Approximation Techniques for Facility Location and Their Applications in Metric Embeddings

Dissertation

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

der Technischen Universität Dortmund

an der Fakultät für Informatik

von

Christiane Lammersen

Dortmund

2010

Tag der mündlichen Prüfung:	7. Dezember 2010
Dekan:	Prof. Dr. Peter Buchholz
Gutachter:	Prof. Dr. Christian Sohler,
	Prof. Dr. Friedhelm Meyer auf der Heide

Abstract

This thesis addresses the development of geometric approximation algorithms for huge datasets and is subdivided into two parts. The first part deals with algorithms for facility location problems, and the second part is concerned with the problem of computing compact representations of finite metric spaces.

Facility location problems belong to the most studied problems in combinatorial optimization and operations research. In the facility location variants considered in this thesis, the input consists of a set of points where each point is a client as well as a potential location for a facility. Each client has to be served by a facility. However, connecting a client incurs connection costs, and opening or maintaining a facility causes so-called opening costs. The goal is to open a subset of the input points as facilities such that the total cost of the system is minimized.

We are particularly interested in facility location problems for large-scale distributed systems of mobile objects. In order to be able to analyze such complex systems, we examine the following partial aspects:

- At first, we present a distributed algorithm that, in case of uniform opening costs for the facilities and uniform demands of the clients, computes in only three communication rounds a constant-factor approximation for the metric facility location problem.

- In Chapter 4, we introduce a mobile facility location problem where the input points move continuously in a constant-dimensional Euclidean space. In contrast to Chapter 3, we also take non-uniform opening costs for the facilities and non-uniform demands of the clients into account. We propose an event-driven data structure that efficiently maintains a subset of the mobile points as open facilities such that, at any time, the total cost of the system is at most a constant factor larger than the optimal facility location cost.

- In Chapter 5, we consider again a uniform facility location problem. However, this time, we develop a streaming algorithm where the input stream consists of insert and delete operations of points from a constant-dimensional Euclidean space. While reading the input stream, our algorithm maintains a summary of the current point set in a subtle way with the result that the required space is polylogarithmic in the size of the input stream and, at any time, it can output a constant-factor approximation of the optimal facility location cost.

- In the next chapter, we give an efficient streaming implementation of a k-means clustering algorithm. The k-means clustering problem is closely related to the facility

location problem. The goal is to place k facilities, the so-called cluster centers, such that the sum of the squared distances of the points to their nearest cluster center is minimized. Our algorithm is based on a coreset construction. A coreset is a small weighted point set that approximates the input point set with respect to the k-means clustering problem.

In the second part of this thesis, we study compact representations of finite metric spaces. Our representations have the property that a large fraction of all the pairwise distances between the points is almost preserved and only a small fraction of all the pairwise distances, the so-called slack, can be arbitrarily distorted. Constructions of such representations are an important tool in the analysis of huge datasets.

In Chapter 7, we apply some space-partitioning techniques from Chapter 5 to construct well-separated pair decompositions with slack for low-dimensional Euclidean spaces. We also show how to transfer this approach to doubling metric spaces.

Afterwards, we extend our techniques to obtain streaming algorithms that compute embeddings with a distortion of at most $1 + \varepsilon$ and with low slack for high-dimensional Euclidean spaces and doubling metric spaces. Furthermore, we investigate embeddings with low distortion and low slack for general metric spaces given as a data stream of points. Besides, we show how to use embedding techniques to get a $(1 \pm \varepsilon)$-approximation algorithm for the high-dimensional Euclidean max-cut problem where the input stream consists of insert and delete operations of points. All of our streaming algorithms need only space that is polylogarithmic in the size of the input stream.

Zusammenfassung

Diese Dissertation beschäftigt sich mit der Entwicklung von geometrischen Approximationsalgorithmen für große Datenmengen und ist in zwei Teile aufgeteilt. Der erste Teil behandelt Algorithmen für verschiedene Arten von Facility-Location-Problemen und der zweite Teil Konstruktionen von kompakten Darstellungen endlicher metrischer Räume.

Facility-Location-Probleme gehören zu den am meisten untersuchten Problemen in der kombinatorischen Optimierung und Operations Research. In den von uns betrachteten Facility-Location-Varianten erhält man als Eingabe eine Menge von Punkten, die sowohl Standorte von Kunden als auch mögliche Standorte von Facilities darstellen. Jeder Kunde soll durch eine Facility versorgt werden. Hierbei fallen für die Kunden Verbindungskosten und für das Öffnen bzw. Aufrechterhalten der genutzten Facilities sogenannte Öffnungskosten an. Das Ziel ist es, eine Teilmenge der möglichen Facilities zu öffnen, so dass die Gesamtkosten minimiert werden.

Wir interessieren uns bei den Facility-Location-Problemen insbesondere für riesige verteilte Systeme mobiler Standorte. Um solch komplexe Systeme untersuchen zu können, haben wir im Einzelnen die folgenden Teilaspekte genauer betrachtet:

- Als erstes stellen wir einen verteilten Algorithmus vor, der im Fall von einheitlichen Öffnungskosten für die Facilities als auch einheitlichen Bedarf der Kunden in nur drei Kommunikationsrunden eine konstante Approximation für das metrische Facility-Location-Problem ausgibt.

- In Kapitel 4 führen wir ein mobiles Facility-Location-Problem ein, bei dem sich die Eingabepunkte kontinuierlich in einem niedrig-dimensionalen euklidischen Raum bewegen. Im Unterschied zu Kapitel 3 berücksichtigen wir diesmal auch uneinheitliche Öffnungskosten für die Facilities und uneinheitlichen Bedarf der Kunden. Wir geben eine ereignisgesteuert Datenstruktur an, die effizient eine Teilmenge der sich bewegenden Punkte als geöffnete Facilties aufrechterhält, so dass zu jeder Zeit die Gesamtkosten des Systems höchstens um einen konstanten Faktor größer sind als die optimalen Gesamtkosten.

- In Kapitel 5 betrachten wir wieder ein uniformes Facility-Location-Problem. Diesmal entwickeln wir jedoch einen Algorithmus der Datenströme bearbeiten kann, die aus Einfüge- und Löschoperationen von Punkten in einem niedrig-dimensionalen euklidischen Raum bestehen. Während des Einlesens hält unser Datenstromalgorithmus geschickt eine Zusammenfassung der aktuellen Punktmenge aufrecht, so dass sein verwendeter Speicherplatz polylogarithmisch in der Größe des Eingabestromes ist und zu jeder Zeit eine konstante Approximation der optimalen Gesamtkosten für das Facility-Location-Problem ausgegeben werden kann.

- Im nächsten Kapitel geben wir eine effiziente Datenstromimplementierung eines k-Means-Clustering-Algorithmus an. Das k-Means-Clustering-Problem ist verwandt mit dem Facility-Location-Problem. Dabei sollen k Clusterzentren so platziert werden, dass die quadrierten Abstände der Punkte zu dem jeweils nächstliegenden Clusterzentrum minimiert werden. Unser Algorithmus basiert auf einer neuen Kernmengenkonstruktion. Eine Kernmenge ist eine kleine gewichtete Punktmenge, die die Eingabepunktmenge gemäß des k-Means-Clustering-Problems approximiert.

Im zweiten Teil der Dissertation beschäftigen wir uns mit kompakten Repräsentationen endlicher metrischer Räume. Unsere Repräsentationen haben die Eigenschaft, dass sie einen großen Anteil der paarweisen Distanzen gut erhalten und nur einen kleinen Anteil, den sogenannten Schlupf, beliebig verzerren. Konstruktionen solcher Repräsentationen bilden ein wichtiges Werkzeug bei der Analyse von großen Datenmengen.

In Kapitel 7 wenden wir einige Raumaufteilungstechniken aus Kapitel 5 an, um wohlseparierte Paar-Dekompositionen mit Schlupf für niedrig-dimensionale euklidische Räume zu konstruieren. Wir zeigen außerdem wie dieser Ansatz auf Doubling-Metriken übertragen werden kann.

Anschließend erweitern wir unsere Techniken, um Datenstromalgorithmen zu erhalten, die Einbettungen mit einer Verzerrung von höchstens $1 + \varepsilon$ und geringem Schlupf von hoch-dimensionalen euklidischen Räumen und Doubling-Metriken berechnen. Des Weiteren untersuchen wir Einbettungen mit geringer Verzerrung und geringem Schlupf für allgemeine Metriken, die als Datenstrom von Punkten gegeben sind. Außerdem zeigen wir, dass man mit Hilfe von Einbettungstechniken einen $(1 \pm \varepsilon)$-Approximationsalgorithmus für das hoch-dimensionale euklidische Max-Cut-Problem erhalten kann, wobei der Eingabestrom aus Einfüge- und Löschoperationen von Punkten besteht. All unsere Datenstromalgorithmen benötigen Speicherplatz, der nur polylogarithmisch in der Größe des jeweiligen Eingabestromes ist.

Acknowledgments

First and foremost, I would like to thank my advisor, Prof. Dr. Christian Sohler, for giving me the opportunity to work with him and under his supervision. I benefited in many ways from his great support. He integrated me into important research communities at an early stage. Even before I started my PhD studies, he offered me to attend a summer school on data stream algorithms and a Dagstuhl seminar on sublinear algorithms. During the whole time, his guidance was invaluable for me. Whenever I got stuck with a problem, I felt free to ask for his advice, which always ended up in new helpful ideas. It was also of great importance to me that he kept faith in my skills. Even when I thought that I would not be able to cope with a challenge, my advisor encouraged me to try it and it always worked out. In a nutshell, I enjoyed it a lot to work with him!

Special thanks also go to my co-advisor, Prof. Dr. Friedhelm Meyer auf der Heide. His comments and questions during seminar talks were really helpful to improve my thesis. Besides, I am very pleased that he welcomed me so heartily each time when I visited his research group in Paderborn.

The results in this thesis have been emerged from collaborations with many smart people. I would like to express my best thanks to my co-authors Dr. Marcel Ackermann, Dr. Bastian Degener, Joachim Gehweiler, Marcus Märtens, Christoph Raupach, Dr. Anastasios Sidiropoulos, Prof. Dr. Christian Sohler, and Kamil Swierkot. It was a pleasure for me to collaborate with all of them.

During my PhD studies, I have been research assistant at Universität Paderborn, Universität Bonn, and Technische Universität Dortmund. This gave me the opportunity to become acquainted with many nice people and to make new friends. Particularly, I would like to name Dr. Bastian Degener, Dominic Dumrauf, and Joachim Gehweiler. For being an amiable three-year office mate and friend, I would like to thank Dr. Morteza Monemizadeh. His amazing knowledge of research results in the area of clustering and data stream algorithms has been very useful for me. Furthermore, I am more than happy that I have found such a sympathetic and caring friend in Melanie Schmidt. The pleasant conversations with her always cheered me up. For many fruitful discussions and a nice working atmosphere, I would also like to thank Dr. Mohammad Ali Abam, Florian Berger, Antje Bertram, Prof. Dr. Beate Bollig, Dr. Olaf Bonorden, Dr. Gereon Frahling, Alexander Gilbers, Dr. André Gronemeier, Frank Hellweg, Prof. Dr. Rolf Klein, Mariele Knepper, Renate Kühn, Dr. Elmar Langetepe, Dr. Mario Mense, Rainer Penninger, Melanie Schmidt, Dr. Dirk Sudholt, Dr. Christian Thyssen, Tim Suess, Heinz-Georg Wassing, and Christine Zarges.

I would like to thank Prof. Dr. Stefano Leonardi for inviting me to a research visit at Sapienza University of Rome, Prof. Dr. Piotr Indyk for helpful discussions at the

MADALGO Summer School on Data Stream Algorithms, and Prof. Dr. Sumit Ganguly for inviting me to the IITK Workshop on Algorithms for Processing Massive Data Sets.

Many thanks go to Dr. Mariano Zelke for carefully proof-reading parts of my thesis and for giving helpful suggestions to improve the readability of my thesis.

Certainly, I would have never managed to get so far without the support and encouragement of my friends and family. Particularly, I would like to thank my brother Markus who sparked my interest in computer science, my parents, Brunhilde and Klemens, whom I own the most of what I am today, and Thomas Friebe whom I can rely on in any situation and who enriches my life in such a wonderful way.

