensitivity $\sigma(A_{i*})$ for $i = 1, \dots, n$ such that*

$$\sum_{i=1}^{n} s(A_{i*}) \leq f_\varepsilon(n,d) \cdot (1+\varepsilon)^2 \cdot (\log n + 1)$$

The running time to compute $\sigma(A_{i*})$ for all points in A is bounded by $n \cdot T(n,d)$.

Proof. The proof uses a nested sequence of ℓ subsets of A for an $\ell \leq n$. $A_1 = A$ is the input set. The other sets are computed iteratively in the following way. If A_u contains at most $f_\varepsilon(n,d)$ points, the sequence ends and $\ell := u$. Otherwise, a coreset S_u of A_u is computed, and A_{u+1} is set to $A_u \backslash S_u$.

The result is a sequence of subsets $A_\ell \subseteq A_{\ell-1} \subseteq A_2 \subseteq A_1 = A$ with $|A_\ell| \leq f_\varepsilon(n,d)$ and a sequence of coresets $S_1, \dots, S_{\ell-1}$. Notice that together with A_ℓ, the coresets partition A. Thus, for $S_\ell := A_\ell$, the sets $S_1, \dots, S_\ell$ form a partitioning of A.

Now let A_{i*} be an input point, and let $v \in \{1, \ldots, \ell\}$ be the index of the coreset that A_{i*} is contained in, i. e., $A_{i*} \in S_v$. Let $\mathcal{U} \in \mathcal{Q}$. The goal is to upper bound the sensitivity of A_{i*} by lower bounding the contribution of the remaining points.

Consider the coresets S_u for $1 \leq u \leq v$ and notice that each of them is a coreset for a point set that contains A_{i*}, because $A_{i*} \in A_u$ for $1 \leq u \leq v$. For each $u \in \{1, \ldots, v\}$, let A_{i_u*} be the point in S_u with maximum distance to $\mathcal{U}$. By the L_∞-coreset property, that implies that

$$\begin{aligned} & \operatorname{dist}(A_{i*}, \mathcal{U}) \leq (1+\varepsilon) \cdot \operatorname{dist}(A_{i_u*}, \mathcal{U}) \\ \Rightarrow \quad & \operatorname{dist}^2(A_{i_u*}, \mathcal{U}) \geq \frac{1}{(1+\varepsilon)^2} \operatorname{dist}^2(A_{i*}, \mathcal{U}). \end{aligned}$$

The cost of $A_{i_1*}, \ldots, A_{i_v*}$ is a lower bound on the cost of all points. Thus, it follows that

$$\sum_{i=1}^{n} \operatorname{dist}^2(A_{i*}, \mathcal{U}) \geq \sum_{u=1}^{v} \operatorname{dist}^2(A_{i_u*}, \mathcal{U}) \geq \frac{v}{(1+\varepsilon)^2} \operatorname{dist}^2(A_{i*}, \mathcal{U}).$$

By the definition of the sensitivity of a point, that implies that

$$\sigma(A_{i*}) = \frac{\operatorname{dist}^2(A_{i*}, \mathcal{U})}{\sum_{i'=1}^{n} \operatorname{dist}^2(A_{i'*}, \mathcal{U})} \leq \frac{\operatorname{dist}^2(A_{i*}, \mathcal{U})}{\frac{v}{(1+\varepsilon)^2} \operatorname{dist}^2(A_{i*}, \mathcal{U})} = \frac{(1+\varepsilon)^2}{v}$$

for all $A_{i*} \in S_v$. Computing the sum over all v, the total sensitivity of A is bounded by

$$\begin{aligned} \sum_{v=1}^{\ell} \sum_{A_{i*} \in A_v} \sigma(A_{i*}) \leq \sum_{v=1}^{\ell} \frac{|S_v|(1+\varepsilon)^2}{v} & \leq f_\varepsilon(n,d) \cdot (1+\varepsilon)^2 \cdot \sum_{v=1}^{\ell} \frac{1}{v} = f_\varepsilon(n,d) \cdot (1+\varepsilon)^2 \cdot \mathcal{H}_n \\ & \leq f_\varepsilon(n,d) \cdot (1+\varepsilon)^2 \cdot (\log n + 1) \end{aligned}$$

where the last inequality follows because $\ell \leq n$ and by known bounds on the harmonic number $\mathcal{H}_n$. □

8.4.2 Constructing L_∞-coresets

Varadarajan and Xiao reduced the problem to bound the sensitivity of A to the problem to compute an L_∞-coreset for A. Thus, we now turn to see results on L_∞-coresets. We start with results on L_∞-coresets for $k = 1$. The following result is one of the first results on coresets for projective clustering problems.

Theorem 8.4.5 (Theorem 3.7 in [AHPV04])**.** *Let $A \in \mathbb{R}^{n \times d}$ be a matrix representing n points and let $\varepsilon > 0$ be given. An L_∞-(d-1,1)-coreset of size $\mathcal{O}(1/\varepsilon^{d-1})$ of A can be computed in time $\mathcal{O}(n + 1/\varepsilon^{d-1})$.*

Notice that even though the result is for affine hyperplanes, it implies a coreset result for subspaces of lower dimension as well.

Observation 8.4.6. *If $S \in \mathbb{R}^{r\times d}$ is a L_∞-(d-1,1)-coreset, then it is also a L_∞-$(j,1)$-coreset for any $j \leq d-1$.*

Proof. Let T be any j-dimensional affine subspace for $j < d-1$, and let x be a point that maximizes the distance to T among all points in A, i. e., $\operatorname{dist}(x,T) = \max_{i=1,\dots,n} \operatorname{dist}(A_{i*},T)$. Consider the affine hyperplane H that goes through $\pi_T(x)$ and is orthogonal to the vector $x - \pi_V(x)$. Notice that H contains T (because T is also orthogonal to $x-\pi_V(x)$), and that $\pi_T(x) = \pi_H(x)$. In particular, the distance between x and H is the same as the distance between x and T. Since S is an L_∞-(d-1,1)-coreset, it contains a point S_{i*} that satisfies

$$\operatorname{dist}(x,T) = \operatorname{dist}(x,H) \leq \max_{i=1,\dots,n} \operatorname{dist}(A_{i*},H) \leq (1+\varepsilon)\operatorname{dist}(S_{i*},H) \leq (1+\varepsilon)\operatorname{dist}(S_{i*},T).$$

This implies that S is an L_∞-$(j,1)$-coreset. □

We can also conclude that the coreset size can be made smaller if the input matrix has a rank which is smaller than d as we see in the following corollary.

Corollary 8.4.7. *Let $A \in \mathbb{R}^{n\times d}$ be a matrix representing n points which is of rank r. Let $0 \leq \varepsilon < 1$ and $j \in \{0,\dots,d-1\}$ be given. There exists an L_∞-(j,1)-coreset of size $\mathcal{O}(1/\varepsilon^{j+r})$ of A.*

Proof. Notice that A is contained in an r-dimensional linear subspace of $\mathbb{R}^d$, and that any affine j-dimensional subspace is contained in a $j+1$-dimensional linear subspace. Let V be an arbitrary subspace of dimension $j+r+1$ that contains A (if $j+r+1 > d$, let $V = \mathbb{R}^d$).

We represent A in an arbitrary basis of V which results in a matrix $A' \in \mathbb{R}^{n\times(r+j+1)}$. By Theorem 8.4.5 and Observation 8.4.6 there exists a subset $S' \subset A'$ of size $\mathcal{O}(1/\varepsilon^{j+r})$ that satisfies the coreset property for A' and all j-dimensional affine subspaces of V. Since A and A' describe the same points, this also holds for A when we replace S' by the corresponding subset S of A. Now let V' be any affine j-dimensional subspace of $\mathbb{R}^d$. We can define a rotation that rotates V' into V while changing neither A nor the distances between points in A and V'. We get a subspace V'' that lies within V. Thus, S satisfies the coreset property for V'' and this implies that it satisfies the coreset property for V' as well. See the proof of Theorem 4.4.1 for the exact way to specify the rotation. □

Now we work through the L_∞-coreset result by Edwards and Varadarajan [EV05] for $k > 1$. The result is originally stated for $j = d-1$ but works for smaller j as well and we will see that it then produces a smaller coreset. In the proof, we need the following modified lemma from [EV05]. It gives a lower and upper bound on the distance of a point on the integer grid $\{1,\dots,\Delta\}$ to an affine subspace spanned by grid points. The affine subspace spanned by $x_0,\dots,x_t$ is the subspace $\{x_0+V\}$ where V is the span of $x_1-x_0,\dots,x_t-x_0$.

Lemma 8.4.8 (Variation of Proposition 1 in [EV05])**.** *Let $A \in \{1,\dots,\Delta\}^{n\times d}$ be a set of n points, let k be an integer and let $0 \leq j \leq d-1$ be an integer. Let $\bar{T}$ be a j-dimensional affine subspace $\{x_0+\bar{V}\}$ where $x_0 \in \{1,\dots,\Delta\}^d$ and V is spanned by j linearly independent $x_1,\dots,x_j \in \{1,\dots,\Delta\}^d$. Then for any $z \in \{1,\dots,\Delta\}^d$, $\operatorname{dist}^2(z,\bar{T})$ is either zero or between $1/(d\Delta)^{f(j)}$ and $\sqrt{d}\Delta$ for a superlinear function f in j.*

Proof. The upper bound is easy to see since the distance between two points from $\{1,\dots,\Delta\}^d$ can at most be $\sqrt{d}\Delta$ and $\bar{T}$ contains a point from $\{1,\dots,\Delta\}^d$. Thus, the distance between any point in A and $\bar{T}$ is bounded by $\sqrt{d}\Delta$.

We start with the case that x_0 is the origin. The points $x_1,\dots,x_j$ form a basis, but they are not orthogonal. An orthogonal basis can be computed using the orthogonalization process by Gram and Schmidt, see A.5.2 in the appendix. The orthogonal basis is given by

$$y_\ell := x_\ell - \sum_{a=1}^{\ell-1} \frac{x_\ell^T y_a}{||y_a||^2} y_a$$

and an orthonormal basis is defined by setting $y'_\ell := y_\ell/||y_\ell||$. The x_ℓ contain coordinates between 1 and Δ. The y_ℓ result from them by subtracting, multiplying and dividing different terms. They can thus have fractional coordinates p/q. Here, p and q come from a range that is increased by every operation. However, since the number of operations only depends on ℓ, y_ℓ contains fractional coordinates p/q for $|p|, q \in \{1,\dots,(d\Delta)^{f'(\ell)}\}$ for a function f' in ℓ. This implies that the coordinates of y'_ℓ are of the form p/q, $|p|, q \in \{1,\dots,(d\Delta)^{f(\ell)}\}$ for another function f in ℓ.

Now for any grid point $z \in \{1,\dots,\Delta\}^d$, we have $\text{dist}^2(z,T) = ||z||^2 - \sum_{\ell=1}^{j}(z^T y'_\ell)^2$ where $||z||^2$ is an integer and the subtrahend can be expressed as a fraction with an integer nominator and a denominator that is at most $(d\Delta)^{f(j)}$ for a function f in j. Thus, each point either costs nothing or at least $1/(d\Delta)^{f(j)}$. If x_0 is not the origin, we compute the distance between $z - x_0$ and $T' := \{x - x_0 \mid x \in T\}$ which is spanned by $x_1 - x_0, \dots, x_j - x_0$. The coordinates of $z - x_0, x_1 - x_0, \dots, x_j - x_0$ are between $-\Delta$ and Δ, so we get the same result with a different function f. □

Let λ be the smallest integer that satisfies $\lambda \geq \log(\sqrt{d}\Delta/(1/(d\Delta)^{f(j)}))$ and notice that $\lambda \in \mathcal{O}(f(j) \cdot \log d\Delta)$.

Theorem 8.4.9 ([EV05])**.** *Let $A \in \mathbb{R}^{n\times d}$ represent a set of n points from $\{1,\dots,\Delta\}^d \subset \mathbb{R}^d$ where $\Delta \geq 2$ is an integer. Let r be the rank of A. Let $k \in \mathbb{N}^+$ and $0 < \varepsilon < 1$ be parameters. An $\mathcal{L}_\infty$-(j,k)-coreset of A with precision ε of size $((\log d\Delta)/\varepsilon)^{f(j,k,r)}$ can be computed for $j \leq d-1$. The term $f(r,j,k)$ depends on r, j and k.*

Proof. The construction by Edwards and Varadarajan works recursively. It builds an L_∞-(j,k)-coreset based on a construction of L_∞-(j,k-1)-coresets. The recursion stops when $k = 1$, and then a coreset of size $\mathcal{O}(1/\varepsilon^{j+r})$ can be constructed by Corollary 8.4.7. If $j + r > d - 1$, a coreset of size $\mathcal{O}(1/\varepsilon^{d-1})$ can be computed instead by using Theorem 8.4.5 with Observation 8.4.6.

Now we describe the construction of a L_∞-(j-k)-coreset based on the construction for $k-1$. For a set $\mathcal{T} \in \mathcal{Q}_{jk}$, we say that $T \in \mathcal{T}$ *handles* a point A_{i*} if $\text{dist}(A_{i*}, T) = \min_{T' \in \mathcal{T}} \text{dist}(A_{i*}, T')$. The key idea underlying the construction is the following observation. Whenever we compute an L_∞-(j,k-1)-coreset S of a point set Q, we gain the following information for any given set of k j-dimensional subspaces $T_1,\dots,T_k$: Either all points in S are handled by (at least) one of the subspaces $T_1,\dots,T_{k-1}$, or there is at least one point

where the closest affine subspace is T_k. In the first case, we can use that S is a L_∞-(j,k-1)-coreset for Q. In the second case, we gain information about the position and the extent of T_k.

This idea is now utilized in process with $r+1$ levels ($d+1$ in the original version). We describe the construction and sketch the proof except for one subproblem that we deal with by a lemma below.

On level 0, an L_∞-(j,k-1)-coreset S is computed for the input A. For any query $\mathcal{T} = \{T_1, \ldots, T_k\}$, we have that either all points in S are handled by $T_1, \ldots, T_{k-1}$ or there is at least one point x_0 in S that is handled by T_k. In the first case, assume that d is the maximum distance between a point in S and its closest subspace. Since all points in S are handled by a subspace T_ℓ with $\ell < k$ and since S is a L_∞-(j,k-1)-coreset, we have

$$\max_{i=1,\ldots,n} \min_{\ell=1,\ldots,k-1} \operatorname{dist}(A_{i*}, T_\ell) \leq (1+\varepsilon) \max_{S_{i*} \in S} \min_{\ell=1,\ldots,k-1} \operatorname{dist}(S_{i*}, T_\ell) = (1+\varepsilon)d.$$

Now assume that the maximum distance between any point in A and its closest subspace in $\mathcal{T}$ is d'. Since we are interested in the *closest* subspace, d' is bounded by the maximum distance of any point in A to its closest subspace in among the first $k-1$ subspaces. Thus, $d' \leq (1+\varepsilon)d$. This means that S satisfies the L_∞-(j,k)-coreset property for this query.

In the other case, one of the points is handled by T_k. Thus, the task reduces to computing a coreset under the assumption that one of the points is handled by T_k. Each point in S could be the relevant point, so the construction computes additional coresets for each choice of a point x_0 from S. The level 1 computation is called for every choice.

Level 1 then is responsible to compute a set that provides that the L_∞-(j,k)-coreset property holds for A and any query where x_0 is handled by T_k. It partitions the points in A according to their distance to x_0. By Lemma 8.4.8 (applied for a 0-dimensional affine subspace spanned by x_0), the distance of any point to x_0 is either zero or at least $(d\Delta)^{f(1)} \geq (d\Delta)^{-f(j)}$ and at most $\sqrt{d}\Delta$ where $f(j)$ is a superlinear function in j. We define $B_0[x_0] := \{x_0\}$ and $B_\ell[x_0] := \{A_{i*} \mid 2^{\ell-1}/(\sqrt{d}\Delta)^{f(j)} \leq \operatorname{dist}(A_{i*}, x_0) < 2^\ell/(\sqrt{d}\Delta)^{f(j)}\}$ for $1 \leq \ell \leq \lambda$. Notice that this is indeed a partitioning.