Contents

Notation and Terminology

This section provides an overview of the notation and terminology for mathematical basics used throughout this thesis. More special notions are introduced gradually in the main chapters. For ease of reading, some of the definitions are even repeated a few times.

$\emptyset$	empty set
$\mathbb{N}$	set of the natural numbers $\{1, 2, 3, \ldots\}$
$\mathbb{N}_0$	set of the natural numbers including 0
$[n]$	set $\{0, 1, \ldots, n-1\}$
$\mathbb{R}$	set of the reals
$\mathbb{R}_{\geq 0}$	set of the non-negative reals
$[a, b]$	closed interval of the reals x with $a \leq x \leq b$
$(a \pm \varepsilon)$	closed interval of the reals x with $a - \varepsilon \leq x \leq a + \varepsilon$
(a, b)	open interval of the reals x with $a < x < b$
$\lvert A \rvert$	cardinality of set A
$A \cup B$	union of sets A and B, i.e., $\{x \mid x \in A \text{ or } x \in B\}$
$A \cap B$	intersection of sets A and B, i.e., $\{x \mid x \in A \text{ and } x \in B\}$
$A \backslash B$	difference set A minus B, i.e., $\{x \mid x \in A \text{ and } x \notin B\}$
$A \times B$	Cartesian product of sets A and B, i.e., $\{(x, y) \mid x \in A \text{ and } y \in B\}$
A^n	set of n-dimensional column vectors with entries from set A
$A^{n \times m}$	set of $(n \times m)$-matrices with entries from set A
$M = (X, \mathrm{D})$	metric space M, where X is a non-empty set of elements and $\mathrm{D} : X \times X \to \mathbb{R}_{\geq 0}$ is a distance function defined on X
$\mathrm{D}(x, y)$	distance between x and y in some metric space
$\mathrm{diam}(X)$	diameter of set X in some metric space
$\mathcal{B}(x, r)$	closed ball of radius r centered at point $x \in X$ in some metric space $M = (X, \mathrm{D})$, i.e., the set $\{y \in X \mid \mathrm{D}(x, y) \leq r\}$
spread of M	ratio of farthest pair distance in X to closest pair distance in X for some finite metric space $M = (X, \mathrm{D})$ with $\mathrm{D}(x, y) \neq 0$ for all pairs $(x, y) \in X \times X$, $x \neq y$
$\mathbb{R}^d$	the d-dimensional Euclidean space
v^{T}	transpose of vector v with entries from $\mathbb{R}$
$v^{\mathrm{T}} \cdot w$	scalar product of column vectors $v, w \in \mathbb{R}^d$
$\lVert v \rVert$	Euclidean norm of column vector $v \in \mathbb{R}^d$

$G = (V, E)$	graph G with vertex set V and edge set E
e	Euler's number $2.7182\ldots$
$\exp(x)$	Euler's number to the power of x, i.e., the value e^x
$\ln(x)$	natural logarithm of x, i.e., logarithm of x to the base e
$\log_b(x)$	logarithm of x to the base b
$\log_b^k(x)$	$\log_b(x)$ to the power of k, i.e., $(\log_b(x))^k$
$\log(x)$	binary logarithm of x, i.e., logarithm of x to the base 2
$\mathcal{O}(g)$	$\{f : \mathbb{N} \to \mathbb{R}_{\geq 0} \mid \exists n_0 \in \mathbb{N}\, \exists c > 0\, \forall n \geq n_0 : f(n) \leq c \cdot g(n)\}$
$\Omega(g)$	$\{f : \mathbb{N} \to \mathbb{R}_{\geq 0} \mid \exists n_0 \in \mathbb{N}\, \exists c > 0\, \forall n \geq n_0 : 0 \leq c \cdot g(n) \leq f(n)\}$
$\Theta(g)$	$\{f : \mathbb{N} \to \mathbb{R}_{\geq 0} \mid f \in \mathcal{O}(g) \text{ and } f \in \Omega(g)\}$
$o(g)$	$\{f : \mathbb{N} \to \mathbb{R}_{\geq 0} \mid \exists n_0 \in \mathbb{N}\, \exists c > 0\, \forall n \geq n_0 : f(n) < c \cdot g(n)\}$
$\omega(g)$	$\{f : \mathbb{N} \to \mathbb{R}_{\geq 0} \mid \exists n_0 \in \mathbb{N}\, \exists c > 0\, \forall n \geq n_0 : 0 \leq c \cdot g(n) < f(n)\}$
$\text{poly}(g)$	$\{f : \mathbb{N} \to \mathbb{R}_{\geq 0} \mid \exists n_0 \in \mathbb{N}\, \exists c \geq 1\, \forall n \geq n_0 : f(n) \leq (g(n))^c\}$
$\text{polylog}(g)$	$\{f : \mathbb{N} \to \mathbb{R}_{\geq 0} \mid \exists n_0 \in \mathbb{N}\, \exists c \geq 1\, \forall n \geq n_0 : f(n) \leq \log^c(g(n))\}$
$\widetilde{\mathcal{O}}(g)$	$\{f : \mathbb{N} \to \mathbb{R}_{\geq 0} \mid f \in \mathcal{O}(g \cdot \text{polylog}(g))\}$
g^H	$\{g^h \mid h \in H\}$ for some $g : \mathbb{N} \to \mathbb{R}_{\geq 0}$ and $H \subset \{f \mid f : \mathbb{N} \to \mathbb{R}_{\geq 0}\}$
$g \cdot H$	$\{g \cdot h \mid h \in H\}$ for some $g : \mathbb{N} \to \mathbb{R}_{\geq 0}$ and $H \subset \{f \mid f : \mathbb{N} \to \mathbb{R}_{\geq 0}\}$
$g + H$	$\{g + h \mid h \in H\}$ for some $g : \mathbb{N} \to \mathbb{R}_{\geq 0}$ and $H \subset \{f \mid f : \mathbb{N} \to \mathbb{R}_{\geq 0}\}$
$\min_{x \in A} f(x)$	value $f(x)$ with $x \in A$ and $f(x) \leq f(y)$ for all $y \in A$
$\max_{x \in A} f(x)$	value $f(x)$ with $x \in A$ and $f(x) \geq f(y)$ for all $y \in A$
$\lceil x \rceil$	smallest integer n with $n \geq x$
$\lfloor x \rfloor$	largest integer n with $n \leq x$
$n!$	factorial of the natural number n, i.e., the value $n \cdot (n-1) \cdot \ldots \cdot 2 \cdot 1$
$\binom{n}{k}$	binomial coefficient n over k, i.e., the value $n!/((n-k)! \cdot k!)$
$\mathbf{Pr}[A]$	probability of the event A
$\mathbf{E}[Z]$	expectation of the random variable Z
$\mathbf{V}[Z]$	variance of the random variable Z

1 Introduction

Facility location problems belong to the most studied problems in operations research and combinatorial optimization. In its classical interpretation, the goal of facility location is to find optimal places for industrial facilities (e.g., restaurants, factories, or supermarkets) such that a combination of the building and maintenance costs for the facilities and the transportation costs for the clients is minimized. However, facility location problems have also applications in many other scenarios. As a result, various types of facility location problems have been investigated until today.

In the facility location variants considered in this thesis, the input consists of a set of points where each point is a client as well as a potential location for a facility. Each client has to be served by a facility. Here, it must be taken into account that, on the one hand, serving a client incurs *connection costs* and, on the other hand, opening or maintaining a facility causes so-called *opening costs*. The goal is to open a subset of the input points as facilities such that the total cost of the system is minimized.

In general, each facility has its individual opening cost, and the connection cost of a client depends proportionally on its individual *demand* as well as on its distance to the nearest open facility. This means, of course, there has to be a distance measure defined on the input points. Obviously, one typical scenario is that the points are from a Euclidean space and the distance measure between points is given by the Euclidean distance. However, other distance measures are also conceivable. In radio networks, for instance, it could be interesting to consider powered Euclidean distances since the energy required for transmitting a message via a certain distance is somewhere between the square and the cube of the distance.

We are particularly interested in facility location problems for large-scale *distributed systems* of mobile objects. In such a system, each object is an autonomous computational entity that has its own local memory and that communicates with the other entities by message passing. Since such systems are too complex to analyze them in their entirety, we examine several partial aspects of them. Applications of our scenario are, for example, in mobile ad-hoc and sensor networks. In these networks, nodes move continuously and interact with each other. Often, they are organized in a hierarchical way where the upper layer offers the lower layer a certain service.

Furthermore, we are interested in designing algorithms that are capable of *clustering* huge Euclidean point sets efficiently. As clustering objective, we focus on k-means. Note that the k-means clustering problem is closely related to the facility location problem, which itself belongs to the clustering problems as well. In general, the goal of a clustering is to partition a set of given objects into subsets, the so-called *clusters*, such that objects from the same cluster are similar to each other and objects from different clusters are

dissimilar. In the k-means clustering problem, the input is a set of points with a distance measure defined on them, and the goal is to place k *cluster centers* such that the sum of the squared distances of the points to their nearest cluster center is minimized. For each cluster center, there exists one cluster containing all the points that are closer to this cluster center than to all the other cluster centers. One application of clustering is the compact representation of huge datasets. For instance, we could map each data item to a point in a Euclidean space and, after having clustered the resulting point set, represent each cluster by its cluster center.

The second part of this thesis concentrates exclusively on compact representations of huge n-point metric spaces. An n-point metric space is a pair $M = (X, \mathrm{D})$ where X is a set of n points and D is a distance measure defined on X that is non-negative, symmetric, and satisfies the triangle inequality. Our goal is to compute a compact representation of M that fairly captures the pairwise distances of M but is structurally simpler than M and uses only sublinear space. To measure the quality of such a representation, we use the notion of *low-distortion embeddings with slack*, which is defined as follows. An embedding from a metric space $M = (X, \mathrm{D})$ into a target metric space $M' = (X', \mathrm{D}')$ is a mapping $\varphi : X \to X'$. We say φ *contracts* the distance between two points x and y in X by a factor of $\alpha \geq 1$ if the embedded distance $\mathrm{D}'(\varphi(x), \varphi(y))$ of x and y is α-times shorter than the original distance $\mathrm{D}(x, y)$. Similarly, we say that φ *expands* the distance between x and y by a factor of $\beta \geq 1$ if the embedded distance $\mathrm{D}'(\varphi(x), \varphi(y))$ of x and y is β-times longer than the original distance $\mathrm{D}(x, y)$. Now, the *distortion* of φ is defined as the product of the maximum contraction and the maximum expansion of all the pairwise distances in X. Finally, we say that φ has distortion $\varrho \geq 1$ and *slack* σ with $0 < \sigma < 1$ if, for a $(1 - \sigma)$-fraction of all the pairwise distances in X, the distortion is ϱ. The remaining pairwise distances, i.e., the slack, can be arbitrarily distorted.

In this thesis, we study the problem of computing embeddings with low distortion and low slack of several n-point metric spaces that are given as a *data stream*. A data stream is a sequence of data items which can only be accessed in one sequential scan that reads the data items one by one. Besides, while reading and processing the data, an algorithm is only allowed to use space that is sublinear in the size of the input stream.

In the following section, we will give a detailed overview of the results presented in this thesis.

1.1 Outline and Main Results

Chapter 2

This chapter provides some preparation for the main chapters. We give formal definitions of the metric spaces and the facility location problems considered in this thesis. Afterwards, we present an existing facility location algorithm due to Mettu and Plaxton [87], which has played an important role in the design of two of our facility location algorithms. Furthermore, we introduce the computational models that have been used to develop our

algorithms and to analyze them in terms of their complexity. This includes the synchronous message passing model for algorithms working in a distributed setting, the kinetic data structure framework for algorithms working in a mobile setting, and data stream models.

Chapter 3

We begin our studies by investigating a special type of metric facility location problem in a distributed setting. In this problem, we assume that each point is a client as well as a potential location for a facility and that the opening costs for the facilities and the demands of the clients are uniform. We present a randomized distributed algorithm that computes with high constant probability a constant-factor approximation for this type of facility location problem. The algorithm uses three rounds of all-to-all communication with message sizes bounded to $\mathcal{O}(\log(n))$ bits, where n is the number of input points. In particular, we show how each point decides locally after the first communication round whether it opens a facility or not. The following two communication rounds are only required to connect the clients to their nearest open facility.