Now a similar argument as above is used. For each ring set $B_\ell[x_0]$, an L_∞-(j,k-1)-coreset $S_\ell[x_0]$ is computed and added to S. Let x_1 be the furthest coreset point from x_0 that is handled by T_k. Then all coreset points in ring sets further away are handled by $T_1, \ldots, T_{k-1}$. Their maximum distance thus approximates the maximum distances of the other points in their ring sets. The problem reduces to providing the coreset property for points that are in the ring set of x_1 or closer to x_0. The point x_1 can be any point of any $S_\ell[x_0]$ (or x_0), so the construction proceeds for any choice of x_1 from $\{x_0\} \cup \cup_{\ell=0}^{\lambda} S_\ell[x_0]$. If $x_1 = x_0$, we are done since x_0 is in S already. Otherwise, the second level is entered with x_0 and x_1 and the set of points in the inner rings, $A[x_0, x_1] := \cup_{\ell=1}^{a} B_\ell[x_0]$, where a is the index such that $x_1 \in B_a[x_0]$.

In Level 2, we know that both x_0 and x_1 are handled by T_k, and that for all points that we need to consider, the distance to x_0 is at most twice the distance between x_0 and x_1. The points are now partitioned based on their distance to the affine subspace T spanned by x_0 and x_1. Since both are grid points, we can again use Lemma 8.4.8 to see that the

distance to T is at least $(d\Delta)^{-f(j)}$ and at most $\sqrt{d}\Delta$. We define $B_0[x_0, x_1]$ as the set of points with distance zero to T. We assume for now that we know a method to compute an L_∞-(j,k)-coreset for $B_0[x_0, x_1]$. We define the set $B_\ell[x_0, x_1]$ by $B_\ell[x_0, x_1] := \{A_{i*} \in A[x_0, x_1] \mid 2^{\ell-1}/(\sqrt{d}\Delta)^{f(j)} \leq \text{dist}(A_{i*}, T) < 2^\ell/(\sqrt{d}\Delta)^{f(j)}\}$, which is a partitioning. An L_∞-(j,k-1)-coreset $S_\ell[x_0, x_1]$ is computed for each $B_\ell[x_0, x_1]$ and added to S. Each point in any $S_\ell[x_0, x_1]$ is a choice for the farthest point from T which is handled by T_k. We denote it by x_2 and its index is again a. So we enter the next level with x_0, x_1 and x_2 and $A[x_0, x_1, x_2] := \cup_{\ell=1}^{a} B_\ell[x_0, x_1]$.

The next levels work similarly. Level t is entered with a series of points $x_0, x_1, \ldots, x_{t-1}$ and a point set $A[x_0, x_1, \ldots, x_{t-1}]$ for which we know that the distance between any x_j and the affine subspace spanned by $x_0, \ldots, x_i$ for $i \leq j$ is bounded by the distance between x_i and this affine subspace. The task is to compute additional points that ensure the coreset property for $A[x_0, x_1, \ldots, x_{t-1}]$. We define a subset $B_0[x_0, \ldots, x_{t-1}]$ containing those points from $A[x_0, \ldots, x_{t-1}]$ that lie on the affine subspace T spanned by $x_0, \ldots, x_{t-1}$ and partition the remaining points according to their distance to T by setting $B_\ell[x_0, \ldots, x_{t-1}] := \{A_{i*} \in A[x_0, \ldots, x_{t-1}] \mid 2^{\ell-1}/(\sqrt{d}\Delta)^{f(j)} \leq \text{dist}(A_{i*}, T) \leq 2^\ell/(\sqrt{d}\Delta)^{f(j)}\}$. An L_∞-(j,k-1)-coreset $S_\ell[x_0, \ldots, x_{t-1}]$ is computed for all $B_\ell[x_0, \ldots, x_{t-1}]$ and it is added to S. Then each point in any $S_\ell[x_0, \ldots, x_{t-1}]$ is a choice for the farthest point that is handled by T_k, we name the point x_t and we enter level $t+1$ with $x_0, \ldots, x_t$ and the set $A[x_0, \ldots, x_t] := \cup_{\ell=1}^{a} B_\ell[x_0, \ldots, x_{t-1}]$ where a is the index of the set x_t belongs to.

We enter level $r+1$ with affinely independent points $x_0, \ldots, x_r$, i. e., $x_1 - x_0, \ldots, x_r - x_0$ are linearly independent. Since A has rank r, there cannot be another point x in A such that x is affinely independent to $x_0, \ldots, x_r$. Thus, all points in $A[x_0, \ldots, x_r]$ lie within the affine subspace spanned by $x_0, \ldots, x_r$. We see below how to proceed for $B_0[x_0, x_1, \ldots, x_r] := A[x_0, \ldots, x_r]$ and the process stops.

This completes the description except for the fact that we skipped the subsets on the affine subspaces spanned by $x_0, \ldots, x_t$ on each level. We name this affine subspace $\bar{T}_t$. So the remaining task is to compute a point set that ensures the L_∞-j-k-coreset property for the subsets $B_0[x_0, x_1]$, …, $B_0[x_0, \ldots, x_r]$ (the subset $B_0[x_0]$ only contains x_0 and x_0 is in S anyway). An example for the procedure is depicted in Figure 8.4. For any $t \in \{0, \ldots, r\}$, we know that for any $1 \leq \ell \leq t$ and $a \geq \ell$ it holds that $\text{dist}(x_a, \bar{T}_{\ell-1}) \leq 2 \cdot \text{dist}(x_\ell, \bar{T}_{\ell-1})$ by construction. Below, we cite a lemma from [EV05] and conclude Observation 8.4.11 that states that under this precondition, the convex hull of $\{x_0, \ldots, x_t\}$ contains a translate of the hyper-rectangle

$$\{f'(j) \cdot (a_1(x_1 - \pi_{\bar{T}_0}(x_1)) + a_2(x_2 - \pi_{\bar{T}_1}(x_2)) + \ldots + a_t(x_t - \pi_{\bar{T}_{t-1}}(x_t))) \mid 0 \leq a_i \leq 1\}$$

where $f'(j)$ is a term that depends on j. Edwards and Varadarajan use this lemma to cover $B_0[x_0, \ldots, x_t]$ by scaled versions of this rectangle. This is possible because $B_0[x_0, \ldots, x_t]$ is covered by the rectangle

$$v_0 + \{(a_1(x_1 - \pi_{\bar{T}_0}(x_1)) + a_2(x_2 - \pi_{\bar{T}_1}(x_2)) + \ldots + a_t(x_t - \pi_{\bar{T}_{t-1}}(x_t))) \mid -2 \leq a_i \leq 2\}.$$

Thus $B_0[x_0, \ldots, x_t]$ can be covered by $\mathcal{O}(1/(f'(j)\varepsilon)^t) = O(\varepsilon^{-t})$ copies of the rectangle

$$\{(\varepsilon/2)f'(j) \cdot (a_1(x_1 - \pi_{\bar{T}_0}(x_1)) + a_2(x_2 - \pi_{\bar{T}_1}(x_2)) + \ldots + a_t(x_t - \pi_{\bar{T}_{t-1}}(x_t))) \mid 0 \leq a_i \leq 1\}.$$

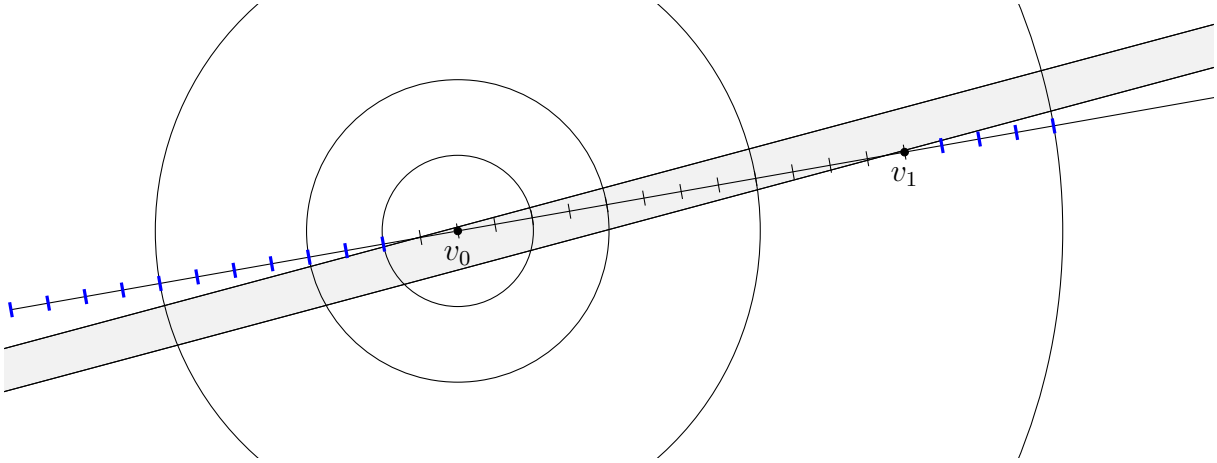


Figure 8.4: An example for an one-dimensional T_k and the construction of the rectangles. The grey area depicts T_k and all points that are within the maximum distance that any coreset point has to T_k. The blue rectangles do not intersect T_k and thus the coreset property for $k-1$ is sufficient. Only the rectangles that intersect T_k but are not contained in T_k can cause additional error, but it is bounded since the rectangles are chosen suitably small.

For each of these copies, an L_∞-(j,k-1)-coreset is computed and added to S. Consider any of the copies together with the input points and coreset points that lie within it. Again, the coreset points of these coresets are either handled by $T_1, \ldots, T_{k-1}$ in which case we are done, or there is at least one point in the copy that is handled by T_k. In particular, T_k intersects this rectangle. The rectangle is a scaled down version of a rectangle that is contained in T_k, and the width that is only $\varepsilon/2$ times the width of the rectangle contained in T_k. Thus, all points in $B_0[x_0, \ldots, x_t]$ are within distance $(1+\varepsilon)$ of the maximum distance between T_k and the coreset point from $B_0[x_0, \ldots, x_t]$ that is contained in T_k. This completes the sketch of the proof of the coreset property.

Let $S(k)$ be an upper bound on the size of the coreset computed when the number of subspaces is set to k. By Corollary 8.4.7, Theorem 8.4.5 and Observation 8.4.6, we have that $S(1) \leq \mathcal{O}(1/\varepsilon^{\min\{d-1,j+r\}})$. We first analyze the coreset size without the additional coreset points from the rectangles. On level 0, S is initialized with $S(k-1)$ points. There are $S(k-1)$ choices for x_0. For each x_0, we compute λ coresets, so there are $\lambda(S(k-1))^2$ points added to S on level 1, and level 2 is entered as often. In general, level t is entered $\lambda^{t-1}(S(k-1))^t$ times and adds $\lambda^t(S(k-1))^{t+1}$ to S, for $t = 1, \ldots, r$. Additionally, we add $\mathcal{O}(\varepsilon^{-t+1})$ coresets for the rectangles on level t for $t = 1, \ldots, r+1$. We get that

$$S(k) \leq \sum_{t=0}^{r} \lambda^t (S(k-1))^{t+1} + \sum_{t=1}^{t+1} \mathcal{O}(\varepsilon^{-t+1}).$$

Together with $S(1) \leq \mathcal{O}(\varepsilon^{j+r})$ and $\lambda \in \mathcal{O}(\log d\Delta)$, we get that the overall coreset size is $((\log d\Delta)/\varepsilon)^{f(j,k,r)}$ for a function that depends on j, k and r. □

Lemma 8.4.10 (Lemma 1 in [EV05], attributed to [BH01]). *Let $x_0, \dots, x_t$ be $t \leq d$ affinely independent points from $\mathbb{R}^d$. Denote the affine t-dimensional subspace spanned by $x_0, \dots, x_t$ by T_i. Assume that*

$$\operatorname{dist}(x_a, T_{\ell-1}) \leq 2 \cdot \operatorname{dist}(x_\ell, T_{\ell-1})$$

holds for all $1 \leq \ell \leq t$ and $a \geq \ell$. Then the convex hull of $\{x_0, \dots, x_t\}$ contains a translate of

$$\{f'(d) \cdot (a_1(x_1 - \pi_{T_0}(x_1)) + a_2(x_2 - \pi_{T_1}(x_2)) + \ldots + a_t(x_t - \pi_{T_{t-1}}(x_t))) \mid 0 \leq a_i \leq 1\}$$

where $f'(d)$ is a term that depends on the dimension d.

Observation 8.4.11. *If $x_0, x_1, \dots, x_t$ are contained in a linear subspace of dimension j, then the statement of Lemma 8.4.10 still holds when $f'(d)$ is replaced by a term $f'(j)$ that depends on j instead of the dimension d.*

Proof. All T_i lie within the subspace that is spanned by $x_0, \dots, x_t$, which is contained in a linear subspace V of dimension j. The convex hull of the $t+1$ vectors also lies within this j-dimensional subspaces. Thus, we can represent all x_i in an arbitrary basis of V without changing the distances between the points and the T_i and without distorting the convex hull. Then, we apply the lemma. Since it is applied to j-dimensional points, the term $f(d)$ is in fact a term $f(j)$. The statement of the lemma then translates back when we change the basis to the original d-dimensional basis. □

8.4.3 Obtaining the coreset result

In this section, we combine the insights from the previous parts and conclude with our coreset result. By Lemma 8.4.1, we know that for any input matrix $A \in \{1, \dots, \Delta\}^{n \times d}$, there are two possibilities. Either the rank of A is bounded by $k(j+1)$, or the optimal cost is at least $1/(d\Delta)^{f(j)}$ for a function f that depends only on j. In the first case, the points in A are contained in a $k(j+1)$-dimensional subspace. We get by Lemma 8.4.2 that $\dim(F)$ is then bounded by $\mathcal{O}(k^2 j)$. Since the rank of A is bounded by $\mathcal{O}(kj)$, Theorem 8.4.9 implies that an L_∞-(j,k)-coreset of size $(\log d\Delta/\varepsilon)^{f(j,k)}$ can be computed, where $f(j,k)$ only depends on j and k. By Lemma 8.4.4, this implies a sensitivity bound. It holds that $\mathfrak{S}(A) \leq \mathcal{O}((\log d\Delta/\varepsilon)^{f(j,k)} \log n)$. Theorem 8.3.4 then implies that there exists a coreset for A of size $\mathcal{O}(k^2 j \varepsilon^{-2} \log n((\log d\Delta)/\varepsilon)^{f(j,k)})$.

In the second case, we use the dimensionality reduction to obtain a small coreset. Let $A^{(m)}$ be the projection of A to its best fit subspace of dimension $m = \lceil 18k(j+1)/\varepsilon^2 \rceil$, and let V be an arbitrary m'-dimensional subspace that contains $A^{(m)}$ for $m' := m + j \in \mathcal{O}(kj/\varepsilon^2)$. We represent $A^{(m)}$ in an orthonormal basis of V and name the resulting $(n \times m')$-matrix A'. For an $0 < \varepsilon < 1$ we now use a grid with cell width $w_0 := (\varepsilon/4) \cdot (1/\sqrt{2nd}) \cdot (1/(d\Delta)^{f(j)})$ and move every point in A' to its closest grid cell. Let A'' be the resulting matrix. Each point is moved by less than

$$(\sqrt{d} w_0)^2 = \frac{1}{2n} \cdot \frac{\varepsilon^2}{16} \cdot \frac{1}{(d\Delta)^{f(j)}} < \frac{\varepsilon^2}{16n} \operatorname{cost}_{\ell_2^2}(A, \mathcal{T})$$

for any $\mathcal{T} \in \mathcal{Q}_{jk}$ where the inequality holds because of the lower bound on the cost function. In particular, moving all points induces a squared movement of less than

$$(\varepsilon^2/16) \min_{\mathcal{T} \in \mathcal{Q}_{jk}} \text{cost}_{\ell_2^2}(A, \mathcal{T}).$$

A'' can now be scaled by $1/w_0$ to obtain a matrix with integer coordinates, and it can be translated such that these coordinates are positive. Name the resulting matrix $A''' \in \{1, \ldots, \Delta'\}^{n \times m'}$. The coordinates of A''' are of size $\mathcal{O}(\varepsilon^{-1}\sqrt{nd}(d\Delta)^{f(j)})$. Since A''' contains points of dimension $m' \in \mathcal{O}(kj/\varepsilon^2)$, its rank can be at most $\mathcal{O}(kj/\varepsilon^2)$. Theorem 8.4.9 thus implies that an L_∞-(j,k)-coreset for A''' can be computed which is of size

$$((\log d\varepsilon^{-1}\sqrt{nd}(d\Delta)^{f(j)}))/\varepsilon)^{f(j,k,kj/\varepsilon^2)} = ((\log dn\Delta)/\varepsilon)^{g(j,k,\varepsilon^{-1})}$$

where f is a function that depends on j, k and r and g depends on j, k, and ε^{-1}. The sensitivity of A''' is thus bounded by $((\log dn\Delta)/\varepsilon)^{g(j,k,\varepsilon^{-1})}$. Notice that translating and scaling does not change the sensitivity, so it is also a bound on the sensitivity of A''. In order to transfer the sensitivity bound to A', we need the following lemma.

Lemma 8.4.12. *Let $0 \leq \varepsilon \leq 1/2$, and let $\mathcal{Q}$ be a familiy of non-empty closed subsets of $\mathbb{R}^d$. Let $A \in \mathbb{R}^{n \times d}$ and $B \in \mathbb{R}^{n \times d}$ be two matrices that satisfy*

$$||A - B||_F^2 \leq \frac{\varepsilon^2}{16} \cdot \min_{\mathcal{U} \in \mathcal{Q}} \text{cost}_{\ell_2^2}(A, \mathcal{U}).$$

Then the total sensitivity of B with respect to $\mathcal{Q}$ is bounded by

$$(1 + \varepsilon) \cdot \mathfrak{S}(A) + \frac{\varepsilon}{4}$$

where $\mathfrak{S}(A)$ is the total sensitivity of A with respect to $\mathcal{Q}$.

Proof. The total sensitivity of B is

$$\mathfrak{S}(B) = \sum_{i=1}^{n} \max_{\mathcal{U} \in \mathcal{Q}} \frac{\text{dist}^2(B_{i*}, \mathcal{U})}{\sum_{j=1}^{n} \text{dist}^2(B_{j*}, \mathcal{U})}$$

For each $i = 1, \ldots, n$, we denote the $\mathcal{U} \in \mathcal{Q}$ that maximizes $\frac{\text{dist}^2(B_{i*}, \mathcal{U})}{\sum_{j=1}^{n} \text{dist}^2(B_{j*}, \mathcal{U})}$ by $\mathcal{U}(B_{i*})$. Notice that Corollary 2.3.2 holds verbatim if we replace C be any closed and non-empty set from $\mathbb{R}^d$. For $\Lambda := \min_{\mathcal{U} \in \mathcal{Q}} \text{cost}_{\ell_2^2}(A, \mathcal{U})$, Corollary 2.3.2 implies that $|\text{dist}^2(B, \mathcal{U}) - \text{dist}^2(A, \mathcal{U})| \leq \varepsilon \cdot \text{dist}^2(A, \mathcal{U})$ holds for all $\mathcal{U} \in \mathcal{Q}$. In particular, it holds for all denominators, implying that

$$\mathfrak{S}(B) = \sum_{i=1}^{n} \frac{\text{dist}^2(B_{i*}, \mathcal{U}(B_{i*}))}{\sum_{j=1}^{n} \text{dist}^2(B_{j*}, \mathcal{U}(B_{i*}))} \leq \sum_{i=1}^{n} \frac{\text{dist}^2(B_{i*}, \mathcal{U}(B_{i*}))}{(1-\varepsilon)\sum_{j=1}^{n} \text{dist}^2(A_{j*}, \mathcal{U}(B_{i*}))}. \tag{8.2}$$

We define a vector $\mathfrak{d} \in \mathbb{R}^n$ by setting

$$\mathfrak{d}_i := \frac{\operatorname{dist}(B_{i*}, \mathcal{U}(B_{i*}))}{\sqrt{(1-\varepsilon)\sum_{j=1}^n \operatorname{dist}^2(A_{j*}, \mathcal{U}(B_{i*}))}}.$$

Notice that $\mathfrak{S}(B) \leq ||\mathfrak{d}||^2$ because $||\mathfrak{d}||^2$ equals the last term in Inequality (8.2). By the triangle inequality, it holds that $\operatorname{dist}(B_{i*}, \mathcal{U}(B_{i*})) \leq \operatorname{dist}(B_{i*}, A_{i*}) + \operatorname{dist}(A_{i*}, \mathcal{U}(B_{i*}))$. Define the two vectors $\mathfrak{a}, \mathfrak{b} \in \mathbb{R}^n$ by setting

$$\mathfrak{a}_i := \frac{\operatorname{dist}(B_{i*}, A_{i*})}{\sqrt{(1-\varepsilon)\sum_{j=1}^n \operatorname{dist}^2(A_{j*}, \mathcal{U}(B_{i*}))}} \text{ and } \mathfrak{b}_i := \frac{\operatorname{dist}(A_{i*}, \mathcal{U}(B_{i*}))}{\sqrt{(1-\varepsilon)\sum_{j=1}^n \operatorname{dist}^2(A_{j*}, \mathcal{U}(B_{i*}))}}.$$

Then we have $\mathfrak{d} = \mathfrak{a} + \mathfrak{b}$ and $||\mathfrak{d}|| \leq ||\mathfrak{a}|| + ||\mathfrak{b}||$ by the triangle inequality in $\mathbb{R}^n$. It holds

$$\begin{aligned} ||\mathfrak{a}||^2 =& \sum_{i=1}^n \frac{\operatorname{dist}^2(B_{i*}, A_{i*})}{(1-\varepsilon)\sum_{j=1}^n \operatorname{dist}^2(A_{j*}, \mathcal{U}(B_{i*}))} \leq \sum_{i=1}^n \frac{\operatorname{dist}^2(B_{i*}, A_{i*})}{(1-\varepsilon)\min_{\mathcal{U}\in\mathcal{Q}}\sum_{j=1}^n \operatorname{dist}^2(A_{j*}, \mathcal{U})} \\ \leq& \frac{\frac{\varepsilon^2}{16}\min_{\mathcal{U}\in\mathcal{Q}} \operatorname{cost}_{\ell_2^2}(A, \mathcal{U})}{(1-\varepsilon)\min_{\mathcal{U}\in\mathcal{Q}}\sum_{j=1}^n \operatorname{dist}^2(A_{j*}, \mathcal{U})} \leq \frac{\varepsilon^2}{16(1-\varepsilon)} \end{aligned}$$

by the precondition of the lemma since $\sum_{i=1}^n \operatorname{dist}^2(B_{i*}, A_{i*}) = |A-B|_F^2$. Additionally $||\mathfrak{b}|| \leq \sqrt{\mathfrak{S}(A)}$ because $\mathcal{U}(B_{i*})$ can only cause the term to be smaller, so it holds that

$$\begin{aligned} ||\mathfrak{b}||^2 = \sum_{i=1}^n \frac{\operatorname{dist}^2(A_{i*}, \mathcal{U}(B_{i*}))}{(1-\varepsilon)\sum_{j=1}^n \operatorname{dist}^2(A_{j*}, \mathcal{U}(B_{i*}))} &\leq \sum_{i=1}^n \max_{\mathcal{U}\in\mathcal{Q}} \frac{\operatorname{dist}^2(A_{i*}, \mathcal{U})}{(1-\varepsilon)\sum_{j=1}^n \operatorname{dist}^2(A_{j*}, \mathcal{U})} \\ &= \mathfrak{S}(A). \end{aligned}$$

We get that

$$\begin{aligned} ||\mathfrak{d}||^2 &\leq (||\mathfrak{a}|| + \mathfrak{b}||)^2 \\ &\leq \frac{\varepsilon^2}{16(1-\varepsilon)} + \frac{\varepsilon}{4\sqrt{1-\varepsilon}}\sqrt{\mathfrak{S}(A)} + \mathfrak{S}(A) \leq (1+\varepsilon)\cdot\mathfrak{S}(A) + \frac{\varepsilon}{4}. \end{aligned}$$

For the last inequality notice that $\varepsilon \leq 1/2$ implies that $1/(1-\varepsilon) \leq 2$, and then distinguish the cases $1/(4\cdot\sqrt{1-\varepsilon}) \leq \mathfrak{S}(A)$, which implies that $\sqrt{\mathfrak{S}(A)}\cdot\varepsilon/(4\sqrt{1-\varepsilon}) \leq \varepsilon\mathfrak{S}(A)$, and $\mathfrak{S}(A) \leq 1/(4\cdot\sqrt{1-\varepsilon})$, which implies that $\sqrt{\mathfrak{S}(A)}\cdot\varepsilon/(4\sqrt{1-\varepsilon}) \leq \varepsilon/8$. The first summand is bounded by $\varepsilon/8$, too, because $\varepsilon \leq 1/2$. □

We use Lemma 8.4.12 to transfer the sensitivity bound from A'' to A'. Since these matrices satisfy $||A'-A''||_F^2 \leq \frac{\varepsilon^2}{16}\cdot\min_{\mathcal{T}\in Q_{jk}} \operatorname{cost}_{\ell_2^2}(A, \mathcal{T})$, we get that

$$\mathfrak{S}(A') \leq (1+\varepsilon)\mathfrak{S}(A'') + \frac{\varepsilon}{4} \leq 3((\log dn\Delta)/\varepsilon)^{g(j,k,\varepsilon^{-1})}.$$

Since A' is in $\mathbb{R}^{n\times m'}$, Lemma 8.4.2 implies that $\dim(F) \in \mathcal{O}(km') \subset \mathcal{O}(k^2 j/\varepsilon^2)$. Theorem 8.3.4 implies the existence of a coreset S with weights w that contains $\mathcal{O}((k^2 j/\varepsilon^2) \cdot 3((\log dn\Delta)/\varepsilon)^{g(j,k,\varepsilon^{-1})})$ points and is a coreset for the matrix A'. The matrix S is also a coreset for $A^{(m)}$ for the family of all sets of k affine subspaces of dimension j that lie within V. As in Theorem 4.4.1, we argue that we can find a rotation that rotates every set of arbitrary k affine subspaces of dimension j from $\mathbb{R}^d$ such that it lies within V and does not change $A^{(m)}$ nor the distances between $A^{(m)}$ and the set. Thus, the coreset property holds for all $\mathcal{T} \in \mathcal{Q}_{jk}$. More precisely, it holds for every $\mathcal{T} \in \mathcal{Q}_{jk}$ that

$$\left| \sum_{S_{i*}\in S} w_i \operatorname{dist}^2(S_{i*}, \mathcal{T}) - \operatorname{cost}_{\ell_2^2}(A^{(m)}, \mathcal{T}) \right| \leq \varepsilon \cdot \operatorname{cost}_{\ell_2^2}(A^{(m)}, \mathcal{T}).$$

By Theorem 3.2.3, we know that

$$\left| \left(\operatorname{cost}_{\ell_2^2}(A^{(m)}, \mathcal{T}) + \sum_{i=m+1}^{d} \sigma_i^2 \right) - \operatorname{cost}_{\ell_2^2}(A, \mathcal{T}) \right| \leq \varepsilon \cdot \operatorname{cost}_{\ell_2^2}(A, \mathcal{T}).$$

holds for all $\mathcal{T} \in \mathcal{Q}_{jk}$ since we chose $m = \lceil 18/k(j+1)/\varepsilon^2 \rceil$ and every $\mathcal{T} \in \mathcal{Q}_{jk}$ is contained in a $k(j+1)$-dimensional linear subspace. Together, we get that

$$\left| \sum_{S_{i*}\in S} w_i \operatorname{dist}^2(S_{i*}, \mathcal{T}) + \sum_{i=m+1}^{d} \sigma_i^2 - \operatorname{cost}_{\ell_2^2}(A, \mathcal{T}) \right| \leq \varepsilon(\operatorname{cost}_{\ell_2^2}(A^{(m)}, \mathcal{T}) + \operatorname{cost}_{\ell_2^2}(A, \mathcal{T})).$$

This term is bounded by $3\varepsilon \cdot \operatorname{cost}_{\ell_2^2}(A, \mathcal{T})$ since $\operatorname{cost}_{\ell_2^2}(A^{(m)}, \mathcal{T}) \leq 2\operatorname{cost}_{\ell_2^2}(A, \mathcal{T})$. We can replace ε by $\varepsilon/3$ to reduce the error to ε. We conclude that the following theorem holds. Notice that we assume $\Delta \in (nd)^{\mathcal{O}(1)}$, so the dependency on Δ is replaced by a dependency on n and d.

Corollary 8.4.13. *Let $A \in \{1, \ldots, \Delta\}^{n\times d}$ with $\Delta \in (nd)^{\mathcal{O}(1)}$. Let $\mathcal{Q}_{jk}$ be the family of k affine subspaces of $\mathbb{R}^d$ of dimension j. There exists a coreset with offset for A for $\mathcal{Q}_{jk}$ of size*

$$((\log dn)/\varepsilon)^{h(j,k,\varepsilon^{-1})}$$

where h is a function that only depends on j, k and ε^{-1}.

Part III

Appendix

A Dissimilarity measures and vector spaces

In this chapter, the input point sets are from a set $\mathbf{X}$ which is not necessarily the Euclidean space but just a (finite or infinite) set of objects together with a dissimilarity measure. Whenever we want to emphasize that we are operating in a space that is not necessarily Euclidean, we use calligraphic letters. Points are written in lower case calligraphic letters as $\mathcal{x}, \mathcal{y} \in \mathbf{X}$, and subsets of $\mathbf{X}$ in upper case letters. The input point set is usually denoted by $\mathcal{P} \subseteq \mathbf{X}$.

We review basic definitions and facts from linear algebra. We cite definitions from different references and reformulate them slightly for brevity and such that they fit together better.

A.1 Dissimilarity measures and metrics

Clustering partitions objects according to a concept of similarity or, in our case, dissimilarity. We denote general dissimilarity measures on a set $\mathbf{X}$ by $D : \mathbf{X} \times \mathbf{X} \to \mathbb{R}^{\geq 0}$ and demand that D is non-negative and zero only for the dissimilarity between a vector and itself.

We extend terms from the Euclidean case to the case of arbitrary dissimilarity measures. The distance between two finite point sets $\mathcal{P}$ and $\mathcal{Q}$ is the smallest pairwise distance between a point from $\mathcal{P}$ and a point from $\mathcal{Q}$, denoted by $D(\mathcal{P}, \mathcal{Q}) := \min\{D(\mathcal{x}, \mathcal{y}) \mid \mathcal{x} \in \mathcal{P}, \mathcal{y} \in \mathcal{Q}\}$. We denote the diameter of a finite point set $\mathcal{P} \subset \mathbf{X}$ by $\operatorname{diam}(\mathcal{P}) := \max_{\mathcal{x},\mathcal{y} \in \mathcal{P}}\{D(\mathcal{x}, \mathcal{y})\}$. For a point $\mathcal{x} \in \mathbf{X}$ and a non-negative value $r \in \mathbb{R}$, we define $\mathcal{B}(\mathcal{x}, r) := \{\mathcal{y} \in \mathbf{X} \mid D(\mathcal{x}, \mathcal{y}) \leq r\}$ as the sphere with center $\mathcal{x}$ and radius r.

When a dissimilarity measure is symmetric, then we call the measure a *distance function*. If it additionally satisfies the triangle inequality, then it is a metric and forms a metric space together with $\mathbf{X}$.

Definition A.1.1 ([Whi68], Definition 2-1 on page 45)**.** *For a non-empty set $\boldsymbol{X}$ and a function* dist $: \boldsymbol{X} \times \boldsymbol{X} \to \mathbb{R}$*, we say that $\boldsymbol{X}$ and* dist *form a metric space and in particular that* dist *is a metric on $\boldsymbol{X}$ if it holds for all $\mathcal{x}, \mathcal{y}$ and $\mathcal{z}$ from $\boldsymbol{X}$ that*

1. $\operatorname{dist}(\mathcal{x}, \mathcal{y}) \geq 0$ *and* $\operatorname{dist}(\mathcal{x}, \mathcal{y}) = 0$ *if and only if* $x = y$
2. $\operatorname{dist}(\mathcal{x}, \mathcal{y}) = \operatorname{dist}(\mathcal{y}, \mathcal{x})$
3. $\operatorname{dist}(\mathcal{x}, \mathcal{y}) \leq \operatorname{dist}(\mathcal{x}, \mathcal{z}) + \operatorname{dist}(\mathcal{z}, \mathcal{y})$.