In the last part of Chapter 3, we extend our distributed algorithm to constant powers of metric spaces. Here, we also obtain a constant-factor approximation algorithm that uses three rounds of all-to-all communication with message sizes bounded to $\mathcal{O}(\log(n))$ bits.

The results of Chapter 3 have been previously published in [J. Gehweiler, C. Lammersen, and C. Sohler. A distributed $\mathcal{O}(1)$-approximation algorithm for the uniform facility location problem. In *Proceedings of the 18th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '06)*, pages 237–243. Association for Computing Machinery, 2006.].

Chapter 4

We reuse some essential ideas of Chapter 3 to approach a mobile facility location problem. This time, we take non-uniform opening costs for the facilities and non-uniform demands for the clients into account. We assume that each point moves along a known trajectory in a d-dimensional Euclidean space and that, at any time, each point is either an open facility or a client. The opening cost that arises for a facility persists during the entire time it is open. Analogously, a client has to pay some cost for its connection to an open facility permanently. This cost depends on the client's demand and its distance to the nearest open facility.

To approach the mobile facility location problem, we propose a deterministic kinetic data structure. This data structure maintains a subset of the moving points as open facilities such that, at any time, the sum of the opening cost for the open facilities and the connection cost for the clients is at most a constant factor larger than the current optimal cost. The space requirement of our data structure is $\mathcal{O}(n(\log^d(n) + \log(nR)))$, where n denotes the number of input points and R is a value depending on the cost and demand values of the input points. In case that each trajectory can be described by a bounded degree polynomial, we process $\mathcal{O}(n^2 \log^2(nR))$ events, each requiring $\mathcal{O}(\log^{d+1}(n) \cdot \log(nR))$

time and $\mathcal{O}(\log(nR))$ status changes. To our knowledge, there had been no kinetic data structures for facility location proposed prior to the work presented in Chapter 4.

Chapter 4 is based on [B. Degener, J. Gehweiler, and C. Lammersen. The kinetic facility location problem. In *Proceedings of the 11th Scandinavian Workshop on Algorithm Theory (SWAT '08)*, pages 378–389. Springer, 2008.] and [B. Degener, J. Gehweiler, and C. Lammersen. Kinetic facility location. *Algorithmica*, 57(3):562–584, July 2010. By invitation for the special issue on selected papers from SWAT '08.].

Chapter 5

We continue our studies of facility location problems. Similar to Chapter 3, we consider a variant in which each input point is a client as well as a potential location for a facility and in which the opening costs for the facilities and the demands of the clients are uniform. However, this time, the input points are given as a dynamic geometric data stream. This means, the input is a sequence of insert and delete operations of points from a discrete Euclidean space $\{1, \ldots, \Delta\}^d$. We assume that the dimension d is a constant.

We present a randomized algorithm that computes a constant-factor approximation for the cost of the uniform facility location problem over dynamic geometric data streams. Our streaming algorithm processes an insertion or deletion of a point in time polylogarithmic in Δ, requires space polylogarithmic in Δ, and has an error probability of less than $1/3$. We remark that this error probability can be reduced by using a standard amplification technique.

The construction of our streaming algorithm is done in three steps. The first step is to define a certain partition of the input space and to relate this partition to the cost which are to be calculated. In particular, we show that if we assign to each cell in this partition a weight that corresponds to the number of points inside the cell times the side length of the cell, the sum of these weights is a constant-factor approximation for the facility location cost. In the next step, we propose a randomized algorithm that utilizes the existence of such a space partition but does not consider streaming. Finally, we explain how our randomized algorithm can be transferred to the dynamic geometric data stream model.

The results from Chapter 5 can be found in [C. Lammersen and C. Sohler. Facility location in dynamic geometric data streams. In *Proceedings of the 16th Annual European Symposium on Algorithms (ESA '08)*, pages 660–671. Springer, 2008.].

Chapter 6

This chapter deals with an efficient implementation of a k-means clustering algorithm for data streams, which we call StreamKM++. The k-means clustering problem is closely related to the facility location problem. The goal is to place k facilities, the so-called cluster centers, such that the sum of the squared distances of the points to their nearest cluster center is minimized. Our algorithm computes a small weighted coreset of the data stream that approximates the input point set with respect to the k-means clustering problem. The problem is then solved by running the k-means++ algorithm [9] on the coreset.

Algorithm StreamKM++ is based on two new techniques. First, we use an adaptive, non-uniform sampling approach similar to the k-Means++ seeding procedure to obtain small coresets from the data stream. This construction is rather easy to implement and, unlike other coreset constructions, its running time has only a small dependency on the dimensionality of the data. Second, we propose a new data structure, which we call *coreset tree*. The use of coreset trees significantly speeds up the time that is necessary for the adaptive, non-uniform sampling during our coreset construction.

To evaluate the performance of our algorithm, we compare it experimentally with two well-known streaming implementations: BIRCH [111] and StreamLS [52]. We show that if the first priority is the quality of the clustering, then StreamKM++ provides a good alternative to BIRCH and StreamLS. This applies particularly if the number of cluster centers is large. To show that the performance of our algorithm is competitive with classical non-streaming algorithms as well, we also compare it with the k-Means++ algorithm and Lloyd's algorithm [39, 80, 82] on some datasets with small or moderate size.

We end our investigation of problems related to facility location with Chapter 6.

Algorithm StreamKM++ together with its experimental study has been previously published in [M. R. Ackermann, C. Lammersen, M. Märtens, C. Raupach, C. Sohler, and K. Swierkot. StreamKM++: A clustering algorithm for data streams. In *Proceedings of the 12th Workshop on Algorithm Engineering and Experiments (ALENEX '10)*, pages 173–187. Society for Industrial and Applied Mathematics, 2010. Invited to the special issue on selected papers from ALENEX '10. Submitted to *ACM Journal on Experimental Algorithmics*.].

Chapter 7

In this chapter, we start to investigate another geometric problem. The problem under consideration is the computation of compact representations of finite metric spaces that capture the metric structure well. One option to tackle this problem is the construction of a well-separated pair decomposition (WSPD). An ε-WSPD of a point set P is a representation of P which gives the guarantee that all pairwise distances of P are $(1\pm\varepsilon)$-preserved, i.e., each pairwise distance is expanded by a factor of at most $(1+\varepsilon)$ or compressed by a factor of at most $1/(1-\varepsilon)$. In order to enable our representation to have size sublinear in $|P|$, we relax this condition and introduce the notion of a *WSPD with slack*. An ε-WSPD with slack σ for P is a representation of P such that at least a $(1-\sigma)$-fraction of all the pairwise distances of P are $(1\pm\varepsilon)$-preserved. The remaining pairwise distances of P can be arbitrarily distorted.

We show how to compute an ε-WSPD with slack σ for a set P consisting of n points from a low-dimensional Euclidean space $\mathbb{R}^d$. The space requirement for this compact representation is $\mathcal{O}(\log(\Delta)/(\varepsilon^d\sigma))$, where Δ is the spread of P. Recall that the spread of a finite metric space is defined by the ratio of the farthest pair distance to closest pair distance occurring in the metric space. The techniques used by our algorithm are also applicable to doubling metric spaces. For a doubling metric space with bounded dimension λ and

spread Δ, our algorithm computes an ε-WSPD with slack σ whose space requirement is $\mathcal{O}(\log^2(\Delta)/(\varepsilon^\lambda\sigma))$.

The results of Chapter 7 can be found in [C. Lammersen, A. Sidiropoulos, and C. Sohler. Streaming embeddings with slack. In *Proceedings of the 11th Algorithms and Data Structures Symposium (WADS '09)*, pages 483–494. Springer, 2009.].

Chapter 8

Chapter 8 addresses the computation of compact representations of finite metric spaces in data stream models. We present randomized streaming algorithms that, given a stream of n points from a metric space M, compute an embedding of M into an n-point metric space M' that has low distortion and low slack. Our algorithms use space polylogarithmic in n and Δ, where Δ is the spread of the metric space. Within such space limitations, it is impossible to store the embedding explicitly. We bypass this obstacle by computing a compact representation of M', without storing the actual mapping from M into M'.

Given a slack parameter σ and a precision parameter ε, our streaming algorithm computes for a set of points P from a high-dimensional Euclidean space a set of points P' from a low-dimensional Euclidean space such that P embeds into P' with distortion $1+\varepsilon$ and slack σ. The algorithm uses the techniques presented in Chapter 7. Furthermore, based on results obtained in Chapter 7, we show how to compute embeddings with distortion $1+\varepsilon$ and slack σ of metric spaces with bounded doubling dimension in the data stream model. For general metric spaces, we propose a streaming embedding of a metric space M into a metric space M' with distortion $\mathcal{O}(1)$ and slack σ. We complement our upper bounds by proving that embedding general metric spaces with distortion less than 2 and slack less than $1/4$ requires $\Omega(n/\log(n) + \log(\log(\Delta)))$ bits of memory.

Besides, we use an embedding to show that there is a randomized streaming algorithm that computes with high constant probability a $(1 \pm \varepsilon)$-approximation of the max-cut problem for a dynamic geometric data stream of high-dimensional Euclidean points.

The results of Chapter 8 are based on [C. Lammersen, A. Sidiropoulos, and C. Sohler. Streaming embeddings with slack. In *Proceedings of the 11th Algorithms and Data Structures Symposium (WADS '09)*, pages 483–494. Springer, 2009.].

Chapter 9

We conclude the thesis with Chapter 9, where we summarize our main results and also give some suggestions for future work.

Appendix A

This appendix contains some additional tables with detailed information about the experiments that we have run with our streaming implementation StreamKM++.

Appendix B

This appendix addresses some mathematical fundamentals which are assumed to be common knowledge throughout this thesis. The facts are stated for reference purposes, without giving any proofs.

1.2 Motivation

There are some relations between facility location, clustering, and computations of compact representations of finite metric spaces. Therefore, problems in these areas can be motivated in a similar way and have similar application scenarios. In the following, we will discuss this in detail for the problems considered in these areas.

1.2.1 Facility Location

Facility location problems capture a large variety of application scenarios in which we have to allocate resources to satisfy some requirement as good as possible while at the same time we have to pay some cost for the used resources. Therefore, it is not surprising that these kind of problems belong to the most studied problems in combinatorial optimization and operations research.

In this thesis, we are particularly interested in facility location problems for large-scale distributed systems of mobile objects. In such a system, each object is an autonomous computational entity that has its own local memory and that communicates with the other entities by message passing.

Applications of our scenario are, for example, in battery-powered wireless ad-hoc and sensor networks. These networks are governed by tight energy constraints. Often, the nodes are organized in energy-efficient clusters where some selected cluster heads offer a certain service to the subset of nodes contained in their cluster. Imagine, for example, we have a sensor network containing hundreds or thousands of homogeneous nodes, and the task of the nodes is to send periodically their sensed data to a distant base station where the end-user can access the data. Then, a service of a cluster head could be aggregating the data of its cluster nodes into a small set of meaningful information and taking on the transmission of the aggregated data to the base station. Each node can act as a cluster head, but, at any time, each node that is set up or maintained as a cluster head has an additional energy consumption due to transmitting data to the distant base station, staying in ready-to-receive mode, etc. The energy consumption of a non-serving cluster node is determined by its demand and the transmission power needed to reach a cluster head. The transmission power has to be higher the longer the distance to the nearest cluster head is. As a result, we have to find a good trade-off to have a small number of cluster heads and at the same time keep the energy needed for transmissions to the cluster heads low. The situation may be further aggravated in case that the nodes move continuously. Now, imagine that, to maintain the total energy consumption of the system low, nodes are allowed to change their status from serving cluster head to demanding cluster node or vice

versa. This scenario can be modeled as a facility location problem for a distributed system of mobile network nodes.

Since such systems, like the one described above, are too complex to analyze them in their entirety, we examine the following partial aspects of them:

Distributed Setting: We study the metric facility location problem with uniform opening costs and demands for large-scale distributed systems of static objects.

Kinetic Setting: Here, we are interested in the Euclidean facility location problem with non-uniform opening costs and demands for large-scale systems of mobile objects.