For example, the standard Euclidean distance is a metric. The squared Euclidean distance, however, does not satisfy the triangle inequality and is not a metric, but it is a distance function.

A.2 Vector spaces

Recall that an abelian group is a set with a commutative and associative operation that has a neutral element and is invertible.

Definition A.2.1 ([All91], Definition 5.3.1 and 5.3.2 on pages 186-187)**.** *An* abelian group $(G, \circ)$ *consists of a set* G *and an operation* $\circ : G \times G \to G$ *where the image of* $a, b \in G$ *is denoted as* $a \circ b$*, such that*

1. $\circ$ *is commutative, i. e.,* $a \circ b = b \circ a$
2. $\circ$ *is associative, i. e.,* $a \circ (b \circ c) = (a \circ b) \circ c$
3. *there exists a neutral element* $e \in G$ *such that* $e \circ a = a$ *for all* $a \in G$
4. *for every* $a \in G$*, there exists an inverse element* $b \in G$ *such that* $a \circ b = e$*.*

A field F is a set with two operations $+$ and $\cdot$ such that $(F, +)$ and $(F \backslash \{0\}, \cdot)$ are abelian groups (where 0 is the neutral element of $+$) and distributivity holds.

Definition A.2.2 ([All91], Axioms 3.2.1 and Definition 3.2.2[1])**.** *A* field *is a set* F *together with two operations* $+ : F \times F \to F$ *and* $\cdot : F \cdot F \to F$ *such that* $(F, +)$ *is an abelian group where we call the neutral element* 0 *and* $(F \backslash \{0\}, \cdot)$ *is an abelian group where we call the neutral element* 1*, and it additionally holds*

5. *distributivity, i. e.,* $a \cdot (b + c) = a \cdot b + a \cdot c$ *for all* $a, b, c \in F$*.*

For example, $\mathbb{R}$ and $\mathbb{C}$ are fields.

Definition A.2.3 ([HK72], Chapter 2, pages 28-29)**.** *A* vector space $\boldsymbol{V}$ *over* F *consists of*

1. *a field* F *of so-called* scalars *with operations* $+ : F \times F \to F$ *and* $\cdot : F \times F \to F$
2. *a set* $\boldsymbol{V}$ *of so-called* vectors
3. *an operation* $+ : \boldsymbol{V} \times \boldsymbol{V} \to \boldsymbol{V}$ *(vector addition) such that* $(\boldsymbol{V}, +)$ *is an abelian group*
4. *an operation* $\cdot : \boldsymbol{V} \times F \to \boldsymbol{V}$ *(scalar multiplication) where the image of* x *and* α *is denoted by* $\alpha \cdot x$ *or* αx *for all* $\alpha \in F$ *and* $x \in \boldsymbol{V}$ *and that satisfies that*
   a) $1 \cdot x = x$ *for all* $x \in \boldsymbol{V}$
   b) $(\alpha \cdot \beta) \cdot x = \alpha \cdot (\beta \cdot x)$ *for all* $\alpha, \beta \in F$ *and all* $x \in \boldsymbol{V}$
   c) $\alpha \cdot (x + y) = \alpha x + \alpha y$ *for all* $\alpha \in F$ *and* $x, y \in \boldsymbol{V}$
   d) $(\alpha + \beta)x = \alpha x + \beta x$ *for all* $\alpha, \beta \in F$ *and* $x \in \boldsymbol{V}$*.*

[1] Notice that Z follows from the other axioms in the case of a field and A1-A4 and M1-M4 are the axioms for the multiplicative / additive abelian group.

A subspace *of a vector space* $\boldsymbol{V}$ *is a subset* $\mathcal{U} \subseteq \boldsymbol{V}$ *that is a vector space itself.*

Often, the field F is either $\mathbb{R}$ or the set of complex numbers $\mathbb{C}$. The Euclidian space $\mathbb{R}^d$ is one example for a vector space, another is the space of all $m \times n$-matrices or the space of all functions from a set to a field F. The latter space is also an example of a space with infinite dimension.

The elements of a vector space are usually called *vectors*. We also continue to use the term points, even though these points can now be differenct objects like for example functions. We still use calligraphic letters for point sets and points from vector spaces.

In vector spaces, lengths are usually measured by a norm which is a mapping that satisfies that the lengths of vectors is non-negative and only zero for the zero vector, that is linear and that satisfies the triangle inequality.

Definition A.2.4 ([Cho69], page 25)**.** *Let* F *be either* $\mathbb{R}$ *or* $\mathbb{C}$ *and let* $\boldsymbol{V}$ *be a vector space over* F*. A mapping* $|\cdot| : \boldsymbol{V} \to \mathbb{R}$ *is a norm if*

1. *from* $|\chi| = 0$ *follows* $\chi = 0$
2. $|\alpha \cdot \chi| = |\alpha| \cdot |\chi|$ *for all* $\alpha \in F$ *and* $\chi \in \boldsymbol{V}$
3. $|\chi + y| \leq |\chi| + |y|$ *for all* $\chi, y \in \boldsymbol{V}$*.*

Notice that for any norm $|\cdot|$, the mapping defined by $(\chi, y) \to |\chi - y|$ is a metric.

A.3 Inner product spaces

One particularly important special case of a metric space is an inner product space. Inner product spaces are vector spaces where it makes sense to define a notion of lengths and angles via an inner product. For this purpose, we need a vector space over either $\mathbb{R}$ or $\mathbb{C}$. Recall that for a point $x \in \mathbb{C}$, $\overline{x}$ denotes the *conjugate* of x, which is the complex number $\mathrm{Re}(x) - \mathrm{Im}(x) \cdot i$ where $\mathrm{Re}(x)$ is the real and $\mathrm{Im}(x)$ is the imaginary part of x. For a real number $x \in \mathbb{R}$, it holds that $\overline{x} = x$.

Definition A.3.1 ([HK72], pages 271+277)**.** *Let* F *be either* $\mathbb{R}$ *or* $\mathbb{C}$ *and let* $\boldsymbol{V}$ *be a vector space over* F*. An* inner product *on* $\boldsymbol{V}$ *is a mapping* $\langle \cdot, \cdot \rangle : \boldsymbol{V} \times \boldsymbol{V} \to F$ *such that* $\langle \cdot, \cdot \rangle$

1. *is linear in the first argument, i. e., it holds for all* $\chi, y, z \in \boldsymbol{V}$ *and* $\alpha \in F$ *that*
   - $\langle \chi + y, z \rangle = \langle \chi, z \rangle + \langle y, z \rangle$
   - $\langle \alpha \cdot \chi, y \rangle = \alpha \cdot \langle \chi, y \rangle$
2. *is conjugate symmetric, i. e.,* $\langle \chi, y \rangle = \overline{\langle y, \chi \rangle}$ *for all* $\chi, y \in \boldsymbol{V}$
3. *is non-negative, i. e.,* $\langle \chi, \chi \rangle \geq 0$ *for all* $\chi \in \boldsymbol{V}$ *and* $\langle \chi, \chi \rangle > 0$ *if* $x \neq 0$*.*

Notice that linearity in the first argument and conjugate symmetry imply that

4. $\langle \chi, y + z \rangle = \overline{\langle y + z, \chi \rangle} = \overline{\langle y, \chi \rangle} + \overline{\langle z, \chi \rangle} = \langle \chi, y \rangle + \langle \chi, z \rangle$ and

5. $\langle \chi, \alpha \cdot y \rangle = \overline{\langle \alpha \cdot y, \chi \rangle} = \overline{\alpha} \cdot \overline{\langle y, \chi \rangle} = \overline{\alpha} \cdot \langle \chi, y \rangle$.

Furthermore, $\langle \chi, \chi \rangle$ is always real because $\langle \chi, \chi \rangle = \overline{\langle \chi, \chi \rangle}$. Otherwise, property 3 would actually not be well-defined and the inner product would not be of much use to define lengths or angles. The following theorem states that the inner product always induces a norm, and by this, always induces a metric. When the context is clear, we use $\mathrm{dist}(\chi, y) := |\chi - y|$ and $\mathrm{dist}^2(\chi, y) := (\mathrm{dist}(\chi, y))^2$ to denote this standard metric defined by the inner product space.

Theorem A.3.2 ([HK72], page 273+277)**.** *Let $\boldsymbol{V}$ be a vector space on the field F and let $\langle \cdot, \cdot \rangle$ be an inner product on $\boldsymbol{V}$. Then, the function $|\cdot| : \boldsymbol{V} \to \mathbb{R}$ defined by $|\chi| := \sqrt{\langle \chi, \chi \rangle}$ is a norm.*

A.4 Sequences and Hilbert spaces

Let $\mathbf{X}$ be a set. A *sequence* in $\mathbf{X}$ is a mapping $\chi : \mathbb{N} \to \mathbf{X}$ that assigns an element from $\mathbf{X}$ to any natural number. It is common to use the abbreviation $\chi_i := \chi(i)$ for sequences, and sequences themselves are often written as $(\chi_i)_{i \in \mathbb{N}}$. Sequences are of particular interest in metric spaces where we have a notion of distance. Then, we can define what it means that a sequence is convergent.

Definition A.4.1 ([Rom92], page 244)**.** *Let $\boldsymbol{X}$ and* dist $: \boldsymbol{X} \times \boldsymbol{X} \to \mathbb{R}^{\geq 0}$ *form a metric space. A sequence $(\chi_i)_{i \in \mathbb{N}}$ in $\boldsymbol{X}$* converges *to χ if for any $\varepsilon > 0$ there exists an $n > 0$ such that for all $m > n$ it holds that*

$$\mathrm{dist}(\chi_m, \chi) < \varepsilon.$$

If there exists an χ such that $(\chi_i)_{i \in \mathbb{N}}$ in $\boldsymbol{X}$ in $\boldsymbol{X}$ converges to χ, we say that $(\chi_i)_{i \in \mathbb{N}}$ in $\boldsymbol{X}$ is convergent *(in $\boldsymbol{X}$) or* converges *(in $\boldsymbol{X}$).*

A special case of convergent sequences are Cauchy sequences.

Definition A.4.2 ([Rom92], page 249)**.** *Let $\boldsymbol{X}$ and* dist $: \boldsymbol{X} \times \boldsymbol{X} \to \mathbb{R}^{\geq 0}$ *form a metric space. A sequence $(\chi)_{\mathbb{N}}$ in $\boldsymbol{X}$ is a* Cauchy sequence *in $\boldsymbol{X}$ if for any $\varepsilon > 0$ from $\mathbb{R}$, there exists an $n > 0$ that satisfies for all $m_1, m_2 \geq n$ that*

$$\mathrm{dist}(\chi_{m_1}, \chi_{m_2}) \leq \varepsilon.$$

We notice (but do not prove) that all sequences that converge are Cauchy sequences. The opposite is not necessarily true.

Definition A.4.3 ([Rom92], page 250)**.** *Let $\boldsymbol{X}$ and* dist $: \boldsymbol{X} \times \boldsymbol{X} \to \mathbb{R}^{\geq 0}$ *form a metric space. The space is* complete *if any Cauchy sequence in $\boldsymbol{X}$ converges.*

We know by Theorem A.3.2 that the inner product in an inner product space induces a norm and thus that every inner product space is a metric space. If this metric space is complete, it is a Hilbert space. We need Hilbert spaces in Section 6.2.

Definition A.4.4 ([Rom92], page 265)**.** *An inner product space that is complete under the metric induced by the inner product is said to be a* Hilbert space*.*

The Euclidean space $\mathbb{R}^d$ is a Hilbert space for any $d \in \mathbb{N}$. Another example are *sequence spaces*, which are inner product spaces containing sequences in $\mathbb{R}$ or $\mathbb{C}$ as vectors.

A.5 Additional facts on Euclidean geometry

Lemma A.5.1 ([HK72])**.** *Every set of $k \leq d$ linearly independent vectors from $\mathbb{R}^d$ can be extended to a basis of $\mathbb{R}^d$.*

Proof. Corollary 2 on page 46 of [HK72] states that every set of linearly independent vectors from a finite-dimensional vector space V are part of a basis of V. As $\mathbb{R}^d$ is a finite dimensional vector space, every set of independent vectors is part of a basis and can thus be extended to such a basis. □

Theorem A.5.2 ([rPG83, Sch07, HK72])**.** *Let V be an inner product space of dimension d and let $a_1, \ldots, a_d$ be any linearly independent vectors in V. Then one can construct orthogonal vectors $b_1, \ldots, b_d$ in V such that for each $k = 1, \ldots, d$ the set $\{b_1, \ldots, b_d\}$ is a basis for the subspace spanned by $a_1, \ldots, a_d$. The vectors are defined by*

$$b_i := a_i - \sum_{j=1}^{i-1} \langle a_i, b_j \rangle \cdot \frac{b_j}{||b_j||^2}$$

Corollary A.5.3. *Let $k \leq d$ and let $a_1, \ldots, a_k \in \mathbb{R}^d$ be orthonormal vectors. Then there exist vectors $b_{k+1}, \ldots, b_d \in \mathbb{R}^d$ such that $\{a_1, \ldots, a_k, b_{k+1}, \ldots, b_d\}$ is an orthonormal basis for $\mathbb{R}^d$.*

Proof. Notice that the dimension of $\mathbb{R}^d$ is d and thus every basis of $\mathbb{R}^d$ consists of d linearly independent vectors. By Lemma A.5.1, $\{a_1, \ldots, a_k\}$ can be extended to a basis of $\mathbb{R}^d$, we name the additional vectors $a'_{k+1}, \ldots, a'_d$. Then, $\{a_1, \ldots, a_k, a'_{k+1}, \ldots, a'_d\}$ can be transformed into a orthogonal basis by using Theorem A.5.2. Notice that the formula $b_i := a_i - \sum_{j=1}^{i-1} \langle a_i, b_j \rangle \cdot \frac{b_j}{||b_j||^2}$ implies $b_i = a_i$ for $i = 1, \ldots, k$ because the a_i are orthonormal. Thus, the orthogonal basis that results from Theorem A.5.2 is of the form $\{a_1, \ldots, a_k, b'_{k+1}, \ldots, b'_d$ for orthogonal vectors $b'_{k+1}, \ldots, b'_d$. By normalizing the b'_j, we get the statement of the corollary. □

The following lemma is a standard statement from linear algebra.

Lemma A.5.4. *Let $B = \{b_1, \ldots, b_k\}$ with $k \leq d$ be a basis of a subspace U of $\mathbb{R}^d$ and let $x \in U$ be a vector. Then there exist unique coefficients $\alpha_1, \ldots, \alpha_k$ such that $x = \sum_{i=1}^{k} \alpha_i b_i$.*

Proof. We defined subspaces of $\mathbb{R}^d$ as subsets of $\mathbb{R}^d$ which are the span of a finite set of vectors. This implies that there are coefficients $\alpha_1, \ldots, \alpha_k$ such that $x = \sum_{i=1}^{k} \alpha_i b_i$. Now assume that there exist coefficients $\beta_1, \ldots, \beta_k$ with $x = \sum_{i=1}^{k} \beta_i b_i$ and such that $\alpha_i \neq \beta_i$ for at least one $i \in \{1, \ldots, k\}$. This implies that

$$\begin{aligned} \sum_{i=1}^{k} \alpha_i b_i = x = x = \sum_{i=1}^{k} \beta_i b_i \\ \Leftrightarrow \sum_{i=1}^{k} (\alpha_i - \beta_i) b_i = 0 \end{aligned}$$

and it holds $\alpha_i - \beta_i \neq 0$ for at least one $i \in \{1, \ldots, k\}$. By our definition of linear dependence and the fact that a basis must be linearly independent, this contradicts that B is a basis. Thus, no such coordinates can exist. □

B Additional information on BICO experiments

B.1 Settings

Parameter	Value
TotalMemSize (in bytes)	p% of dataset size
TotalBufferSize (in bytes)	5% of TotalMemSize
TotalOutlierTreeSize (in bytes)	5 % of TotalMemSize
WMflag	0
W vector	$(1, 1, \ldots, 1)$
M vector	$(0, 0, \ldots, 0)$
PageSize (in bytes)	1024
BDtype	4
Ftype	0
Phase1Scheme	0
RebuiltAlg	0
StatTimes	3
NoiseRate	0.25
Range	2000
CFDistr	0
H	0
Bars vector	$(100, 100, \ldots, 100)$
K	number of clusters k
InitFt	0
Ft	0
Gtype	1
GDtype	2
Qtype	0
RefineAlg	1
NoiseFlag	0
MaxRPass	1

	Covertype	Tower	Census	BigCross
p	10	5	5 (8)	25 (26)

Table B.1: Parameter settings for BIRCH.