Dynamic Data Streams: We investigate the problem of computing the cost for the Euclidean facility location problem with uniform opening costs and demands for large-scale system of mobile objects where the movement of the objects is given as a dynamic data stream of update operations.

Distributed Setting

Typically, no node in a large-scale ad-hoc or sensor network knows all the distance information that is needed to solve or even approximate the facility location problem. Besides, it is not possible to gather this information at a single node since this would cause much communication between the nodes, which in turn would result in a high energy consumption. In such a scenario, a distributed algorithm is required.

The distributed algorithm presented in Chapter 3 computes a constant-factor approximation of the metric facility location problem using only three rounds of communication where the message size is logarithmic in the total number of nodes.

Kinetic Setting

Maintaining clusters in a large-scale network of mobile nodes is a challenging task. Good facility location algorithms should ensure a trade-off between the quality of the solution at any given point of time and its stability and efficiency under motion since each status change from demanding cluster node to serving cluster head incurs some costs.

Motivated by the fact that the KDS framework is common in the field of computational geometry and well-suited to maintain a combinatorial structure of continuously moving objects efficiently [2, 15, 54], we will develop a KDS for a mobile facility location problem in Chapter 4. Surprisingly, prior to our work, there was no KDS for the facility location problem known. Besides, it does not seem that the only known $(1 + \varepsilon)$-approximation algorithms [8, 76] can be translated to the kinetic setting. This is because the authors used the Arora-scheme [7] including dynamic programming techniques, which do not well comply with kinetization. So, at this stage, the best we can hope for is to construct a KDS maintaining a constant-factor approximation for the facility location problem, which we will do in Chapter 4.

Dynamic Data Streams

In battery-powered wireless ad-hoc and sensor networks of mobile nodes, it is common to communicate new positions of network nodes in form of a stream of update operations. Such an update may, for example, specify the 'name' of the network node, its 'old position' and its 'new position'. Thus, we can also think of it as a deletion of the node from its old position followed by an insertion of the same node at its new position.

The model of dynamic geometric data streams addresses such a scenario. We are given a stream of insert and delete operations of points from a discrete Euclidean space $\{1, \ldots, \Delta\}^d$, and our goal is to maintain some information about the aggregated data. The difficulty is that the size of the processed data prevents us from storing it completely. This restriction is modeled by allowing only space polylogarithmic in Δ and the length of the stream.

Since, in the facility location problem, the number of open facilities can be as large as the considered point set (and this can be as large as Δ^d), we cannot compute a solution in the dynamic data stream model. Instead, we focus on approximating the cost of a solution.

We remark that monitoring the cost can be very useful in resource allocation problems. For instance, it is often too costly to maintain, at all times, a nearly optimal set of open facilities in a distributed way. Instead, we can keep the same set of open facilities for some period of time but maintain, at all times, a good estimation of the optimal facility location cost. Then, we recompute a new set of open facilities by running a distributed algorithm as soon as the current cost estimation differs too much from the cost that we had after the latest run of the distributed algorithm.

Chapter 5 deals with an algorithm that computes a constant-factor approximation of the cost for the uniform facility location problem over dynamic geometric data streams.

1.2.2 Clustering

Clustering is the problem to partition a given set of objects into clusters such that objects from the same cluster are similar to each other and objects from different clusters are dissimilar. The goal is to simplify data by replacing a cluster by one or a few representatives, classify objects into groups of similar objects, or find patterns in some given data. Regarding to this, clustering has applications in various areas, including data mining, database systems, data compression, and machine learning. In many of these applications, the data occurs in the form of data streams or is stored on hard disks. This means that streaming access is orders of magnitude faster than random access or is even the only possible access to the data. To sum up, clustering algorithms for data streams are basic tools in the analysis of huge datasets.

One of the most widely used clustering algorithms is Lloyd's algorithm (sometimes also called *the k-means algorithm*) [39, 80, 82]. This algorithm is based on two observations: (1) Given a fixed set of centers, we obtain the best clustering by assigning each point to the nearest center and (2) given a cluster, the best center of the cluster is the center of gravity (i.e., the mean) of its points. Lloyd's algorithm applies these two local optimizations steps repeatedly to the current solution, until no more improvement is possible. It is known that

the algorithm converges to a local optimum [100], and no approximation guarantee can be given [73]. Recently, Arthur and Vassilvitskii [9] developed the k-MEANS++ algorithm, which is a seeding procedure for Lloyd's k-means algorithm that guarantees a solution with certain quality and gives good experimental results. An advantage of this algorithm is that it works also well for high-dimensional datasets. However, a disadvantage is that, like Lloyd's algorithm, the k-MEANS++ algorithm needs random access to the data items and, thus, is not suitable for data streams.

In Chapter 6, we will present a clustering algorithm for data streams that is based on the idea of the k-MEANS++ seeding procedure. Our algorithm utilizes the performance of the k-MEANS++ seeding on high-dimensional data but avoids random access to the data items.

1.2.3 Compact Representations of Finite Metric Spaces

Compact representations of finite metric spaces that fairly capture the pairwise distances and use only sublinear space are an important tool in the analysis of huge point sets since they can be stored in small space and much information about the point sets can be obtained from the corresponding pairwise distances. One option to obtain such a representation is the construction of a WSPD. Unfortunately, unless the input metric space is very simple (e.g., given is a multiset of points with many duplicates), one cannot find a sublinear space representation which preserves all pairwise distances. It is not even possible to guarantee that all pairwise distances are preserved up to any fixed factor. Simply said, it is unavoidable that we loose some distances in the sense that they can get arbitrarily distorted. According to this, we extend the classic notion of WSPD to WSPD with slack where the slack quantifies the fraction of pairwise distances that are not well preserved.

In Chapter 7, we will show how to construct a WSPD with low slack for low-dimensional Euclidean spaces. Our construction is based on techniques that compute small summaries for point clouds consisting of a certain number of closely located points. Due to this construction, for several points, the distances to points in their immediate vicinity can be arbitrarily distorted. However, the distances to all the other points, which are further away, are well preserved. Therefore, problems related to finding furthest neighbors in large point sets can be efficiently approximated by our compact representation. One such problem that has recently been considered is the computation of reverse furthest neighbors [110]. Given a huge set of points P, a small or moderate query set Q, and a query point $q \in Q$, the task is to find all the points in P with the property that q is their furthest neighbor among all points in Q. One application for this problem could be the placement of an obnoxious industrial facility. Given a set P of residential sites and a set Q of potential locations for building such an obnoxious facility, one reasonable strategy is to select a location in Q that is further away from as many residential sites as possible, i.e., the location with the largest set of reverse furthest neighbors in P. The reader is referred to [110] for more examples in this area.

Another application for our compact representation is hierarchical clustering. For instance, given a Euclidean point set P, the complete linkage clustering (or the furthest

neighbor method) starts with $|P|$ singleton clusters and successively merges the two nearest clusters where the distance between two clusters is defined as the distance between the two furthest objects in the two clusters. The merge step is repeated until the number of clusters corresponds to the desired number of clusters. In this scenario, our compact representation can be seen as a certain stage, when we have already performed a sequence of several merge steps. Thus, by applying the clustering method on the representation, we get a good approximation of the clustering for P. This is especially useful when the true clusters of P are compact and roughly equal in size.

In many applications, the datasets are high-dimensional and given in form of a data stream. Examples of such datasets include the web graph, Internet traffic logs, clickstreams, and genome data. To analyze such datasets, streaming algorithms that embed a set of high-dimensional points with low distortion and low slack into a low-dimensional space can be of particular interest. Besides the fact that the embedded point set can be stored in small space, another benefit is that it might be useful to detect some structure in the original data more easily. For example, let us assume, we map the input data to the Euclidean plane or to $\mathbb{R}^3$. Then, it is much simpler for the human visual system to detect structure in this data, tight clusters or isolated points, for instance.

Chapter 8 addresses the development of streaming algorithms for computing embeddings with low distortion and low slack of high-dimensional Euclidean spaces, doubling metric spaces with low doubling dimension, and general metric spaces.

1.3 Related Work

Facility location variants as well as techniques to compute compact representations of metric spaces have extensively been studied in computer science. It goes beyond the scope of this thesis to give a comprehensive overview of the vast available literature. In the following, we will focus our summary of the work emerged in both areas on the results which are most relevant to this thesis.

1.3.1 Facility Location

One of the most studied facility location problems is the problem which we refer to as the *general facility location problem*. Compared to the variants considered in this thesis, in the general facility location problem, only a subset of the input points are potential facility locations. More precisely, we are given a set of facilities $\mathcal{F}$ and a set of clients $\mathcal{C}$. With each facility $x_i \in \mathcal{F}$, there is a non-negative opening cost f_i associated. Furthermore, there is a distance measure D defined on the input points, and, for each facility-client pair $(x_i, y_j) \in \mathcal{F} \times \mathcal{C}$, the distance $\mathrm{D}(x_i, y_j)$ specifies the cost for connecting y_j to x_i. The goal is to open a subset $F \subseteq \mathcal{F}$ of the facilities and to connect each client to an open facility so as to minimize the sum of the opening costs for F and the connection costs for $\mathcal{C}$.

Note that, in the literature, the general facility location problem is also often called *uncapacitated facility location problem*. This indicates that each facility can serve an un-

limited number of clients, whereas, in capacitated versions of the problem, each facility can serve only a certain limited number of clients. Since we only consider uncapacitated facility location problems in this thesis, we omit the attribute 'uncapacitated'.

An instance of the general facility location problem is said to be *metric* if the distance measure D is non-negative, symmetric, and satisfies the triangle inequality. The general metric facility location problem is known to be NP-hard. The first polynomial-time constant-factor approximation algorithm for this problem was given by Shmoys et al. [102]. Later, several other polynomial-time constant-factor approximations have been proposed [12, 18, 27, 30, 51, 68, 69, 70, 77, 83, 87, 104]. These algorithms can roughly be grouped into algorithms using mainly linear programming (LP) rounding techniques, primal-dual methods, local search strategies, greedy strategies, or combinations of these techniques.

The approximation algorithm of Shmoys et al. [102] relies on the LP rounding technique due to Lin and Vitter [79]. An LP rounding algorithm proceeds in two steps: The first step is to solve the linear relaxation of an integer programming formulation of the considered problem, and the second step is to round the obtained fractional LP solution to an integer solution. In this way, the algorithm of Shmoys et al. [102] achieves an approximation ratio of 3.16. Guha and Khuller [51] improved the LP rounding algorithm of Shmoys et al. [102] and combined it with a simple local search phase. Starting with the solution obtained from the LP rounding, in the local search phase, the amount of cost that is saved by opening a closed facility is computed for each closed facility. While there exists a facility whose amount of saved cost is positive, the facility that maximizes the decrease of the total facility location cost is opened. This combination of LP rounding and local search achieves an approximation ratio of 2.41. Chudak and Shmoys [30] developed a 1.736-approximation algorithm by improving the algorithm of Shmoys et al. [102]. The key elements to their improvement are a new rounding procedure for the LP relaxation of the facility location problem and the use of information about the dual linear program to the LP relaxation of the problem. A further improvement of the LP rounding algorithm has been proposed by Sviridenko [104] resulting in an approximation ratio of 1.582.

Korupolu et al. [77] proposed a local search algorithm for the general metric facility location problem. In general, a local search algorithm starts with a feasible solution to the considered problem and applies iteratively a local improvement step in which minor modifications are made in order to obtain a solution of lower cost. The local improvement step presented in [77] searches for one facility or a pair of one open and one closed facility such that changing the status of the involved facilities decreases the total facility location cost. The algorithm of Korupolu et al. [77] yields an approximation ratio of $5 + \varepsilon$ for any constant $\varepsilon > 0$, and, according to [27], it can be implemented to run in $\mathcal{O}(n^4 \cdot \log(n/\varepsilon))$ time. By using other techniques in the analysis, Arya et al. [12] proved that the local search algorithm of Korupolu et al. [77] actually achieves an approximation ratio of $3 + \varepsilon$.