B.2 Numerical results

		Running time (mean, in seconds)					
Dataset	k	StreamLS	StreamKM++	BICO	kmeans++	MacQueen	BIRCH
BigCross	15	6.88e+03	3.24e+03	3.26e+02	4.86e−01	2.14e+02	6.29e+02
	20	9.82e+03	3.38e+03	3.29e+02	8.86e−01	2.36e+02	6.27e+02
	25	1.60e+04	3.48e+03	3.46e+02	1.40e+00	2.53e+02	6.25e+02
	30	1.91e+04	3.60e+03	3.43e+02	2.36e+00	2.79e+02	6.21e+02
	50	–	3.95e+03	3.99e+02	7.97e+00	3.51e+02	6.25e+02
	100	–	5.02e+03	4.61e+02	3.28e+01	5.87e+02	6.21e+02
	250	–	1.21e+04	9.63e+02	2.52e+02	1.18e+03	6.30e+02
	1000	–	1.71e+05	1.42e+04	6.15e+03	4.24e+03	6.17e+02
Caltech128	50	–	3.23e+03	4.58e+02	6.99e+00	3.29e+02	–
	100	–	4.54e+03	6.07e+02	4.00e+01	4.49e+02	–
	250	–	1.61e+04	1.19e+03	3.81e+02	8.18e+02	–
	1000	–	5.06e+05	6.57e+03	1.66e+04	2.70e+03	–
Census	10	8.67e+02	9.40e+02	9.32e+01	1.31e−01	5.70e+01	1.86e+02
	20	2.30e+03	9.89e+02	1.00e+02	5.52e−01	7.39e+01	1.83e+02
	30	4.74e+03	1.06e+03	1.08e+02	1.38e+00	7.75e+01	1.83e+02
	40	9.07e+03	1.13e+03	1.19e+02	2.56e+00	8.71e+01	1.87e+02
	50	9.28e+03	1.20e+03	1.32e+02	3.93e+00	9.90e+01	1.83e+02
	100	–	1.77e+03	2.51e+02	2.00e+01	1.50e+02	1.86e+02
	250	–	6.20e+03	8.87e+02	1.59e+02	3.15e+02	1.85e+02
	1000	–	1.70e+05	7.77e+03	4.08e+03	1.13e+03	1.84e+02
Covertype	10	1.62e+02	1.54e+02	1.47e+01	2.24e−01	9.00e+00	2.55e+01
	20	4.99e+02	1.83e+02	1.68e+01	7.90e−01	1.11e+01	2.56e+01
	30	7.79e+02	2.33e+02	2.04e+01	2.09e+00	1.31e+01	2.58e+01
	40	1.44e+03	2.97e+02	2.39e+01	4.72e+00	1.48e+01	2.55e+01
	50	1.77e+03	3.91e+02	2.70e+01	8.97e+00	1.70e+01	2.57e+01
Tower	20	1.18e+03	9.14e+01	1.66e+01	1.02e−01	4.33e+00	4.23e+01
	40	3.16e+03	9.57e+01	1.67e+01	3.87e−01	7.46e+00	4.21e+01
	60	8.35e+03	1.09e+02	1.81e+01	9.04e−01	1.08e+01	4.22e+01
	80	1.33e+04	1.27e+02	1.98e+01	1.91e+00	1.40e+01	4.31e+01
	100	–	1.52e+02	2.34e+01	3.40e+00	2.02e+01	4.21e+01
	250	–	5.90e+02	5.89e+01	2.42e+01	4.70e+01	4.20e+01
	1000	–	1.65e+04	1.25e+02	5.63e+02	1.87e+02	4.18e+01

Table B.2: Running times of all tested algorithms on all tested data sets. For the randomized algorithms, the values are mean values of 100 runs.

		Running time (median, in seconds)					
Dataset	k	StreamLS	StreamKM++	BICO	kmeans++	MacQueen	BIRCH
BigCross	15	7.41e+03	3.24e+03	3.28e+02	4.60e−01	2.14e+02	6.29e+02
	20	1.03e+04	3.36e+03	3.28e+02	8.85e−01	2.36e+02	6.27e+02
	25	1.28e+04	3.47e+03	3.42e+02	1.38e+00	2.53e+02	6.25e+02
	30	1.99e+04	3.59e+03	3.40e+02	2.36e+00	2.79e+02	6.21e+02
	50	–	3.95e+03	4.04e+02	7.99e+00	3.51e+02	6.25e+02
	100	–	5.00e+03	4.80e+02	3.24e+01	5.87e+02	6.21e+02
	250	–	1.20e+04	9.41e+02	2.52e+02	1.18e+03	6.30e+02
	1000	–	1.69e+05	1.44e+04	6.02e+03	4.24e+03	6.17e+02
Caltech128	50	–	3.23e+03	4.63e+02	7.00e+00	3.29e+02	–
	100	–	4.53e+03	6.05e+02	3.97e+01	4.49e+02	–
	250	–	1.61e+04	1.19e+03	3.71e+02	8.18e+02	–
	1000	–	5.04e+05	6.77e+03	1.84e+04	2.70e+03	–
Census	10	8.30e+02	9.41e+02	9.32e+01	1.30e−01	5.70e+01	1.86e+02
	20	1.89e+03	9.88e+02	1.00e+02	5.35e−01	7.39e+01	1.83e+02
	30	3.64e+03	1.06e+03	1.07e+02	1.35e+00	7.75e+01	1.83e+02
	40	1.02e+04	1.13e+03	1.19e+02	2.53e+00	8.71e+01	1.87e+02
	50	7.46e+03	1.20e+03	1.32e+02	3.88e+00	9.90e+01	1.83e+02
	100	–	1.76e+03	2.50e+02	2.01e+01	1.50e+02	1.86e+02
	250	–	6.08e+03	8.85e+02	1.59e+02	3.15e+02	1.85e+02
	1000	–	1.68e+05	7.85e+03	4.01e+03	1.13e+03	1.84e+02
Covertype	10	1.25e+02	1.54e+02	1.47e+01	2.20e−01	9.00e+00	2.55e+01
	20	4.13e+02	1.83e+02	1.68e+01	7.90e−01	1.11e+01	2.56e+01
	30	7.75e+02	2.31e+02	2.04e+01	2.04e+00	1.31e+01	2.58e+01
	40	1.77e+03	3.00e+02	2.39e+01	4.64e+00	1.48e+01	2.55e+01
	50	1.61e+03	3.90e+02	2.69e+01	8.72e+00	1.70e+01	2.57e+01
Tower	20	1.27e+03	9.14e+01	1.65e+01	1.00e−01	4.33e+00	4.23e+01
	40	3.22e+03	9.55e+01	1.66e+01	3.60e−01	7.46e+00	4.21e+01
	60	7.67e+03	1.08e+02	1.79e+01	8.90e−01	1.08e+01	4.22e+01
	80	1.55e+04	1.26e+02	1.95e+01	1.88e+00	1.40e+01	4.31e+01
	100	–	1.52e+02	2.27e+01	3.43e+00	2.02e+01	4.21e+01
	250	–	5.85e+02	5.62e+01	2.44e+01	4.70e+01	4.20e+01
	1000	–	1.65e+04	1.16e+02	5.60e+02	1.87e+02	4.18e+01

Table B.3: Running times of all tested algorithms on all tested data sets. For the randomized algorithms, the values are median values of 100 runs.

		Running time (coefficient of variation)					
Dataset	k	StreamLS	StreamKM++	BICO	kmeans++	MacQueen	BIRCH
BigCross	15	0.6361	0.0034	0.0877	0.2284	n.a.	n.a.
	20	0.6003	0.0114	0.0247	0.1823	n.a.	n.a.
	25	0.4200	0.0042	0.0532	0.2309	n.a.	n.a.
	30	0.5221	0.0037	0.0392	0.1636	n.a.	n.a.
	50	–	0.0077	0.0695	0.1987	n.a.	n.a.
	100	–	0.0237	0.1032	0.1127	n.a.	n.a.
	250	–	0.0798	0.1340	0.1067	n.a.	n.a.
	1000	–	0.1221	0.1308	0.1768	n.a.	n.a.
Caltech128	50	–	0.0114	0.1180	0.1796	n.a.	–
	100	–	0.0273	0.0771	0.1318	n.a.	–
	250	–	0.0903	0.0711	0.1401	n.a.	–
	1000	–	0.1075	0.1098	0.2783	n.a.	–
Census	10	0.4917	0.0035	0.0373	0.1515	n.a.	n.a.
	20	0.5103	0.0033	0.0415	0.1681	n.a.	n.a.
	30	0.5094	0.0068	0.0319	0.1863	n.a.	n.a.
	40	0.2709	0.0081	0.0320	0.1256	n.a.	n.a.
	50	0.5476	0.0121	0.0299	0.1561	n.a.	n.a.
	100	–	0.0415	0.0328	0.1754	n.a.	n.a.
	250	–	0.1261	0.0505	0.0907	n.a.	n.a.
	1000	–	0.1701	0.0792	0.1244	n.a.	n.a.
Covertype	10	0.6474	0.0061	0.0098	0.1543	n.a.	n.a.
	20	0.6471	0.0224	0.0149	0.1399	n.a.	n.a.
	30	0.6461	0.0475	0.0218	0.1448	n.a.	n.a.
	40	0.4677	0.0651	0.0342	0.1259	n.a.	n.a.
	50	0.2110	0.0673	0.0385	0.1434	n.a.	n.a.
Tower	20	0.6026	0.0149	0.0186	0.1724	n.a.	n.a.
	40	0.4067	0.0137	0.0324	0.2622	n.a.	n.a.
	60	0.4886	0.0237	0.0533	0.1315	n.a.	n.a.
	80	0.3375	0.0361	0.0535	0.1300	n.a.	n.a.
	100	–	0.0496	0.1116	0.1419	n.a.	n.a.
	250	–	0.1054	0.1670	0.1093	n.a.	n.a.
	1000	–	0.0845	0.2725	0.0930	n.a.	n.a.

Table B.4: Coefficients of variation of the running times of the 100 runs of all tested algorithms on all tested data sets. The coefficient of variation is defined as the ratio of the standard deviation to the mean: $\mathrm{cv}(x) = \mathrm{sd}(x) \,/\, \mathrm{mean}(x)$.

		Costs (mean)				
Dataset	k	StreamLS	StreamKM++	BICO	MacQueen	BIRCH
BigCross	15	5.14e+12	5.04e+12	5.04e+12	6.66e+12	6.38e+12
	20	4.24e+12	4.16e+12	4.16e+12	5.88e+12	5.82e+12
	25	3.68e+12	3.59e+12	3.58e+12	5.31e+12	4.69e+12
	30	3.17e+12	3.18e+12	3.17e+12	4.54e+12	3.89e+12
	50	–	2.36e+12	2.34e+12	3.42e+12	2.81e+12
	100	–	1.63e+12	1.61e+12	2.18e+12	1.83e+12
	250	–	1.03e+12	1.01e+12	1.14e+12	1.14e+12
	1000	–	5.24e+11	5.13e+11	7.17e+11	5.79e+11
Caltech128	50	–	3.26e+11	3.24e+11	3.16e+11	–
	100	–	3.07e+11	3.01e+11	2.93e+11	–
	250	–	2.81e+11	2.68e+11	2.63e+11	–
	1000	–	2.43e+11	2.28e+11	2.27e+11	–
Census	10	2.44e+08	2.48e+08	2.48e+08	2.82e+08	3.79e+08
	20	1.87e+08	1.90e+08	1.90e+08	2.04e+08	3.04e+08
	30	1.57e+08	1.59e+08	1.60e+08	1.75e+08	2.35e+08
	40	1.35e+08	1.40e+08	1.41e+08	1.63e+08	2.24e+08
	50	1.26e+08	1.28e+08	1.28e+08	1.46e+08	2.13e+08
	100	–	1.03e+08	9.95e+07	1.08e+08	1.70e+08
	250	–	8.00e+07	7.24e+07	7.71e+07	1.24e+08
	1000	–	5.87e+07	4.67e+07	5.08e+07	9.22e+07
Covertype	10	3.51e+11	3.42e+11	3.41e+11	5.51e+11	4.43e+11
	20	2.14e+11	2.06e+11	2.05e+11	4.77e+11	2.46e+11
	30	1.62e+11	1.57e+11	1.56e+11	4.10e+11	1.86e+11
	40	1.37e+11	1.31e+11	1.30e+11	3.57e+11	1.55e+11
	50	1.11e+11	1.15e+11	1.14e+11	3.51e+11	1.36e+11
Tower	20	6.15e+08	6.22e+08	6.22e+08	3.36e+09	9.19e+08
	40	3.21e+08	3.33e+08	3.32e+08	2.70e+09	4.70e+08
	60	1.91e+08	2.43e+08	2.42e+08	1.98e+09	3.75e+08
	80	1.68e+08	1.95e+08	1.93e+08	1.97e+09	3.25e+08
	100	–	1.65e+08	1.63e+08	1.53e+09	3.05e+08
	250	–	8.83e+07	8.72e+07	1.15e+09	2.25e+08
	1000	–	3.63e+07	3.63e+07	5.53e+08	1.76e+08

Table B.5: Costs of all tested algorithms on all tested data sets. For the randomized algorithms, the values are mean values of 100 runs.

		Memory usage (mean, in KB)					
Dataset	k	StreamLS	StreamKM++	BICO	kmeans++	MacQueen	BIRCH
BigCross	15	1.27e+07	1.07e+05	2.75e+04	1.02e+04	8.05e+03	6.84e+05
	20	1.27e+07	1.41e+05	3.53e+04	1.20e+04	9.58e+03	6.84e+05
	25	1.27e+07	1.75e+05	4.35e+04	1.35e+04	1.12e+04	6.84e+05
	30	1.27e+07	1.95e+05	5.17e+04	1.60e+04	1.27e+04	6.84e+05
	50	–	3.22e+05	8.63e+04	2.36e+04	1.89e+04	6.84e+05
	100	–	5.96e+05	1.64e+05	3.87e+04	3.44e+04	6.84e+05
	250	–	1.25e+06	4.03e+05	8.87e+04	8.08e+04	6.84e+05
	1000	–	4.12e+06	1.56e+06	3.52e+05	3.13e+05	6.84e+05
Caltech128	50	–	5.32e+05	1.67e+05	4.17e+04	7.80e+04	–
	100	–	9.71e+05	3.28e+05	8.16e+04	1.52e+05	–
	250	–	1.97e+06	7.90e+05	2.11e+05	3.75e+05	–
	1000	–	6.56e+06	3.13e+06	8.08e+05	1.49e+06	–
Census	10	3.15e+06	7.80e+04	2.40e+04	9.29e+03	7.86e+03	1.70e+05
	20	3.15e+06	1.41e+05	4.30e+04	1.34e+04	1.22e+04	1.70e+05
	30	3.16e+06	1.94e+05	6.18e+04	1.73e+04	1.66e+04	1.70e+05
	40	3.16e+06	2.57e+05	8.05e+04	2.13e+04	2.10e+04	1.70e+05
	50	3.16e+06	2.93e+05	9.81e+04	2.42e+04	2.54e+04	1.70e+05
	100	–	5.29e+05	1.90e+05	4.64e+04	4.73e+04	1.70e+05
	250	–	1.18e+06	4.53e+05	1.02e+05	1.13e+05	1.70e+05
	1000	–	3.67e+06	1.72e+06	3.93e+05	4.42e+05	1.70e+05
Covertype	10	6.41e+05	5.85e+04	2.07e+04	8.72e+03	6.38e+03	5.03e+04
	20	6.41e+05	1.04e+05	3.60e+04	1.11e+04	9.36e+03	5.03e+04
	30	6.41e+05	1.40e+05	5.08e+04	1.44e+04	1.23e+04	5.03e+04
	40	6.41e+05	1.85e+05	6.54e+04	1.87e+04	1.53e+04	5.03e+04
	50	6.41e+05	2.07e+05	8.10e+04	2.37e+04	1.83e+04	5.03e+04
Tower	20	1.39e+06	4.68e+04	9.74e+03	6.32e+03	3.38e+03	6.11e+04
	40	1.39e+06	8.40e+04	1.51e+04	7.25e+03	3.50e+03	6.11e+04
	60	1.39e+06	1.16e+05	2.05e+04	8.37e+03	3.60e+03	6.11e+04
	80	1.39e+06	1.53e+05	2.60e+04	9.66e+03	3.70e+03	6.11e+04
	100	–	1.77e+05	3.16e+04	1.12e+04	3.78e+03	6.11e+04
	250	–	4.03e+05	7.25e+04	2.04e+04	4.51e+03	6.11e+04
	1000	–	1.30e+06	2.48e+05	5.73e+04	8.21e+03	6.11e+04

Table B.6: Maximum memory usage of all tested algorithms on all tested data sets. For the randomized algorithms, the values are mean values of 100 runs.

		Costs (median)				
Dataset	k	StreamLS	StreamKM++	BICO	MacQueen	BIRCH
BigCross	15	5.22e+12	5.02e+12	5.03e+12	6.66e+12	6.38e+12
	20	4.34e+12	4.16e+12	4.16e+12	5.88e+12	5.82e+12
	25	3.69e+12	3.59e+12	3.58e+12	5.31e+12	4.69e+12
	30	3.28e+12	3.18e+12	3.17e+12	4.54e+12	3.89e+12
	50	–	2.36e+12	2.34e+12	3.42e+12	2.81e+12
	100	–	1.63e+12	1.61e+12	2.18e+12	1.83e+12
	250	–	1.03e+12	1.01e+12	1.14e+12	1.14e+12
	1000	–	5.24e+11	5.13e+11	7.17e+11	5.79e+11
Caltech128	50		3.26e+11	3.24e+11	3.16e+11	–
	100		3.07e+11	3.00e+11	2.93e+11	–
	250		2.81e+11	2.67e+11	2.63e+11	–
	1000		2.43e+11	2.28e+11	2.27e+11	–
Census	10	2.40e+08	2.48e+08	2.47e+08	2.82e+08	3.79e+08
	20	1.86e+08	1.90e+08	1.90e+08	2.04e+08	3.04e+08
	30	1.56e+08	1.59e+08	1.59e+08	1.75e+08	2.35e+08
	40	1.34e+08	1.40e+08	1.41e+08	1.63e+08	2.24e+08
	50	1.26e+08	1.28e+08	1.28e+08	1.46e+08	2.13e+08
	100	–	1.02e+08	9.95e+07	1.08e+08	1.70e+08
	250	–	8.00e+07	7.24e+07	7.71e+07	1.24e+08
	1000	–	5.86e+07	4.67e+07	5.08e+07	9.22e+07
Covertype	10	3.59e+11	3.41e+11	3.40e+11	5.51e+11	4.43e+11
	20	2.15e+11	2.06e+11	2.04e+11	4.77e+11	2.46e+11
	30	1.64e+11	1.57e+11	1.56e+11	4.10e+11	1.86e+11
	40	1.37e+11	1.31e+11	1.30e+11	3.57e+11	1.55e+11
	50	1.11e+11	1.15e+11	1.14e+11	3.51e+11	1.36e+11
Tower	20	6.35e+08	6.22e+08	6.21e+08	3.36e+09	9.19e+08
	40	3.13e+08	3.33e+08	3.32e+08	2.70e+09	4.70e+08
	60	1.99e+08	2.43e+08	2.42e+08	1.98e+09	3.75e+08
	80	1.68e+08	1.95e+08	1.93e+08	1.97e+09	3.25e+08
	100	–	1.65e+08	1.63e+08	1.53e+09	3.05e+08
	250	–	8.83e+07	8.72e+07	1.15e+09	2.25e+08
	1000	–	3.63e+07	3.63e+07	5.53e+08	1.76e+08

Table B.7: Costs of all tested algorithms on all tested data sets. For the randomized algorithms, the values are median values of 100 runs.

		Costs (coefficient of variation)				
Dataset	k	StreamLS	StreamKM++	BICO	MacQueen	BIRCH
BigCross	15	0.0551	0.0073	0.0075	n.a.	n.a.
	20	0.0698	0.0064	0.0061	n.a.	n.a.
	25	0.0251	0.0068	0.0057	n.a.	n.a.
	30	0.0722	0.0064	0.0067	n.a.	n.a.
	50	–	0.0058	0.0036	n.a.	n.a.
	100	–	0.0044	0.0027	n.a.	n.a.
	250	–	0.0035	0.0020	n.a.	n.a.
	1000	–	0.0026	0.0011	n.a.	n.a.
Caltech128	50		0.0019	0.0072	n.a.	–
	100		0.0027	0.0051	n.a.	–
	250		0.0041	0.0044	n.a.	–
	1000		0.0082	0.0019	n.a.	–
Census	10	0.0490	0.0220	0.0184	n.a.	n.a.
	20	0.0229	0.0139	0.0150	n.a.	n.a.
	30	0.0167	0.0132	0.0139	n.a.	n.a.
	40	0.0235	0.0116	0.0130	n.a.	n.a.
	50	0.0118	0.0098	0.0100	n.a.	n.a.
	100	–	0.0080	0.0072	n.a.	n.a.
	250	–	0.0152	0.0040	n.a.	n.a.
	1000	–	0.0404	0.0018	n.a.	n.a.
Covertype	10	0.0730	0.0061	0.0049	n.a.	n.a.
	20	0.0527	0.0065	0.0054	n.a.	n.a.
	30	0.0466	0.0068	0.0060	n.a.	n.a.
	40	0.0164	0.0061	0.0041	n.a.	n.a.
	50	0.0195	0.0051	0.0034	n.a.	n.a.
Tower	20	0.0777	0.0123	0.0105	n.a.	n.a.
	40	0.0614	0.0075	0.0064	n.a.	n.a.
	60	0.0739	0.0068	0.0058	n.a.	n.a.
	80	0.0130	0.0069	0.0048	n.a.	n.a.
	100	–	0.0055	0.0046	n.a.	n.a.
	250	–	0.0034	0.0025	n.a.	n.a.
	1000	–	0.0015	0.0015	n.a.	n.a.

Table B.8: Coefficients of variation of the costs of the 100 runs of all tested algorithms on all tested data sets. The coefficient of variation is defined as the ratio of the standard deviation to the mean: $\mathrm{cv}(x) = \mathrm{sd}(x) \,/\, \mathrm{mean}(x)$.

		Memory usage (median, in KB)					
Dataset	k	StreamLS	StreamKM++	BICO	kmeans++	MacQueen	BIRCH
BigCross	15	1.27e+07	1.07e+05	2.75e+04	1.01e+04	8.05e+03	6.84e+05
	20	1.27e+07	1.41e+05	3.53e+04	1.20e+04	9.58e+03	6.84e+05
	25	1.27e+07	1.75e+05	4.36e+04	1.30e+04	1.12e+04	6.84e+05
	30	1.27e+07	1.95e+05	5.17e+04	1.60e+04	1.27e+04	6.84e+05
	50	–	3.22e+05	8.63e+04	2.42e+04	1.89e+04	6.84e+05
	100	–	5.96e+05	1.64e+05	3.87e+04	3.44e+04	6.84e+05
	250	–	1.25e+06	4.03e+05	8.87e+04	8.08e+04	6.84e+05
	1000	–	4.11e+06	1.56e+06	3.52e+05	3.13e+05	6.84e+05
Caltech128	50	–	5.32e+05	1.67e+05	4.20e+04	7.80e+04	–
	100	–	9.70e+05	3.28e+05	8.18e+04	1.52e+05	–
	250	–	1.97e+06	7.91e+05	2.11e+05	3.75e+05	–
	1000	–	6.56e+06	3.13e+06	8.09e+05	1.49e+06	–
Census	10	3.15e+06	7.80e+04	2.40e+04	9.27e+03	7.86e+03	1.70e+05
	20	3.15e+06	1.41e+05	4.29e+04	1.34e+04	1.22e+04	1.70e+05
	30	3.16e+06	1.94e+05	6.18e+04	1.73e+04	1.66e+04	1.70e+05
	40	3.16e+06	2.57e+05	8.05e+04	2.14e+04	2.10e+04	1.70e+05
	50	3.16e+06	2.93e+05	9.80e+04	2.41e+04	2.54e+04	1.70e+05
	100	–	5.29e+05	1.90e+05	4.46e+04	4.73e+04	1.70e+05
	250	–	1.18e+06	4.53e+05	1.02e+05	1.13e+05	1.70e+05
	1000	–	3.66e+06	1.71e+06	3.93e+05	4.42e+05	1.70e+05
Covertype	10	6.41e+05	5.85e+04	2.07e+04	8.72e+03	6.38e+03	5.03e+04
	20	6.41e+05	1.04e+05	3.60e+04	1.11e+04	9.36e+03	5.03e+04
	30	6.41e+05	1.40e+05	5.08e+04	1.44e+04	1.23e+04	5.03e+04
	40	6.41e+05	1.85e+05	6.54e+04	1.87e+04	1.53e+04	5.03e+04
	50	6.41e+05	2.06e+05	8.10e+04	2.37e+04	1.83e+04	5.03e+04
Tower	20	1.39e+06	4.68e+04	9.74e+03	6.34e+03	3.38e+03	6.11e+04
	40	1.39e+06	8.40e+04	1.50e+04	7.07e+03	3.50e+03	6.11e+04
	60	1.39e+06	1.16e+05	2.05e+04	8.37e+03	3.60e+03	6.11e+04
	80	1.39e+06	1.53e+05	2.60e+04	9.66e+03	3.70e+03	6.11e+04
	100	–	1.77e+05	3.16e+04	1.12e+04	3.78e+03	6.11e+04
	250	–	4.03e+05	7.24e+04	2.04e+04	4.51e+03	6.11e+04
	1000	–	1.30e+06	2.49e+05	5.75e+04	8.21e+03	6.11e+04

Table B.9: Maximum memory usage of all tested algorithms on all tested data sets. For the randomized algorithms, the values are median values of 100 runs.

		Memory usage (coefficient of variation)					
Dataset	k	StreamLS	StreamKM++	BICO	kmeans++	MacQueen	BIRCH
BigCross	15	0.0000	0.0003	0.0065	0.0528	n.a.	n.a.
	20	0.0000	0.0004	0.0073	0.0462	n.a.	n.a.
	25	0.0000	0.0004	0.0083	0.0782	n.a.	n.a.
	30	0.0000	0.0005	0.0079	0.0338	n.a.	n.a.
	50	–	0.0005	0.0081	0.0738	n.a.	n.a.
	100	–	0.0005	0.0076	0.0284	n.a.	n.a.
	250	–	0.0017	0.0057	0.0146	n.a.	n.a.
	1000	–	0.0078	0.0047	0.0038	n.a.	n.a.
Caltech128	50	–	0.0026	0.0062	0.0718	n.a.	–
	100	–	0.0027	0.0094	0.0662	n.a.	–
	250	–	0.0039	0.0118	0.0247	n.a.	–
	1000	–	0.0208	0.0144	0.0382	n.a.	–
Census	10	0.0000	0.0008	0.0105	0.0329	n.a.	n.a.
	20	0.0000	0.0010	0.0095	0.0519	n.a.	n.a.
	30	0.0000	0.0009	0.0077	0.0761	n.a.	n.a.
	40	0.0000	0.0010	0.0086	0.0623	n.a.	n.a.
	50	0.0000	0.0012	0.0084	0.0727	n.a.	n.a.
	100	–	0.0013	0.0069	0.1099	n.a.	n.a.
	250	–	0.0015	0.0084	0.0279	n.a.	n.a.
	1000	–	0.0069	0.0093	0.0079	n.a.	n.a.
Covertype	10	0.0000	0.0007	0.0068	0.0111	n.a.	n.a.
	20	0.0000	0.0007	0.0077	0.0141	n.a.	n.a.
	30	0.0000	0.0007	0.0065	0.0116	n.a.	n.a.
	40	0.0000	0.0007	0.0064	0.0076	n.a.	n.a.
	50	0.0000	0.0012	0.0086	0.0058	n.a.	n.a.
Tower	20	0.0000	0.0009	0.0067	0.0090	n.a.	n.a.
	40	0.0000	0.0010	0.0115	0.0383	n.a.	n.a.
	60	0.0000	0.0010	0.0051	0.0060	n.a.	n.a.
	80	0.0000	0.0010	0.0042	0.0057	n.a.	n.a.
	100	–	0.0010	0.0060	0.0218	n.a.	n.a.
	250	–	0.0018	0.0046	0.0064	n.a.	n.a.
	1000	–	0.0198	0.0229	0.0200	n.a.	n.a.

Table B.10: Coefficients of variation of the memory usage of the 100 runs of all tested algorithms on all tested data sets. The coefficient of variation is defined as the ratio of the standard deviation to the mean: $\mathrm{cv}(x) = \mathrm{sd}(x) \,/\, \mathrm{mean}(x)$.

Bibliography

[AC09] Nir Ailon and Bernard Chazelle, *The fast johnson-lindenstrauss transform and approximate nearest neighbors*, SIAM Journal on Computing **39** (2009), 302 – 322.

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

[ADHP09] Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat, *NP-hardness of Euclidean sum-of-squares clustering*, Machine Learning **75** (2009), no. 2, 245 – 248.

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

[AE99] Pankaj K. Agarwal and Jeff Erickson, *Geometric range searching and its relatives*, Contemporary Mathematics **223** (1999), 1–56.

[AGK+04] Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit, *Local search heuristics for k-median and facility location problems*, SIAM Journal on Computing **33** (2004), no. 3, 544 – 562.

[AHP01] Pankaj K. Agarwal and Sariel Har-Peled, *Maintaining the approximate extent measures of moving points*, Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms (SODA), 2001, pp. 148 – 157.

[AHPV04] Pankaj K. Agarwal, Sariel Har-Peled, and Kasturi R. Varadarajan, *Approximating extent measures of points*, Journal of the ACM **51** (2004), no. 4, 606 – 635.

[AJM09] Nir Ailon, Ragesh Jaiswal, and Claire Monteleoni, *Streaming k-means approximation*, Proceedings of the 22nd Annual Conference on Neural Information Processing Systems (NIPS), 2009, pp. 10–18.

[All91] Reg B. J. T. Allenby, *Rings, fields and groups*, second ed., Edward Arnold, 1991.

[ALM+10] Marcel R. Ackermann, Christiane Lammersen, Marcus Märtens, Christoph Raupach, Christian Sohler, and Kamil Swierkot, *Implementation of StreamKM++*, `http://www.cs.uni-paderborn.de/en/research-group/ag-bloemer/research/clustering/streamkmpp.html`, 2010, accessed: 2014-04-13.

[Alo03] Noga Alon, *Problems and results in extremal combinatorics – i*, Discrete Mathematics **273** (2003), no. 1-3, 31 – 53.

[AMO93] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin, *Network flows: Theory, algorithms, and applications*, Prentice Hall, 1993.

[AMR+12] Marcel R. Ackermann, Marcus Märtens, Christoph Raupach, Kamil Swierkot, Christiane Lammersen, and Christian Sohler, *Streamkm++: A clustering algorithm for data streams*, ACM Journal of Experimental Algorithmics **17** (2012), article 2.4, 1–30.

[AMS99] Noga Alon, Yossi Matias, and Mario Szegedy, *The space complexity of approximating the frequency moments*, Journal of Computer and System Sciences**58** (1999), no. 1, 137 – 147.

[AN07] Arthur U. Asuncion and David J. Newman, *UCI machine learning repository*, 2007.

[And73] Michael R. Anderberg, *Cluster analysis for applications*, Academic Press, 1973.

[AP03] Pankaj K. Agarwal and Cecilia Magdalena Procopiuc, *Approximation algorithms for projective clustering*, Journal of Algorithms **46** (2003), no. 2, 115–139.

[APV05] Pankaj K. Agarwal, Cecilia Magdalena Procopiuc, and Kasturi R. Varadarajan, *Approximation algorithms for a k-line center*, Algorithmica **42** (2005), no. 3 - 4, 221 – 230.

[Ari] Aristotle, *Categories*, The Complete Works of Aristotle – The Revised Oxford Translation (Jonathan Barnes, ed.), Princeton University Press.

[Aro98] Sanjeev Arora, *Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems*, Journal of the ACM **45** (1998), no. 5, 753 – 782.