One of the most elegant approximation algorithms for the general metric facility location problem is the primal-dual method developed by Jain and Vazirani [70]. In general, the goal of a primal-dual method is to simultaneously compute a feasible integer solution for the original problem as well as a feasible solution to the dual linear program to its LP relaxation. In case that the considered problem is a minimization problem, like the general

facility location problem, the cost of a feasible solution to the dual LP can be used as a lower bound on the optimal cost. Jain and Vazirani [70] proved that their primal-dual method for facility location is a 3-approximation algorithm that can be implemented to run in $\mathcal{O}(n^2 \cdot \log(n))$ time. Later, Jain et al. [68] used a primal-dual method in the analysis of two greedy algorithms. Their first greedy algorithm iteratively opens among all currently closed facilities the facility that minimizes, for any subset U of all currently unconnected clients, the ratio of the opening costs of the particular facility plus the connection cost of U to the size of U. This algorithm achieves an approximation ratio of 1.861 and has a running time of $\mathcal{O}(n^2 \cdot \log(n))$. The second greedy algorithm was obtained from the first one by small modifications resulting in an improved approximation ratio of 1.61 and a higher running time of $\mathcal{O}(n^3)$. Furthermore, Mettu and Plaxton [87] presented a greedy algorithm for the facility location variant with $\mathcal{C} = \mathcal{F}$ which implicitly uses the primal-dual method of Jain and Vazirani [70]. This is done by defining so-called 'radii' for amortizing the cost needed to open a facility at a particular location. The algorithm opens iteratively the facility $x_i \in \mathcal{F}$ with the smallest radius that has no other open facility in the ball whose center is x_i and whose radius is twice the radius of x_i. The algorithm yields an approximation ratio of 3 and has a running time of $\mathcal{O}(n^2)$. Finally, the second best approximation algorithm for the general facility location problem is based on the algorithm of Jain et al. [68]. This algorithm also involves the primal-dual method. It achieves an approximation ratio of 1.52 and has a running time of $\tilde{\mathcal{O}}(n^2)$ [83].

Another algorithm for the general facility location problem has been developed by Charika and Guha [27]. This algorithm is a 1.728-approximation algorithm that combines the primal-dual method of Jain and Vazirani [70], a modified version of the local search technique presented by Guha and Khuller [51], and the LP rounding algorithm of Chudak and Shmoys [30]. Furthermore, Byrka and Aardal [18] proposed an algorithm that uses a modified version of the algorithm of Chudak and Shmoys [30] and combines this with the algorithm of Jain et al. [68]. The algorithm yields the best approximation ratio up to now, which is 1.5.

Concerning hardness results, Guha and Khuller [51] proved by a reduction from set cover that the general metric facility location problem of n input points cannot be approximated in polynomial time within a factor of 1.463, unless NP $\subseteq$ DTIME$[n^{\mathcal{O}(\log(\log(n)))}]$. Combining this result with an observation of Sviridenko implies that the approximation lower bound of 1.463 also holds, unless P = NP (see [108]). Furthermore, Thorup proved [106] that any constant-factor approximation algorithm, even a randomized one, requires $\Omega(n^2)$ time to compute a solution to the general metric facility location problem. Bădoiu et al. [14] extended this hardness result by showing that any bounded-factor approximation algorithm, even a randomized one, requires $\Omega(n^2)$ time to compute the *cost* of the general metric facility location problem. This result holds even for the variant with uniform opening costs.

However, for some special variants of the metric facility location problem, the hardness results mentioned above are no longer valid. For instance, Bădoiu et al. [14] considered the metric facility location problem with uniform opening costs in which every point can open a facility. In a first step, they proved that the sum of the radii defined by Mettu

and Plaxton [87] is an estimator that approximates the optimal facility location cost to within a constant factor. In a second step, they showed how to obtain a constant-factor approximation of this estimator by using an adaptive sampling approach. This resulted in an algorithm for the considered variant of the metric facility location problem that computes a constant-factor approximation of the cost in $\mathcal{O}(n \cdot \log^2(n))$ time. Furthermore, the non-approximability result of Guha and Khuller [51] is no longer valid in the special case of Euclidean spaces. A first randomized polynomial-time approximation scheme for the general Euclidean facility location problem in the plain has been developed by Arora et al. [8]. This algorithm is based on the Arora-scheme [7] and computes a $(1+\varepsilon)$-approximation in $\mathcal{O}(n^{1+\mathcal{O}(1/\varepsilon)} \log(n))$ time. The result of Arora et al. [8] was then improved by Kolliopoulos and Rao [76]. Assuming that there exists any polynomial-factor approximation for the total connection cost, they obtained a randomized polynomial-time approximation scheme that works in any constant-dimensional Euclidean space and has a running time of $\mathcal{O}(2^{\mathcal{O}((\log(1/\varepsilon)/\varepsilon)^{d-1})} n \log^{d+6}(n))$.

For a more comprehensive overview of results on facility location problems in a classical setting, we refer the reader to the surveys by Shmoys [101] and Vygen [108].

The facility location problem has also been investigated in other settings. In the following, we will summarize the results obtained in distributed and kinetic settings as well as in data stream models.

Distributed Setting

Surprisingly, the first algorithm for a distributed facility location problem was proposed just a few years ago [90]. Given a set of m facilities and a set of n clients, Moscibroda and Wattenhofer [90] investigated the general non-metric facility location problem (i.e., distances do not have to satisfy the triangle inequality) in a synchronous message passing model. In their considered model, the communication network is a complete bipartite graph with communication links between each facility-client pair, and each node can send in each communication round a message containing $\mathcal{O}(\log(n))$ bits to each neighbor in the communication network. To approach the distributed facility location problem, Moscibroda and Wattenhofer used some ideas from the centralized primal-dual method of Jain and Vazirani [70]. The obtained distributed primal-dual method provides a trade-off between the number of communication rounds and the resulting approximation ratio. In particular, it achieves an $\mathcal{O}(\sqrt{k}(m\varrho)^{1/\sqrt{k}} \log(m+n))$ approximation in $\mathcal{O}(k)$ communication rounds with a message size of $\mathcal{O}(\log(n))$ bits. Here, ϱ is a coefficient that depends on the cost values of the input instance.

In Chapter 3, we consider the metric facility location variant with uniform opening costs for the facilities and $X := \mathcal{C} = \mathcal{F}$ in the synchronous message passing model where the communication network is a clique. Compared to the problem studied in [90], our problem is much simpler, and so the algorithm presented in Chapter 3 is incomparable with the algorithm of Moscibroda and Wattenhofer. We developed our randomized distributed algorithm based on results from Mettu and Plaxton [87] and Bădoiu et al. [14]. As mentioned before, Bădoiu et al. [14] proved that the sum of the radii defined by Mettu and

Plaxton [87] is a constant-factor approximation of the optimal facility location cost. Furthermore, for any facility $x_i \in X$, they gave a lower bound on the number of points located in the ball whose center is x_i and whose radius equals the radius of x_i. Using this lower bound, we designed our randomized distributed algorithm in a way that it opens a subset of the potential facilities such that, with high constant probability, the total opening cost is at most a constant factor larger than the sum of the radii and each client x_i has an open facility in a ball whose center is x_i and whose radius is at most a constant factor larger than the radius of x_i. Hence, our algorithm computes with high constant probability a constant-factor approximation for X.

In a follow-up study on the results of Moscibroda and Wattenhofer [90] and the results presented in Chapter 3, Pandit and Pemmaraju [97] further investigated the metric version of the problem studied in [90]. Based on the primal-dual method of Jain and Vazirani [70] and a rapid randomized sparsification of graphs due to Gfeller and Vicari [49], they obtained a 7-approximation in $\mathcal{O}(\log(m) + \log(n))$ communication rounds with a message size of $\mathcal{O}(\log(m+n))$ bits. This technique was then generalized to get an algorithm that, for each constant k, runs in k communication rounds and computes a solution whose cost is only a factor of $O(m^{2/\sqrt{k}} \cdot n^{3/\sqrt{k}})$ larger than the optimal cost. We point out that the technique of Pandit and Pemmaraju can also be used to obtain a constant-factor approximation in $\mathcal{O}(\log(n))$ communication rounds for the variant of our considered metric facility location problem where the opening cost of facilities are non-uniform. Therefore, their result can be seen as a generalization of our result.

For more information about distributed computing, the reader is referred to [81, 98].

Kinetic Setting

Some frameworks have been proposed for handling kinetic data. In this thesis, we consider a common model for processing points in motion, called kinetic data structures (KDS), which was introduced by Basch et al. [15].

Prior to the work presented in Chapter 4, there was no KDS for the facility location problem known. However, some results have been obtained in the KDS framework for problems related to clustering, to which the facility location problem belongs. For instance, Gao et al. [46] provided a randomized KDS to maintain a set of centers among moving points in the plane such that, given a specified radius, all the points are covered by balls of the given radius centered at the chosen center points. Gao et al. showed that the size of the center set is at most a constant factor larger than the minimum one. Hershberger investigated a similar problem in [60]. More precisely, he proposed a deterministic KDS for maintaining a covering of moving points in $\mathbb{R}^d$ by unit boxes such that the number of boxes is always within a factor of 3^d of the optimal static covering at any instance. Another clustering problem that has been studied in the KDS framework is the kinetic k-center problem. The goal of the kinetic k-center problem is to maintain a set of k centers so as to minimize the maximum distance of any point to its closest center at any point of time. Gao et al. [47] proposed a deterministic KDS that maintains, for a set of moving points in $\mathbb{R}^d$, an 8-approximation of the discrete k-center problem, i.e., the centers have to be a subset of

the moving input points. Bereg et al. [17] studied 1-center problems in which the center is not necessarily located at one of the moving input points. Among other results, he showed that, given a precision parameter ε, $0 < \varepsilon < 1$, there is a strategy for moving a center such that the location of this center provides a $(1+\varepsilon)$-approximation of the 1-center problem for a set of moving points in the plane and, assuming each input point moves with velocity at most 1, the velocity of the center never exceeds $(2+\varepsilon)(1+\varepsilon)/\sqrt{2\varepsilon+\varepsilon^2}$. Furthermore, a KDS for the k-center problem in the context of outliers can be found in [45]. Har-Peled [56] investigated the k-center problem in a mobile setting different from the KDS framework. Instead of handling events, a static set which ensures a constant-factor approximation at all times is provided. However, the size of this set is $k^{\mu+1}$, where μ is the degree of the polynomial of the trajectories. Finally, we are aware of another result concerning clustering which addresses a randomized KDS for the Euclidean max-cut problem [31, 42].

For other work on KDSs, we refer the reader to the surveys by Guibas [54, 55].

Unfortunately, it does not seem that the only known $(1+\varepsilon)$-approximation algorithms for facility location [8, 76] can be transferred to the kinetic setting since they are based on the Arora-scheme [7] including dynamic programming techniques, which do not well comply with kinetization. Our KDS for the mobile Euclidean facility location problem combines a modified version of the greedy algorithm of Mettu and Plaxton [87] with a counting argument of Bădoiu et al. [14]. Given any static Euclidean point set P, the original greedy algorithm opens as few facilities as possible in a way that each point $p_i \in P$ has at least one open facility in the ball with center p_i and twice the radius of p_i. This results in a constant-factor approximation of the facility location problem for P. Concerning the radii defined by Mettu and Plaxton, the counting argument of Bădoiu et al. asserts that, for any facility $p_i \in P$, a constant-factor approximation of the radius of p_i can be computed by just counting the number of points from P contained in exponentially growing balls centered at p_i. This counting argument facilitates us to efficiently kinetize a modified version of the static greedy algorithm proposed by Mettu and Plaxton.

Data Streams

Although, many geometric approximation algorithms have been developed in data stream models, we are only aware of three results concerning facility location problems.

In [41], Fotakis presented a streaming algorithm for the metric facility location variant in which every input point is a potential facility location. The algorithm combines an online facility location algorithm due to Meyerson [88] with an incremental facility location algorithm due to Fotakis [40]. The course of the algorithm is controlled by so-called *final distances*. The final distance of a point is an upper bound on the distance of this point to the nearest facility at any future point of time. While reading the input stream, the next point is chosen as open facility with a probability that is proportional to the ratio between the final distance of this point and the opening cost. In case that a point is chosen as open facility, it is stored in memory and replaces every currently stored facility which has the property that its distance to the new facility is at most a certain fraction of its final distance. In this way, the algorithm maintains a set of open facilities such that the

total associated facility location cost is at most a constant factor larger than the optimal facility location cost. Unfortunately, both the update time and the space requirement of the algorithm are linear in the number of opened facilities, which can be linear in the input size.