[ARR98] Sanjeev Arora, Prabhakar Raghavan, and Satish Rao, *Approximation schemes for euclidean k-medians and related problems*, Proceedings of the 30th ACM Symposium on the Theory of Computing (STOC), 1998, pp. 106 – 113.

[AV06] David Arthur and Sergei Vassilvitskii, *How slow is the k-means method?*, Proceedings of the 22nd ACM Symposium on Computational Geometry (SoCG), 2006, pp. 144 – 153.

[AV07] ———, *k-means++: the advantages of careful seeding*, Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA), 2007, pp. 1027 – 1035.

[AY09] Charu C. Aggarwal and Philip S. Yu, *A survey of uncertain data algorithms and applications*, IEEE Transactions on Knowledge and Data Engineering **21** (2009), no. 5, 609 – 623.

[Baj88] Chandrajit L. Bajaj, *The algebraic degree of geometric optimization problems*, Discrete & Computational Geometry **3** (1988), 177 – 191.

[Bar96] Yair Bartal, *Probabilistic approximations of metric spaces and its algorithmic applications*, Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 1996, pp. 184 – 193.

[Bar98] ———, *On approximating arbitrary metrices by tree metrics*, Proceedings of the 30th ACM Symposium on the Theory of Computing (STOC), 1998, pp. 161 – 168.

[BC03] Mihai Bădoiu and Kenneth L. Clarkson, *Smaller core-sets for balls*, Proceedings of the 14th ACM-SIAM Symposium on Discrete Algorithms (SODA), 2003, pp. 801 – 802.

[BC05] Bo Brinkman and Moses Charikar, *On the impossibility of dimension reduction in l_1*, Journal of the ACM **52** (2005), no. 5, 766 – 788.

[BC08] Mihai Bădoiu and Kenneth L. Clarkson, *Optimal core-sets for balls*, Computational Geometry **40** (2008), no. 1, 14–22.

[BEL13] Maria-Florina Balcan, Steven Ehrlich, and Yingyu Liang, *Distributed clustering on graphs*, CoRR **abs/1306.0604** (2013), accessed: 2015-05-03.

[Ber06] Pavel Berkhin, *A survey of clustering data mining techniques*, Grouping Multidimensional Data, Springer, 2006, pp. 25 – 71.

[BH01] Gill Barequet and Sariel Har-Peled, *Efficiently approximating the minimum-volume bounding box of a point set in three dimensions*, Journal of Algorithms **38** (2001), no. 1, 91 – 109.

[BHPI02] Mihai Bădoiu, Sariel Har-Peled, and Piotr Indyk, *Approximate clustering via core-sets*, Proceedings of the 34th ACM Symposium on the Theory of Computing (STOC), 2002, pp. 250 – 257.

[BI75] E. M. Bronshteyn and L. D. Ivanov, *The approximation of convex sets by polyhedra*, Siberian Mathematical Journal **16** (1975), no. 5, 852–853.

[BMD09] Christos Boutsidis, Michael W. Mahoney, and Petros Drineas, *Unsupervised feature selection for the k-means clustering problem*, Proceedings of the 23rd Annual Conference on Neural Information Processing Systems (NIPS), 2009, pp. 153 – 161.

[BMO+11] Vladimir Braverman, Adam Meyerson, Rafail Ostrovsky, Alan Roytman, Michael Shindler, and Brian Tagiku, *Streaming k-means on well-clusterable data*, Proceedings of the 22th ACM-SIAM Symposium on Discrete Algorithms (SODA), 2011, pp. 26 – 40.

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

[BS80] Jon L. Bentley and James B. Saxe, *Decomposable searching problems i: Static-to-dynamic transformation*, Journal of Algorithms **1** (1980), no. 4, 301 – 358.

[BZD10] Christos Boutsidis, Anastasios Zouzias, and Petros Drineas, *Random Projections for k-means Clustering* , Proceedings of the 24th Annual Conference on Neural Information Processing Systems (NIPS), 2010, pp. 298 – 306.

[CCGG98] Moses Charikar, Chandra Chekuri, Ashish Goel, and Sudipto Guha, *Rounding via Trees: Deterministic Approximation Algorithms for Group Steiner Trees and* k*-Median*, Proceedings of the 30th ACM Symposium on the Theory of Computing (STOC), 1998, pp. 114 – 123.

[CCKN06] Michael Chau, Reynold Cheng, Ben Kao, and Jackey Ng, *Uncertain data mining: An example in clustering location data*, Proceedings of the 10th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), 2006, pp. 199 – 204.

[CEM+14] Michael B. Cohen, Sam Elder, Cameron Musco, Christopher Musco, and Madalina Persu, *Dimensionality reduction for k-means clustering and low rank approximation*, CoRR **abs/1410.6801** (2014), accessed: 2015-05-03.

[CG05] Moses Charikar and Sudipto Guha, *Improved combinatorial algorithms for facility location problems*, SIAM Journal on Computing **34** (2005), no. 4, 803 – 824.

[CGTS02] Moses Charikar, Sudipto Guha, Éva Tardos, and David B. Shmoys, *A constant-factor approximation algorithm for the k-median problem*, Journal of Computer and System Sciences **65** (2002), no. 1, 129 – 149.

[Che09] Ke Chen, *On coresets for k-median and k-means clustering in metric and euclidean spaces and their applications*, SIAM Journal on Computing **39** (2009), no. 3, 923 – 947.

[Cho69] Gustave Choquet, *Lectures on analysis: Volume i: Integration and topological vector spaces*, W. A. Benjamin, 1969.

[CM08] Graham Cormode and Andrew McGregor, *Approximation algorithms for clustering uncertain data*, Proceedings of the 27th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS), 2008, pp. 191–200.

[COP03] Moses Charikar, Liadan O'Callaghan, and Rina Panigrahy, *Better streaming algorithms for clustering problems*, Proceedings of the 35th ACM Symposium on the Theory of Computing (STOC), 2003, pp. 30 – 39.

[CS04] Artur Czumaj and Christian Sohler, *Sublinear-Time Approximation for Clustering Via Random Sampling*, Lecture Notes in Computer Science **3142** (2004), 396 – 407.

[CT92] Yi-Jen Chiang and Roberto Tamassia, *Dynamic algorithms in computational geometry*, Proceedings of the of the IEEE, Special Issue on Computational Geometry **89** (1992), no. 9, 1412 – 1434.

[CW79] J. Lawrence Carter and Mark N. Wegman, *Universal classes of hash functions*, Journal of Computer and System Sciences **18** (1979), no. 2, 143 – 154.

[Dal50] Tore Dalenius, *The problem of optimum stratification i*, Scandinavian Actuarial Journal **1950** (1950), no. 3 – 4, 203 –213.

[Das03] Sanjoy Dasgupta, *How fast is k-means?*, Proceedings of the 16th Conference on Learning Theory (COLT), 2003, p. 735.

[Das08] ———, *The hardness of k-means clustering*, Tech. Report CS2008-0916, University of California, 2008.

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

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

[DKM+94] Martin Dietzfelbinger, Anna R. Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Heide, Hans Rohnert, and Robert Endre Tarjan, *Dynamic perfect hashing: Upper and lower bounds*, SIAM Journal on Computing **23** (1994), no. 4, 738 – 761.

[dlVKKR03] Wenceslas Fernandez de la Vega, Marek Karpinski, Claire Kenyon, and Yuval Rabani, *Approximation schemes for clustering problems*, Proceedings of the 35th ACM Symposium on the Theory of Computing (STOC), 2003, pp. 50 – 58.

[DRVW06] Amit Deshpande, Luis Rademacher, Santosh Vempala, and Grant Wang, *Matrix approximation and projective clustering via volume sampling*, Theory of Computing **2** (2006), no. 1, 225 – 247.

[DTV11] Amit Deshpande, Madhur Tulsiani, and Nisheeth K. Vishnoi, *Algorithms and hardness for subspace approximation*, Proceedings of the 22nd ACM-SIAM Symposium on Discrete Algorithms (SODA), 2011, pp. 482–496.

[DV07] Amit Deshpande and Kasturi R. Varadarajan, *Sampling-based dimension reduction for subspace approximation*, Proceedings of the 39th ACM Symposium on the Theory of Computing (STOC), 2007, pp. 641 – 650.

[EKSX96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu, *A density-based algorithm for discovering clusters in large spatial databases with noise*, Knowledge Discovery and Data Mining, 1996, pp. 226 – 231.

[ELL09] Brian S. Everitt, Sabine Landau, and Morven Leese, *Cluster analysis*, 4 ed., Wiley, 2009.

[ES04] Michelle Effros and Leonard J. Schulman, *Deterministic clustering with data nets*, Electronic Colloquium on Computational Complexity (ECCC) (2004), no. 050.

[EV05] Michael Edwards and Kasturi R. Varadarajan, *No coreset, no cry: II*, Proceedings of the 25th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS), 2005, pp. 107–115.

[FFS06] Dan Feldman, Amos Fiat, and Micha Sharir, *Coresets forweighted facilities and their applications*, Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 2006, pp. 315 – 324.

[FGS+13] Hendrik Fichtenberger, Marc Gillé, Melanie Schmidt, Chris Schwiegelshohn, and Christian Sohler, *BICO: BIRCH Meets Coresets for k-Means Clustering* , Proceedings of the 21st Annual European Symposium on Algorithms (ESA), 2013, pp. 481–492.

[Fis87] Douglas H. Fisher, *Knowledge acquisition via incremental conceptual clustering*, Machine Learning **2** (1987), no. 2, 139 – 172.

[FL11a] Dan Feldman and Michael Langberg, *A unified framework for approximating and clustering data*, Proceedings of the 43th ACM Symposium on the Theory of Computing (STOC), 2011, pp. 569 – 578.

[FL11b] ———, *A unified framework for approximating and clustering data*, CoRR **abs/1106.1379v3** (v1 in 2011), accessed: 2015-01-15.

[FM83] Philippe Flajolet and G. Nigel Martin, *Probabilistic counting*, Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 1983, pp. 76 – 82.

[FM85] ———, *Probabilistic counting algorithms for data base applications*, Journal of Computer and System Sciences**31** (1985), no. 2, 182 – 209.

[FM88] Peter Frankl and Hiroshi Maehara, *The johnson-lindenstrauss lemma and the sphericity of some graphs*, Journal of Combinatorial Theory, Series B **44** (1988), no. 3, 355 – 362.

[FMS07] Dan Feldman, Morteza Monemizadeh, and Christian Sohler, *A ptas for k-means clustering based on weak coresets*, Proceedings of the 23rd ACM Symposium on Computational Geometry (SoCG), 2007, pp. 11 – 18.

[FMSW10] Dan Feldman, Morteza Monemizadeh, Christian Sohler, and David P. Woodruff, *Coresets and sketches for high dimensional subspace approximation problems*, Proceedings of the 21st ACM-SIAM Symposium on Discrete Algorithms (SODA), 2010, pp. 630–649.

[For65] Edward W. Forgy, *Cluster analysis of multivariate data: Efficiency versus interpretability of classifications*, Biometrics **21** (1965), 768 – 769.

[FP11] Gernot A. Fink and Thomas Plötz, *Open source project ESMERALDA*, `http://sourceforge.net/projects/esmeralda/files/esmeralda/open%201.20/`, 2011, accessed: 2014-04-13.

[FS05] Gereon Frahling and Christian Sohler, *Coresets in dynamic geometric data streams*, Proceedings of the 37th ACM Symposium on the Theory of Computing (STOC), 2005, pp. 209 – 217.

[FS08] ———, *A fast k-means implementation using coresets*, International Journal of Computational Geometry and Applications (2008), 605 – 625.

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

[GK94] Peter Gritzmann and Victor Klee, *On the complexity of some basic problems in computational convexity: II. volume and mixed volumes*, Universität Trier, Mathematik/Informatik, Forschungsbericht **94 − 07** (1994).

[GKMS01] Anna C. Gilbert, Yannis Kotidis, S. Muthukrishnan, and Martin Strauss, *Surfing wavelets on streams: One pass summaries for approximate aggregate queries*, Proceedings of the 27th International Conference on Very Large Data Bases (VLDB), 2001, pp. 79 – 88.

[GL96] Gene H. Golub and Charles F. Van Loan, *Matrix computations*, third ed., The Johns Hopkins University Press, 1996.

[GM09] Sudipto Guha and Kamesh Munagala, *Exceeding expectations and clustering uncertain data*, Proceedings of the 28th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS), 2009, pp. 269 – 278.

[GMM+03] Sudipto Guha, Adam Meyerson, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan, *Clustering data streams: Theory and practice*, IEEE Transactions on Knowledge and Data Engineering **15** (2003), no. 3, 515 – 528.

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

[Goo14] Google, *Google Scholar: Citations for [Jai10]*, `http://scholar.google.de/scholar?cites=16205305846126879965&as_sdt=2005&sciodt=0,5&hl=en`, 2014, accessed: 2014-01-31.

[Gor96] A. Gordon, *Null models in cluster validation*, From data to knowledge : theoretical and practical aspects of classification, data analysis, and knowledge organization, Springer, 1996, pp. 32 – 44.

[GRS00] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim, *Rock: A robust clustering algorithm for categorical attributes*, Information Systems **25** (2000), no. 5, 345–366.

[GRS01] ———, *Cure: An efficient clustering algorithm for large databases*, Information Systems **26** (2001), no. 1, 35 –58.

[Guh00] Sudipto Guha, *Approximation algorithms for facility location problems*, Ph.D. thesis, Stanford University, 2000.

[Har75] John A. Hartigan, *Clustering algorithms*, Wiley, 1975.

[Hau90] David Haussler, *Decision theoretic generalizations of the pac learning model*, Proceedings of the 1st International Workshop on Algorithmic Learning Theory (ALT), 1990, pp. 21 – 41.

[Hau92] ———, *Decision theoretic generalizations of the pac model for neural net and other learning applications*, Information and Computation **100** (1992), no. 1, 78 – 150.

[HBV01] Maria Halkidi, Yannis Batistakis, and Michalis Vazirgiannis:, *On clustering validation techniques*, Journal of Intelligent Information Systems (JIIS) **17** (2001), no. 2-3, 107 – 145.

[HIIK93] Susumu Hasegawa, Hiroshi Imai, Mary Inaba, and Naoki Katoh, *Efficient algorithms for variance-based k-clustering*, Proceedings of the 1st Pacific Conference on Computer Graphics and Applications, 1993, pp. 75 – 89.

[HJ97] Pierre Hansen and Brigitte Jaumard, *Cluster analysis and mathematical programming*, Mathematical Programming **79** (1997), 191 – 215.

[HK72] Kenneth M. Hoffman and Ray Kunze, *Linear algebra*, second ed., Pearson, 1972.

[HN79] Wen-Lian Hsu and George L. Nemhauser, *Easy and hard bottleneck location problems*, Discrete Applied Mathematics **1** (1979), 209 – 215.

[Hoe63] Wassily Hoeffding, *Probability Inequalities for Sums of Bounded Random Variables*, Journal of the American Statistical Association **58** (1963), no. 301, 13–30.

[HP04] Sariel Har-Peled, *No coreset, no cry*, Proceedings of the 24th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS), 2004, pp. 324 – 335.

[HPK07] Sariel Har-Peled and Akash Kushal, *Smaller coresets for k-median and k-means clustering*, Discrete & Computational Geometry **37** (2007), no. 1, 3–19.

[HPM04] Sariel Har-Peled and Soham Mazumdar, *On coresets for k-means and k-median clustering*, Proceedings of the 36th ACM Symposium on the Theory of Computing (STOC), 2004, pp. 291 – 300.

[HPS05] Sariel Har-Peled and Bardia Sadri, *How fast is the k-means method?*, Algorithmica **41** (2005), no. 3, 185 – 202.

[HPV01] Sariel Har-Peled and Kasturi R. Varadarajan, *Approximate shape fitting via linearization*, Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS), 2001, pp. 66 – 73.

[HPV02] ———, *Projective clustering in high dimensions using core-sets*, Proceedings of the 18th ACM Symposium on Computational Geometry (SoCG), 2002, pp. 312 – 318.

[HPV04] ———, *High-dimensional shape fitting in linear time*, Discrete & Computational Geometry **32** (2004), no. 2, 269 – 288.

[Hs85] Dorit S. Hochbaum and David B. shmoys, *A best possible heuristic for the k-center problem*, Mathematics of Operations Research **10** (1985), 180 – 184.