Chang [26] developed a multi-pass streaming algorithm for the metric facility location variant in which every input point is a potential facility location. In contrast to the data stream models considered in this thesis, in the multi-pass streaming model, an algorithm is allowed to perform more than one sequential scan over the input data. During and after each such pass, the amount of available local memory space is assumed to be sublinear in the size of the input stream. To approach the facility location problem, Chang used an iterative algorithm that is based on a technique proposed by Indyk [61]. In each iteration, the algorithm takes a random sample from the input stream and computes a subset of open facilities by applying some known facility location algorithm on the sample set. Then, the algorithm removes all the points from consideration that are served sufficiently well and iterates on all the remaining points. This is repeated, until all input points are served sufficiently well. Chang showed that his algorithm uses $\mathcal{O}(\ell)$ passes and $\tilde{\mathcal{O}}(kn^{2/\ell})$ space to compute a set of open facilities such that the total associated facility location cost is at most a factor of $\mathcal{O}(\ell)$ larger than the optimal facility location cost. Here, k is the number of open facilities and n is the number of input points. Thus, similar to Fotakis' streaming algorithm, there exist facility location instances for which the space requirement of Chang's algorithm is not sublinear in the input size. However, Chang justified his approach by proving that, for the considered facility location problem, any randomized ℓ-pass streaming algorithm requires $\Omega(n/\ell)$ bits of memory to compute even a polynomial-factor approximation of the optimal facility location cost.

Previous to the result presented in Chapter 5, the only real streaming algorithm for facility location was proposed in [64], where the author introduced the model of dynamic geometric data streams and studied different geometric problems in this model. A dynamic geometric data stream is a sequence of insert and delete operations on a point set $P \subseteq \{1, \ldots, \Delta\}^d$ in a discrete d-dimensional Euclidean space. In the facility location variant studied in [64], the opening costs are uniform and every point in P can open a facility. For the purpose of guaranteeing a space requirement that is only polylogarithmic in the size of the input stream, Indyk developed an algorithm that approximates the optimal facility location cost for P instead of an optimal set of open facilities. This is done by defining a certain partition of the space into nested square grids and a set of cells in this partition such that the number of these cells gives an $\mathcal{O}(\log(\Delta))$-approximation of the optimal facility location cost. During the approximation process to estimate the number of these cells, the algorithm of [64] looses another factor of $\mathcal{O}(\log(\Delta))$[1].

In Chapter 5, we use a similar partition of the space into nested square grids as in [64], and we show that opening a subset of the cells defined in [64] results in a constant-factor approximation of the optimal facility location cost. This leads to a streaming algorithm

[1]The author of [64] mentions that, with the help of a more intricate analysis, the approximation factor can be improved to $\mathcal{O}(\log(\Delta))$.

that computes a constant-factor approximation of the cost for the facility location problem considered in [64], which strongly improves Indyk's result.

We point out that the approximation of the facility location cost was considered again in [14]. As mentioned at the beginning of this section, the authors of [14] proposed a sublinear-time algorithm that computes in $\mathcal{O}(n \log^2(n))$ time a constant-factor approximation for the cost of the metric facility location variant in which every input point is a potential facility location and in which the opening costs for the facilities are uniform[2]. Unfortunately, despite the relation of streaming and sublinear-time algorithms, the techniques cannot be transferred to the other model.

Note that the facility location problem in which every input point is a potential facility location and in which the opening costs for the facilities are uniform is closely related to the k-median and k-means clustering problems. In the k-median clustering problem, we are given a set of points and an integer k, and the goal is to determine a set of k centers such that the sum of the distances from the input points to their corresponding nearest center is minimized. The cost function of the k-means clustering problem differs from the one of the k-median clustering problem only in the way that we sum up the squared distances from the input points to their corresponding nearest center. For both clustering problems, a number of streaming algorithms have been developed [4, 28, 29, 36, 37, 44, 53, 57, 58]. Like the streaming algorithm presented in Chapter 6, many of these algorithms apply a merge-and-reduce technique based on a decomposition technique of Bentley and Saxe [16] to obtain a small coreset (see [2] for the introduction of the notion of coresets). Our coreset construction for the k-means clustering problem is based on the k-means++ seeding procedure [9]. We point out that the k-Means++ seeding has also been investigated in [3] and [4]. However, our result differs from the results given in [3] and [4] and was obtained independently.

In any case, all known algorithms for the k-median and k-means clustering problem require space $\Omega(k)$. Thus, they implicitly assume that k is small, i.e., $k \in \text{polylog}(\Delta)$ in dynamic data streams and $k \in \text{polylog}(n)$ in insertion-only data streams, where Δ is the spread of the input points and n is the length of the stream. As mentioned above, in facility location problems in which every input point is a potential facility location, the number of cluster centers k can be as large as the maximum size of the point set under consideration. In Chapter 5, we will show that we can approximate the cost for such a facility location problem in space $o(k)$. No similar result is known for the k-median and k-means clustering problems.

For other work in data stream models, we refer the reader to [66, 92, 93].

1.3.2 Compact Representations of Finite Metric Spaces

The compact representations of finite metric spaces considered in this thesis are well-separated pair decompositions with slack (WSPDs with slack) and metric embeddings

[2]Since the size of the representation of an n-point metric space is $\Theta(n^2)$, the complexity of this algorithm is sublinear with respect to the input size.

with slack. Since we are not aware of any prior work on WSPDs with slack, the following overview deals with classical WSPDs. Afterwards, we will summarize the results obtained in the area of metric embeddings with slack.

WSPD

The notion of WSPD has been introduced in [22]. In the same paper, Callahan and Kosaraju showed that, for any set of n points from any constant-dimensional Euclidean space and for any constant ε with $0 < \varepsilon < 1$, there always exists an ε-WSPD consisting of $\mathcal{O}(n)$ pairs and such an ε-WSPD can be computed in $\mathcal{O}(n \log(n))$ time. The construction was later simplified by Har-Peled and Mendel [59], who observed that a WSPD can directly be generated from a compressed quadtree [22]. Also based on a compressed quadtree construction, Chan [25] showed that a WSPD for a Euclidean point set can be found in linear time if the spread of the point set is polynomially bounded in the size of the point set. Concerning dynamic point sets, Callahan [20] presented a deterministic algorithm and later Fischer and Har-Peled [38] a simpler randomized algorithm that maintains a WSPD for a constant-dimensional Euclidean point set in polylogarithmic time under insertions and deletions. In high dimensions, it is known that a WSPD can have quadratic complexity. One example is the uniform n-point metric (with all pairwise distances equal to 1), which can be realized as the vertices of a simplex in $\mathbb{R}^{n-1}$.

Since WSPDs are useful data structures to represent distances between points efficiently, they have been applied for solving many proximity problems for point sets in a Euclidean space [10, 11, 19, 21, 22, 34, 50, 59, 78, 94].

Talwar [105] extended the notion of WSPD to spaces with low doubling dimension. He showed that, given any constant ε with $0 < \varepsilon < 1$ and any n-point metric space with constant doubling dimension and spread Δ, there always exists an ε-WSPD consisting of $\mathcal{O}(n \log(\Delta))$ pairs. Furthermore, Gao and Zhang studied the construction of WSPDs for unit-disk graphs [48].

Metric Embedding with Slack

The theory of metric embeddings received much attention in recent years, and embedding techniques have been applied in the development and analysis of many algorithms that operate on an underlying metric space. For recent work on metric embeddings, we refer the reader to the surveys [63, 67, 84]. In the following overview of prior work, we focus on the results that are related to metric embeddings with slack or that have been relevant in designing the algorithms presented in Chapter 8.

Kleinberg et al. [75] introduced the notion of embeddings with slack. Among other results, they showed that, for any constant σ with $0 < \sigma < 1$, any metric space with bounded doubling dimension can be embedded with distortion $\mathcal{O}(1)$ and slack σ into a constant-dimensional Euclidean space. The results from [75] have been extended to arbitrary metric spaces and to embeddings under any ℓ_p norm, $p \geq 1$, by Chan et al. [23]. Furthermore,

Abraham et al. [1] developed embeddings with low distortion and low slack for arbitrary metric spaces that additionally guarantee a constant average distortion.

Metric approximation with slack has also been investigated in the setting of graph spanners. Chan et al. [24] showed that, for any weighted graph G and any ε with $0 < \varepsilon < 1$, there exists a spanner of G with linear number of edges achieving stretch $\mathcal{O}(\log(1/\varepsilon))$ and slack ε. The authors also gave a spanner construction which is the starting point of the embedding with slack of general metric spaces presented in Chapter 8. In order to transform this construction to the streaming model, we use a technique that has been applied by Czumaj and Sohler [32] to achieve 2-pass streaming algorithms for clustering problems. We point out that, in the same paper, Czumaj and Sohler [32] introduced the concept of α-preserving metric embeddings, which is closely related to embeddings with slack. Their concept can be seen as a generalization of coresets. The goal is to embed a metric space into a structurally simpler metric space that approximates the original metric up to a factor of α with respect to a given optimization problem.

Embeddings of point sets into trees via a quadtree partitioning have been used by Indyk [64] to obtain approximation algorithms for several geometric problems. Also, Frahling and Sohler [44] applied a similar quadtree partitioning to get streaming algorithms for different clustering problems. In Chapter 8, we use a similar partitioning technique to embed Euclidean metric spaces with low distortion and low slack.

2 Preliminaries

This chapter deals with definitions that are used throughout the whole thesis. More special definitions, which are only used to describe or analyze a certain algorithm, are introduced in the corresponding main chapters. Therefore, the first section of most of the main chapters is for preliminaries, which include such special definitions.

In this thesis, we will develop approximation algorithms for geometric problems in various metric spaces. In particular, in Chapters 7 and 8, we will present algorithms for computing compact space representations of different types of finite metric spaces. Section 2.1 covers definitions of these metric spaces.

A big part of this thesis is devoted to facility location problems. We will consider various facility location problems in different kinds of settings. More precisely, we will present facility location algorithms for distributed and mobile settings as well as for data streams. Formal definitions of the considered facility location problems are given in Section 2.2. Our facility location algorithms for distributed and mobile settings are based on the greedy algorithm of Mettu and Plaxton [87]. We will present this algorithm in Section 2.3. Finally, in Section 2.4, we will introduce the computational models that we have applied to develop our algorithms in the different kinds of settings.

2.1 Distance Functions and Metric Spaces

An important class of distance functions are metric spaces. In this section, we will give a formal definition of general, Euclidean, and doubling metric spaces.

Distance Functions

Let X be any non-empty set of elements. A function $\mathrm{D} : X \times X \to \mathbb{R}$ is a *distance function* on X if it satisfies the following axioms:

- *Non-Negativity*: For any $x, y \in X$, we have $\mathrm{D}(x, y) \geq 0$.
- *Symmetry*: For any $x, y \in X$, we have $\mathrm{D}(x, y) = \mathrm{D}(y, x)$.

We generalize the definition of distance functions to sets. More precisely, for any finite set X and any distance function D on X, we define

$$\forall\, x \in X \;\; \forall\, Y \subseteq X : \mathrm{D}(x, Y) := \min_{y \in Y} \mathrm{D}(x, y)$$

and

$$\forall\, Y \subseteq X \;\; \forall\, Z \subseteq X : \mathrm{D}(Y, Z) := \min_{y \in Y} \mathrm{D}(y, Z) \;\; .$$

General Metric Spaces

A *metric space* M is a pair (X, D), where X is a non-empty set of elements and D is a distance function on X that satisfies the following axioms:

- *Reflexivity*: For any $x, y \in X$, we have $\mathrm{D}(x, y) = 0$ if and only if $x = y$.
- *Triangle Inequality*: For any $x, y, z \in X$, we have $\mathrm{D}(x, z) \leq \mathrm{D}(x, y) + \mathrm{D}(y, z)$.

The complexities of several algorithms presented in this thesis depend on the spread of the given input metric space. For a finite metric space $M = (X, \mathrm{D})$ with $\mathrm{D}(x, y) \neq 0$ for all pairs $(x, y) \in X \times X$ with $x \neq y$, the spread of M is defined as the ratio of the farthest pair distance in X to the closest pair distance in X.

Euclidean Metric Spaces

Our distance measure will often be the *Euclidean distance*. The Euclidean distance between two points is given by the *Euclidean length* of the difference vector of both points. More precisely, let $x := \left(x^{(1)}, x^{(2)}, \ldots, x^{(d)}\right)$ and $y := \left(y^{(1)}, y^{(2)}, \ldots, y^{(d)}\right)$ be any two points from the Euclidean space $\mathbb{R}^d$, where the dimension $d \in \mathbb{N}$ is any natural number. Then, the Euclidean distance between x and y is defined as

$$\mathrm{D}(x, y) := \|x - y\| = \sqrt{\sum_{i=1}^{d} \left(x^{(i)} - y^{(i)}\right)^2} \ .$$

Since the Euclidean distance satisfies the condition of reflexivity and the triangle inequality, it is a metric space.

Doubling Metric Spaces

A metric space $M = (X, \mathrm{D})$ is called a *doubling metric space* if, there exists some $\lambda \in \mathbb{N}$, such that each ball with any radius r centered at any point in X can be covered by 2^λ balls each of radius $r/2$ and centered at a point in X. The value λ is called the *doubling dimension* of M.

The doubling dimension can be seen as a generalization of the Euclidean dimension since $\mathbb{R}^d$ has a doubling dimension of $\Theta(d)$ [59]. Besides, the doubling dimension extends the notion of growth restricted metric spaces defined by Karger and Ruhl [74].

2.2 Facility Location Problems

In this section, we will define different types of facility location problems. The first problem will be a facility location problem in general metric spaces. The problem definition is then extended to powers of metric spaces. Finally, we will introduce a mobile facility location problem in Euclidean spaces.

Metric Facility Location Problem

In the *metric facility location problem*, we are given a metric space $(\mathcal{F} \cup \mathcal{C}, \mathrm{D})$, where $\mathcal{F} := \{x_1, x_2, \ldots, x_m\}$ is a set of m *facilities* and $\mathcal{C} := \{y_1, y_2, \ldots, y_n\}$ is a set of n *clients*. With each facility $x_i \in \mathcal{F}$, there is a non-negative *opening cost* f_i associated. Each client $y_j \in \mathcal{C}$ has a non-negative *demand* d_j. The goal is to find a subset $F \subseteq \mathcal{F}$ of *open facilities* such that the objective

$$\mathrm{FacLoc}((\mathcal{F}, \mathcal{C}), F) := \sum_{x_i \in F} f_i + \sum_{y_j \in \mathcal{C}} d_j \cdot \mathrm{D}(y_j, F)$$

is minimized. The first part of the objective is the opening cost related to the open facilities in F. The second part of the objective is the cost related to all clients in $\mathcal{C}$, which we call the *connection cost*.

Throughout the whole thesis, we will only consider the variant of the metric facility location problem with $X := \mathcal{F} = \mathcal{C}$, where $X := \{x_1, x_2, \ldots, x_n\}$ is a set of n points. We then shortly write the facility location cost as

$$\mathrm{FacLoc}(X, F) := \mathrm{FacLoc}((\mathcal{F}, \mathcal{C}), F) \ .$$

In the *uniform metric facility location problem* with $X := \mathcal{F} = \mathcal{C}$, both the opening costs of the facilities and the demands of the clients are uniform. More precisely, we assume that, for each $x_i \in X$, we have $f_i = f$ for some fixed value $f \geq 0$ and $d_i = 1$. Then, the goal is to find a subset $F \subseteq X$ of open facilities such that the objective

$$\mathrm{FacLoc}(X, F, f) := f \cdot |F| + \sum_{x_j \in X} \mathrm{D}(x_j, F)$$

is minimized.

In case that the given metric space is a Euclidean space, we call the problem the (*uniform*) *Euclidean facility location problem*.

Facility Location Problem for Powers of Metric Spaces

In the *facility location problem for powers of metric spaces*, we are given a metric space $(\mathcal{F} \cup \mathcal{C}, \mathrm{D})$ and a constant *metric exponent* $\ell \geq 1$. As well as for the metric facility location problem, in this thesis, we will only consider the variant with $X := \mathcal{F} = \mathcal{C}$, where $X := \{x_1, x_2, \ldots, x_n\}$ is a set of n points. With each point $x_i \in X$, there is a non-negative opening cost f_i and a non-negative demand d_i associated. The goal is to find a subset $F \subseteq X$ of open facilities such that the objective

$$\mathrm{FacLoc}(X, F, \ell) := \sum_{x_i \in F} f_i + \sum_{x_j \in X} d_j \cdot \mathrm{D}(x_j, F)^\ell$$

is minimized.

In the *uniform facility location problem for powers of metric spaces* with $X := \mathcal{F} = \mathcal{C}$, both the opening costs and the demands of the points are uniform. We assume that, for

each $x_i \in X$, we have $f_i = f$ for some fixed value $f \geq 0$ and $d_i = 1$. Then, the goal is to find a subset $F \subseteq X$ of open facilities such that the objective

$$\mathrm{FacLoc}(X, F, f, \ell) := f \cdot |F| + \sum_{x_j \in X} \mathrm{D}(x_j, F)^\ell$$

is minimized.

Mobile Facility Location Problem

In the *mobile facility location problem*, we are given a set of moving facilities $\mathcal{F}$ and a set of moving clients $\mathcal{C}$ in a Euclidean space $\mathbb{R}^d$. As described before, in this thesis, we will only consider the mobile facility location problem with $P := \mathcal{F} = \mathcal{C}$, where $P := \{p_1, p_2, \ldots, p_n\}$ is a set of n moving points in $\mathbb{R}^d$. Let $p_i(t)$ denote the position of p_i at time t, and let $P(t) := \{p_1(t), p_2(t), \ldots, p_n(t)\}$. For each point $p_i \in P$, there exists a non-negative opening cost f_i and a non-negative demand d_i. Observe that both the opening cost and the demand of a point do not change over time. The mobile facility location problem is to maintain, at each point of time t, a subset $F(t) \subseteq P(t)$ of open facilities such that

$$\mathrm{FacLoc}(P(t), F(t)) := \sum_{p_i(t) \in F(t)} f_i + \sum_{p_j(t) \in P(t)} d_j \cdot \mathrm{D}(p_j(t), F(t))$$

is minimized.

2.3 The Mettu-Plaxton Algorithm

This section addresses the greedy algorithm of Mettu and Plaxton [87] that computes a constant-factor approximation for the metric facility location problem. Let (X, D) be a metric space, where $X = \{x_1, \ldots, x_n\}$ is a set of n points and D is a distance function defined on X. Following the definitions from Section 2.2, the opening cost of a point $x_i \in X$ is denoted by f_i and its demand by d_i.

As mentioned in the previous chapter, the Mettu-Plaxton algorithm implicitly applies the primal-dual method of Jain and Vazirani proposed in [70]. This is done by defining so-called 'radii' for amortizing the cost needed to open a facility at a particular location. The idea of the Mettu-Plaxton algorithm is to open only a few facilities but, at the same time, to guarantee that each point $x_i \in X$ has at least one open facility in the ball with center x_i and twice the radius of x_i. After giving a formal definition of balls and radii, we describe the algorithm in more detail.

Balls. For a point $x_i \in X$ and a non-negative value r, we define $\mathcal{B}(x_i, r)$ to be the ball with center x_i and radius r. Given such a ball $\mathcal{B}(x_i, r)$, we let $\mathrm{weight}(\mathcal{B}(x_i, r))$ denote the sum of the demands of all the points in X that are located in the ball $\mathcal{B}(x_i, r)$, i.e., we define

$$\mathrm{weight}(\mathcal{B}(x_i, r)) := \sum_{x_j \in X \cap \mathcal{B}(x_i, r)} d_j \,.$$

Radius Associated with a Point. According to [87], for each point $x_i \in X$, we define the value r_i to be the radius of the ball with center x_i that satisfies

$$\sum_{x_j \in X \cap \mathcal{B}(x_i, r_i)} d_j \cdot (r_i - \mathrm{D}(x_i, x_j)) = f_i \ . \tag{2.1}$$

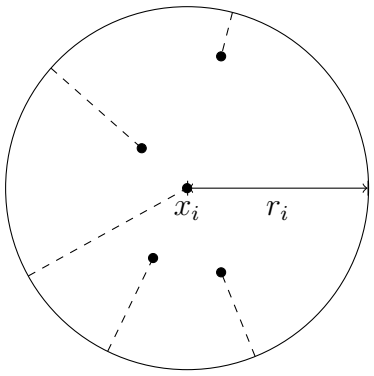


Figure 2.1: Illustration of $\sum_{x \in X \cap \mathcal{B}(x_i, r_i)} (r_i - \mathrm{D}(x_i, x))$ (in case of uniform demands with $d_j = 1$ for all $x_j \in X$). The dashed lines correspond to the distances summed up.

Figure 2.1 illustrates the definition of the radius r_i associated with a point x_i. Observe that the sum on the left hand side of Equation (2.1) is continuous and strictly monotonically increasing with r_i. Hence, there exists a unique value r_i satisfying the equation. Moreover, for any point $x_i \in X$, the radius r_i ranges between

$$r_{\min} := \frac{\min_{x_j \in X} f_j}{n \cdot \max_{x_j \in X} d_j} \quad \text{and} \quad r_{\max} := \frac{\max_{x_j \in X} f_j}{\min_{x_j \in X} d_j} \ .$$

The lower limit of the range is met if (*i*) $f_i = \min_{x_j \in X} f_j$, (*ii*) all the points in X are at the same position, and (*iii*) the demands of all the points are uniform such that $d_\ell = \max_{x_j \in X} d_j$ for any $\ell \in \{1, \ldots, n\}$. Because of Conditions (*ii*) and (*iii*), the contribution of each point $x_j \in X$ to the sum is $r_i \cdot \max_{x_j \in X} d_j$, which is the highest possible value. The upper limit of the range is met if (*i*) $f_i = \max_{x_j \in X} f_j$, (*ii*) x_i is the only point in the ball with radius r_i and center x_i, and (*iii*) $d_i = \min_{x_j \in X} d_j$. In this case, due to Condition (*ii*), the contribution of each point $x_j \in X \backslash \{x_i\}$ to the sum is 0, and, due to Condition (*iii*), the contribution of x_i is $r_i \cdot \min_{x_j \in X} d_j$, which is the lowest possible value.

The Algorithm. First, the Mettu and Plaxton algorithm computes for each point $x_i \in X$ its associated radius r_i. Then, it goes through all the points in X in non-decreasing order of their radii and opens a facility at a point $x_i \in X$ if x_i has no open facility in the ball with center x_i and radius $2r_i$. A pseudocode listing of the Mettu and Plaxton algorithm is given by Algorithm 2.3.1.

Let $\mathrm{FacLoc}^*(X)$ be the optimal facility location cost for X. Then, Mettu and Plaxton obtained the following result:

Algorithm 2.3.1 METTU-PLAXTON-FACLOC(X)

1: calculate the radius r_i for each point $x_i \in X$
2: sort all points in non-decreasing order according to their radii
3: let $x_1, x_2, \ldots, x_n$ be the sorted sequence
4: **for** $i \leftarrow 1$ **to** n **do**
5: **if** there is no open facility in $\mathcal{B}(x_i, 2 \cdot r_i)$ **then**
6: open facility at x_i

Theorem 1 ([87]). *Given any n-point metric space (X, D), algorithm* METTU-PLAXTON-FACLOC *computes a subset $F \subseteq X$ of open facilities such that we have*

$$\mathrm{FacLoc}(X, F) \leq 3 \cdot \mathrm{FacLoc}^*(X) \ .$$

The running time needed to compute F is $\mathcal{O}(n^2)$.

2.4 Computational Models

In this section, we will describe the computational models that we apply to measure the complexity of our algorithms. This includes the synchronous message passing model for algorithms working in a distributed setting, the kinetic data structure framework for algorithms working in a mobile setting, and data stream models. Before we will give an overview of these models, we will briefly describe the real random access machine model because, except for algorithms working in the synchronous message passing model, we measure the time and space complexities of our algorithms based on this model.