[IKI94] Mary Inaba, Naoki Katoh, and Hiroshi Imai, *Applications of weighted voronoi diagrams and randomization to variance-based k-clustering (extended abstract)*, Proceedings of the 10th ACM Symposium on Computational Geometry (SoCG), 1994, pp. 332–339.

[IM98] Piotr Indyk and Rajeev Motwani, *Approximate nearest neighbors: Towards removing the curse of dimensionality*, Proceedings of the 30th ACM Symposium on the Theory of Computing (STOC), 1998, pp. 604 – 613.

[IM04] Piotr Indyk and Jiří Matoušek, *Low-distortion embeddings of finite metric spaces*, Handbook of Discrete and Computational Geometry (Jacob E. Goodman and Joseph O'Rourke, eds.), Chapman & Hall, 2nd ed., 2004, pp. 177 – 196.

[IN07] Piotr Indyk and Assaf Naor, *Nearest neighbor preserving embeddings*, ACM Transactions on Algorithms (TALG) **3** (2007), article 31.

[Ind99] Piotr Indyk, *Sublinear time algorithms for metric space problems*, Proceedings of the 31st ACM Symposium on the Theory of Computing (STOC), 1999, pp. 428 – 434.

[Ind01] ———, *Algorithmic applications of low-distortion geometric embeddings*, Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS), 2001, pp. 10 – 33.

[Ind04] ———, *Algorithms for dynamic geometric problems over data streams*, Proceedings of the 36th ACM Symposium on the Theory of Computing (STOC), 2004, pp. 373 – 380.

[Jai10] Anil K. Jain, *Data clustering: 50 years beyond k-means*, Pattern Recognition Letters **31** (2010), no. 8, 651 – 666.

[JD88] Anil K. Jain and Richard C. Dubes, *Algorithms for clustering data*, Prentice Hall, 1988.

[JDM00] Anil K. Jain, Robert P. W. Duin, and Jianchang Mao, *Statistical pattern recognition: A review*, IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI) **22** (2000), no. 1, 4 – 37.

[JKS14] Ragesh Jaiswal, Amit Kumar, and Sandeep Sen, *A Simple D^2-Sampling Based PTAS for k-Means and Other Clustering Problems*, Algorithmica **70** (2014), no. 1, 22 – 46.

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

[JMF99] Anil K. Jain, M. Narasimha Murty, and Patrick J. Flynn, *Data clustering: A review*, ACM Computing Surveys **31** (1999), no. 3, 264 – 323.

[JMS02] Kamal Jain, Mohammad Mahdian, and Amin Saberi, *A new greedy approach for facility location problems*, Proceedings of the 34th ACM Symposium on the Theory of Computing (STOC), 2002, pp. 731 – 740.

[JV01] 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 **48** (2001), no. 2, 274 – 296.

[JW13] T. S. Jayram and David P. Woodruff, *Optimal bounds for johnson-lindenstrauss transforms and streaming problems with subconstant error*, ACM Transactions on Algorithms **9** (2013), no. 3, article 26.

[KH79a] Oded Kariv and Seifollah Louis Hakimi, *An algorithmic approach to network location problems. part i: The p-centers*, SIAM Journal on Applied Mathematics **37** (1979), no. 3, 513 – 538.

[KH79b] ———, *An algorithmic approach to network location problems. part ii: The p-medians*, SIAM Journal on Applied Mathematics **37** (1979), no. 3, 539 – 560.

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

[KMN11] Daniel M. Kane, Raghu Meka, and Jelani Nelson, *Almost optimal explicit johnson-lindenstrauss transformations*, Proceedings of the 15th International Workshop on Randomization and Computation (RANDOM), 2011, pp. 628 – 639.

[KMY03a] Piyush Kumar, Joseph S. B. Mitchell, and E. Alper Yıldırım, *Approximate minimum enclosing balls in high dimensions using core-sets*, ACM Journal of Experimental Algorithmics **8** (2003), article 1.1, 1–29.

[KMY03b] ———, *Computing core-sets and approximate smallest enclosing hyperspheres in high dimensions*, Proceedings of the 5th Workshop on Algorithm Engineering and Experiments (ALENEX), 2003, pp. 45 – 55.

[KN10] Daniel M. Kane and Jelani Nelson, *A derandomized sparse johnson-lindenstrauss transform*, CoRR **abs/1006.3585** (v3 in 2010), accessed: 2015-01-15.

[KN12] ———, *Sparser johnson-lindenstrauss transforms*, Proceedings of the 23rd ACM-SIAM Symposium on Discrete Algorithms (SODA), 2012, pp. 1195 – 1206.

[KNW10] Daniel M. Kane, Jelani Nelson, and David P. Woodruff, *An optimal algorithm for the distinct elements problem*, Proceedings of the 29th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS), 2010, pp. 41 – 52.

[KP05a] Hans-Peter Kriegel and Martin Pfeifle, *Density-based clustering of uncertain data*, Proceedings of the 11th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), 2005, pp. 672 –677.

[KP05b] ———, *Hierarchical density-based clustering of uncertain data*, Proceedings of the 5th IEEE International Conference on Data Mining (ICDM), 2005, pp. 689 – 692.

[KR07] Stavros G. Kolliopoulos and Satish Rao, *A nearly linear-time approximation scheme for the euclidean k-median problem*, SIAM Journal on Computing **37** (2007), no. 3, 757 – 782.

[KSS10] Amit Kumar, Yogish Sabharwal, and Sandeep Sen, *Linear-time approximation schemes for clustering problems in any dimensions*, Journal of the ACM **57** (2010), no. 2, 5:1 – 5:32.

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

[Len11] James Lennox, *Aristotle's biology*, The Stanford Encyclopedia of Philosophy, Fall 2011 Edition (Edward N. Zalta, ed.), Fall 2011 ed., 2011, `http://plato.stanford.edu/archives/fall2011/entries/aristotle-biology/`, accessed: 2014-02-09.

[Llo57] Stuart P. Lloyd, *Least squares quantization in PCM*, Bell Laboratories Technical Memorandum (1957), later published as [Llo82].

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

[LLS01] Yi Li, Philip M. Long, and Aravind Srinivasan, *Improved bounds on the sample complexity of learning*, Journal of Computer and System Sciences (JCSS) **62** (2001), no. 3, 516–527.

[LN04] James R. Lee and Assaf Naor, *Embedding the diamond graph in l_p and dimension reduction in l_1*, Geometric & Functional Analysis (GAFA) **14** (2004), no. 4, 745 – 747.

[LN14] Kasper G. Larsen and Jelani Nelson, *The Johnson-Lindenstrauss lemma is optimal for linear dimensionality reduction*, CoRR **abs/1411.2404** (v1 in 2014), accessed: 2015-04-29.

[Low04] David G. Lowe, *Distinctive image features from scale-invariant keypoints*, International Journal of Computer Vision **60** (2004), no. 2, 91 – 110.

[LS10] Michael Langberg and Leonard J. Schulman, *Universal epsilon-approximators for integrals*, Proceedings of the 21th ACM-SIAM Symposium on Discrete Algorithms (SODA), 2010, pp. 598–607.

[LS13] Shi Li and Ola Svensson, *Approximating k-median via pseudo-approximation*, Proceedings of the 45th ACM Symposium on the Theory of Computing (STOC), 2013, pp. 901 – 910.

[LSS12] Christiane Lammersen, Melanie Schmidt, and Christian Sohler, *Probabilistic k-median clustering in data streams*, Proceedings of the 10th Workshop on Approximation and Online Algorithms (WAOA), 2012, pp. 70 – 81.

[LSS14] ———, *Probabilistic k-median clustering in data streams*, Theory of Computing Systems (2014), 1–40.

[LV92] Jyh-Han Lin and Jeffrey Scott Vitter, *ϵ-approximations with minimum packing constraint violation*, Proceedings of the 24th ACM Symposium on the Theory of Computing (STOC), 1992, pp. 771 – 782.

[Mac67] James B. MacQueen, *Some methods for classification and analysis of multivariate observations*, Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability, 1967, pp. 281– 297.

[Mat00] Jiří Matoušek, *On approximate geometric k-clustering*, Discrete & Computational Geometry **24** (2000), no. 1, 61 – 84.

[Mat02] ———, *Lectures on discrete geometry*, Springer, 2002.

[Mat05] ———, *Embedding finite metric spaces into normed spaces*, Revised version of chapter 15 in [Mat02] available at `http://kam.mff.cuni.cz/~matousek/dg-nmetr.ps.gz`, 2005, accessed: 2014-02-27.

[Mat08] ———, *On variants of the johnson-lindenstrauss lemma*, Random Structures and Algorithms **33** (2008), no. 2, 142 – 156.

[Meh84] Kurt Mehlhorn, *Data structures and algorithms 3: Multi-dimensional searching and computational geometry*, Monographs in Theoretical Computer Science. An EATCS Series, vol. 3, Springer, 1984.

[Mel73] Z. A. Melzak, *Companion to concrete mathematics*, Wiley, 1973.

[MN90] Kurt Mehlhorn and S. Näher, *Dynamic fractional cascading*, Algorithmica **5** (1990), no. 2, 215–241.

[MNV09] Meena Mahajan, Prajakta Nimbhorkar, and Kasturi R. Varadarajan, *The Planar k-means Problem is NP-Hard*, Proceedings of the 3rd Workshop on Algorithms and Computation (WALCOM), 2009, pp. 274 – 285.

[MOP01] Nina Mishra, Daniel Oblinger, and Leonard Pitt, *Sublinear time approximate clustering*, Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms (SODA), 2001, pp. 439 – 447.

[MOP04] Adam Meyerson, Liadan O'Callaghan, and Serge A. Plotkin, *A* k*-Median Algorithm with Running Time Independent of Data Size*, Machine Learning **56** (2004), no. 1–3, 61–87.

[MP04] Ramgopal R. Mettu and C. Greg Plaxton, *Optimal time bounds for approximate clustering*, Machine Learning **56** (2004), no. 1 –3, 35 – 60.

[MS84] Nimrod Megiddo and Kenneth J. Supowit, *On the complexity of some common geometric location problems*, SIAM Journal on Computing **13** (1984), no. 1, 182–196.

[MS06] Keith E. Muller and Paul W. Steward, *Linear model theory: Univariate, multivariate, and mixed models*, first ed., Wiley Interscience, 2006.

[MU05] Michael Mitzenmacher and Eli Upfal, *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*, Cambridge University Press, 2005.

[Mut05] S. Muthukrishnan, *Data streams: Algorithms and applications*, Foundations and Trends in Theoretical Computer Science **1** (2005), no. 2, 117 – 236.

[NH02] Raymond T. Ng and Jiawei Han, *Clarans: A method for clustering objects for spatial data mining*, IEEE Transactions on Knowledge and Data Engineering **14** (2002), no. 5, 1003–1016.

[NKC+06] Wang Kay Ngai, Ben Kao, Chun Kit Chui, Reynold Cheng, Michael Chau, and Kevin Y. Yip, *Efficient clustering of uncertain data*, Proceedings of the 6th IEEE International Conference on Data Mining (ICDM), 2006, pp. 436 – 445.

[OMM+02] Liadan O'Callaghan, Adam Meyerson, Rajeev Motwani, Nina Mishra, and Sudipto Guha, *Streaming-data algorithms for high-quality clustering*, Proceedings of the 18th IEEE International Conference on Data Engineering, 2002, pp. 685 – 694.

[OR02] Rafail Ostrovsky and Yuval Rabani, *Polynomial-time approximation schemes for geometric min-sum median clustering*, Journal of the ACM **49** (2002), no. 2, 139 – 156.

[Orl93] James B. Orlin, *A faster strongly polynomial minimum cost flow algorithm*, Operations Research **41** (1993), 338 – 350.

[Ove83] Mark H. Overmars, *The design of dynamic data structures*, Lecture Notes in Computer Science, vol. 156, 1983.

[Pap81] Christos H. Papadimitriou, *Worst-Case and Probabilistic Analysis of a Geometric Location Problem*, Siam Journal on Computing **10** (1981), no. 3, 542 – 557.

[Pol84] David Pollard, *Convergence of stochastic processes*, Springer, 1984.

[PR04] Rasmus Pagh and Flemming Friche Rodler, *Cuckoo hashing*, Journal of Algorithms **51** (2004), no. 2, 122–144.

[Rom92] Steven Roman, *Advanced linear algebra*, 1st ed., Springer, 1992.

[rPG83] Jørgen Pedersen Gram, *Ueber die Entwicklung reeler Functionen in Reihen mittelst der Methode der kleinsten Quadrate*, Journal für die reine und angewandte Mathematik (1883), 41 – 73.

[RTG00] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas, *The earth mover's distance as a metric for image retrieval*, International Journal of Computer Vision **40** (2000), no. 2, 99–121.

[Sam05] Hanan Samet, *Foundations of multidimensional and metric data structures*, Morgan Kaufmann Publishers Inc., 2005.

[Sch07] Erhard Schmidt, *Zur Theorie der linearen und nichtlinearen Integralgleichungen. I. Teil: Entwicklung willkürlicher Funktionen nach Systemen vorgeschriebener*, Mathematische Annalen (1907), no. 63, 433 – 476.

[Sie89] Alan Siegel, *On universal classes of fast high performance hash functions, their time-space tradeoff, and their applications*, Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 1989, pp. 20 – 25.

[SR10] Mingjun Song and Sanguthevar Rajasekaran, *Fast algorithms for constant approximation k-means clustering*, Transactions on Machine Learning and Data Mining **3** (2010), no. 2, 67 – 79.

[STC04] John Shawe-Taylor and Nello Cristianini, *Kernel methods for pattern analysis*, Cambridge University Press, 2004.

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

[Ste93] Gilbert W. Stewart, *On the early history of the singular value decomposition*, SIAM Review **35** (1993), 551 – 566.

[SV12] Nariankadu D. Shyamalkumar and Kasturi R. Varadarajan, *Efficient subspace approximation algorithms*, Discrete & Computational Geometry **47** (2012), no. 1, 44–63.

[TWH01] Robert Tibshirani, Guenther Walther, and Trevor Hastie, *Estimating the number of clusters in a dataset via the gap statistic*, Journal of the Royal Statistical Society: Series B (Statistical Methodology) **63** (2001), 411 – 423.

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

[VX12a] Kasturi R. Varadarajan and Xin Xiao, *A near-linear algorithm for projective clustering integer points*, Proceedings of the 23rd ACM-SIAM Symposium on Discrete Algorithms (SODA), 2012, pp. 1329 – 1342.

[VX12b] ———, *On the sensitivity of shape fitting problems*, Proceedings of the 32nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS), 2012, pp. 486–497.

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

[Whi68] Alan J. White, *Real analysis: an introduction*, Addison-Wesley, 1968.

[Wik] Wikipedia, *Cluster analysis*, `http://en.wikipedia.org/wiki/Cluster_analysis`, accessed: 2014-02-10.

[WKQ+08] Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus F. M. Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, and Dan Steinberg, *Top 10 algorithms in data mining*, Knowledge and Information Systems **14** (2008), no. 1, 1 – 37.

[Woo04] David P. Woodruff, *Optimal space lower bounds for all frequency moments*, Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms (SODA), 2004, pp. 167 – 175.

[XL08] Huajie Xu and Guohui Li, *Density-based probabilistic clustering of uncertain data*, Proceedings of the 1st International Conference on Computer Science and Software Engineering (CSSE), vol. 4, 2008, pp. 474 – 477.

[XW05] Rui Xu and Donald C. Wunsch II, *Survey of clustering algorithms*, IEEE Transactions on Neural Networks **16** (2005), no. 3, 645 – 678.

[Yıl08] E. Alper Yıldırım, *Two algorithms for the minimum enclosing ball problem*, SIAM Journal on Optimization **19** (2008), no. 3, 1368 – 1391.

[YZ03] Yingyu Ye and Jiawei Zhang, *An improved algorithm for approximating the radii of point sets*, Proceedings of the 6th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX), 2003, pp. 178 – 187.

[ZRL97a] Tian Zhang, Raghu Ramakrishnan, and Miron Livny, *BIRCH: A New Data Clustering Algorithm and Its Applications* , Data Mining and Knowledge Discovery **1** (1997), no. 2, 141 – 182.

[ZRL97b] ———, *Implementation of BIRCH*, `http://pages.cs.wisc.edu/~vganti/birchcode/`, 1997, accessed: 2014-04-13.