2.4.1 Real Random Access Machine Model

The *real random access machine* (*RAM*) *model* is a simplified and idealized model of a real computer, which is often used in computational geometry. In this model, a memory cell can store a real number and is called a *memory unit*. The set of allowed operations are

- arithmetic operations $(+, -, \cdot, \div)$,
- comparisons of two memory cells $(<, \leq, =, \neq, \geq, >)$, and
- some standard operations (raising a number to a given power[1], extracting a root[2], logarithmic calculus[3], trigonometric functions[4]).

[1] Our algorithms only raise natural numbers to a power greater than a small constant.
[2] We use an extraction of a root once in our distributed facility location algorithm for powers of metric spaces, once per embedding of a set of high-dimensional Euclidean points into a low-dimensional Euclidean space, and many times for our KDS to compute the points of intersection of two trajectories.
[3] Some of our algorithms compute a few values of the form $\lceil \log(x) \rceil$ for some real number $x > 1$. Since the running time of such an algorithm is $\Omega(\lceil \log(x) \rceil)$, the value $\lceil \log(x) \rceil$ can even be computed by linear search for the smallest $i \in \mathbb{N}_0$ such that $2^i \geq x$, with negligible increase in the running time.
[4] Our algorithms do not use trigonometric functions.

It is assumed that each of these allowed operations can be executed in a constant number of *time units*.

In the analysis of our algorithms, we assume that each coordinate of a point can be represented by using one memory unit, and the distance between two constant-dimensional points can be computed in a constant number of time units. These assumptions are commonly made in computational geometry. Unless otherwise stated, we measure the running time of our algorithms in time units and the space requirement in memory cells.

2.4.2 Synchronous Message Passing Model

The synchronous message passing model is well-known and one of the most frequently used models to design algorithms in a distributed setting [81, 98]. In this model, a network is an undirected graph, where the nodes are the processors and the edges are the bidirectional communication channels between the processors. Each node has a unique ID and knows the total number of nodes in the network but does not know the topology of the network.

At the beginning, the knowledge of a node about the network topology is limited to the neighbor nodes. To solve a given global problem, the nodes are allowed to communicate with each other. A global problem could be to solve the facility location problem on the network nodes, for instance. For sake of simplicity, the communication is assumed to be synchronous, i.e., there are globally defined communication rounds. In each such round, each node can send a message to each of its neighbors. In the process, the message sizes are bounded to B bits, where B is the bandwidth parameter of the network. Often, it is assumed that the bandwidth parameter is logarithmic in the number of nodes. In this way, each message can contain a constant number of node IDs (a.o. message sender and receiver).

The time complexity of a distributed algorithm that works in the synchronous message passing model is the number of required communication rounds.

2.4.3 Kinetic Data Structures

In 1999, Basch et al. [15] introduced the kinetic data structure (KDS) framework, which has been used as a central model for processing objects in motion ever since (see, e.g., [2, 15, 54] and the references therein). A KDS is a data structure that maintains a certain attribute of a set of continuously moving objects. For instance, in case of a facility location problem, this could be a set of open facilities that minimizes the facility location cost. The input of a KDS is a set of objects and a *flight plan*, i.e., each object moves continuously along a known trajectory. Furthermore, at any time, it is possible to change the flight plan by performing a so-called *flight plan update*, which means that one object changes its trajectory. The main idea is now that the continuous motion of the objects is utilized in a way that updates of the KDS take place only at discrete points of time and can be processed fast. As a result, a lot of computational effort can be saved by maintaining the KDS compared to handling just a series of instances of the corresponding static problem. To guarantee that the attribute is correct at any time, a KDS ensures that certain *certificates* are always

valid. Whenever a certificate fails, we call this an *event*, and an update is required. In case of a facility location problem, such an event occurs, for instance, when a client has moved so far away from all the open facilities that its connection cost exceeds the opening cost of a facility. To be able to handle each event at the correct time, an *event queue* is maintained.

There are four important properties to measure the quality of a KDS. The worst-case amount of time to process an update is called *responsiveness*. The second and third properties are *compactness* and *locality*. The compactness is given by the ratio between the maximum number of certificates ever present to prove the correctness of the attribute and the number of the moving objects. The locality addresses the maximum number of events in the event queue in which one object can be involved. As a result, the locality is a measure of how easily flight plan updates can be performed. The fourth property, the *efficiency* of a KDS, is the ratio between the worst-case total number of processed events and the worst-case number of processed events where the attribute changes. These worst-case numbers are specified under certain assumptions on the trajectories of the objects. Common assumptions are that the motions are linear or can be described by bounded-degree polynomials. A KDS is called *responsive*, *compact*, *local*, and *efficient*, respectively, if the associated value is at most polylogarithmic in the size of the input.

For a more detailed description of the concepts of a KDS, the reader is referred to [15, 54, 55].

2.4.4 Data Stream Models

A data stream consists of a long sequence of data items. The length of this sequence restricts the amount of resources that is available to process the data and the type of access to the data. In general, the amount of data is too large to be stored in main memory. Often it is even larger than the capacity of modern hard disks. As a result, the data has to be processed on the fly, and the only possible access to the data is sequential reading. Typical examples of data streams are network traffic data, measurements of sensor networks, or web crawls.

In order to design efficient algorithms for data streams, computer scientists have invented many different data stream models. In this section, we will provide a description of the two models considered in this thesis. These are the *insertion-only data stream model* and the *dynamic data stream model*, which are both frequently used in the field of geometry. For information about other data stream models and an overview of recent research, we refer the reader to [66, 92, 93].

Insertion-Only Data Stream Model

In the insertion-only data stream model[5], the input is a sequence (of insert operations) of points $p_1, \ldots, p_i, \ldots, p_n$ in worst-case order. As mentioned above, the type of access

[5]The insertion-only data stream model is a special type of the *cash register model* (confer [92]).

to the input points and the amount of resources to process them are restricted. More precisely, instead of having random access to the input points, which would be very time consuming, algorithms perform one sequential scan over the input stream that reads the points one by one in increasing order of the index i. Furthermore, it is only allowed to use space that is sublinear in the size of the input stream. To deal with these restrictions, streaming algorithms try to maintain, at any time, a summary of all the data seen so far. Such a summary is a small-space representation that fairly approximates the input data with respect to a given problem, i.e., a solution computed on the original input data can be approximated by using the small summary.

The complexity of a streaming algorithm is measured by its space requirement, its *update time* needed to process an element of the input stream, and its time needed to extract a solution for the given problem from the maintained summary. All of these three requirements are assumed to be only polylogarithmic in the size of the input stream.

Note that most of the streaming algorithms presented in this thesis have the property that they do not require extra time to extract a solution from the maintained summary since all necessary computations are done during an update. According to this, we will only specify the third complexity measure of those algorithms for which this property does not hold.

Dynamic Data Stream Model

The dynamic data stream model[6] is an extension of the insertion-only data stream model which also allows delete operations of points.

In this thesis, our focus is on a special type of this model which is called the *dynamic geometric data stream model*. This model was introduced by Indyk [64] and is defined as follows. The input is a sequence of m update operations on a point set $P \subseteq \{1, \ldots, \Delta\}^d$ in a discrete d-dimensional Euclidean space. At the beginning, the point set P is empty. For any point $p \in \{1, \ldots, \Delta\}^d$, the operation $\textsc{Insert}(p)$ inserts p into P, and, analogously, the operation $\textsc{Delete}(p)$ deletes p from P. We assume that the update operations occur in worst case order with the constraint that the stream is consistent, i.e., no point is removed that is not present in the current point set, and no point is added twice. Furthermore, we use n as an upper bound on the size of the current point set P. Obviously, we have $n \in \mathcal{O}(\Delta^d)$ and $n \leq m$.

Algorithms that work in the dynamic geometric data stream model are only allowed to perform one sequential scan over the input stream. The space requirement, the update time, and the time to extract a solution of the given problem from the maintained summary are each assumed to be only polylogarithmic in m and Δ and, therefore, in n since $n \leq m$.

[6] The dynamic data stream model is a special type of the *turnstile model* (confer [92]).

3 Facility Location in a Distributed Setting

This chapter addresses a randomized constant-factor approximation algorithm for the uniform metric facility location problem in a distributed setting. Our algorithm works in the synchronous message passing model where the underlying network is a clique with each node being a client as well as a potential location for a facility.

Our algorithm is based on two facts that Bădoiu et al. [14] discovered in case of the uniform metric facility location problem: (*i*) Given any point set X from a metric space, the sum of the radii defined by Mettu and Plaxton [87] is a constant-factor approximation of the optimal facility location cost for X, and (*ii*) for any facility $x_i \in X$, there exists a lower bound on the number of points located in the ball whose center is x_i and whose radius equals the radius of x_i. Using these two facts, we designed our randomized distributed algorithm in a way that it determines in three communication rounds, with message sizes bounded to $\mathcal{O}(\log(|X|))$ bits, a subset of the input points as open facilities such that, with high constant probability, the following condition is satisfied: The total opening cost is at most a constant factor larger than the sum of the radii and each facility $x_i \in X$ has an open facility in a ball whose center is x_i and whose radius is at most a constant factor larger than the radius of x_i. Thus, with high constant probability, our algorithm computes a constant-factor approximation of the uniform facility location problem for X.

Note that, in some settings, the transmission cost between two nodes is not linear in the distance. In radio networks, for example, it is a typical assumption that the energy required for transmitting a message via a certain distance is somewhere between the square and the cube of the distance. Motivated by this fact, we also extended our distributed algorithm to the uniform facility location problem for constant powers of metric spaces.

The remainder of this chapter is organized as follows. In Section 3.1, we specify the used synchronous message passing model and generalize the two facts mentioned above to the uniform facility location problem for powers of metric spaces. Our distributed algorithm for the uniform metric facility location problem is presented in Section 3.2. The extension to constant powers of metric spaces can be found in Section 3.3.

3.1 Preliminaries

In this chapter, we consider the uniform facility location problem for metric spaces and powers of metric spaces in a distributed setting. Given is a uniform opening cost f and a metric space (X, D), where $X = \{x_1, \ldots, x_n\}$ is a set of n points and D is a distance function defined on X. In the uniform facility location problem for powers of metric spaces, we are additionally given a constant metric exponent ℓ. Recall the definition of the facility location cost for both considered problems from Section 2.2.

We denote the cost of an optimal solution to the uniform metric facility location problem by $\mathrm{FacLoc}^*(X, f)$ and the cost of an optimal solution to the uniform facility location problem for powers of metric spaces by $\mathrm{FacLoc}^*(X, f, \ell)$.

3.1.1 The Distributed Setting

We consider the synchronous message passing model described in Section 2.4.2 where the communication network is a clique. This means that, in each communication round, each node can send a message to all other nodes. In the course of this, the message size is bounded to $\mathcal{O}(\log(n))$ bits. Furthermore, we assume that every node knows the distance to all other nodes, and each distance can be represented by $\mathcal{O}(\log(n))$ bits. Since we want to develop an approximation algorithm, we can always achieve this by appropriate rounding.

Note that although in our setting we allow all-to-all communication, it is not possible to solve the problem by accumulating all information at one node and then solve the problem with a classical (centralized) algorithm. The problem is that every node only knows the distance to its neighbors. Since every node receives $\mathcal{O}(n \log(n))$ bits of information in every communication round, it requires $\Omega(n)$ rounds to gather the information about all pairwise distances at a single node. As shown in [106], we essentially require all this information because it is not possible to compute a constant-factor approximation to the facility location problem (with uniform opening costs and demands) without looking at $\Omega(n^2)$ distances.

3.1.2 The Radii

Radius Associated with a Point. We extend the original definition of a radius associated with a point, given in Section 2.3, to powers of metric spaces. More precisely, for each point $x_i \in X$, we define the value r_i to be the radius of the ball with center x_i that satisfies

$$\sum_{x \in X \cap \mathcal{B}(x_i, r_i)} \left(r_i^\ell - \mathrm{D}(x_i, x)^\ell \right) = f \ . \tag{3.1}$$

Observe that there still exists only one solution to the radius r_i since the left hand side of Equation (3.1) is continuous and strictly monotonically increasing with r_i. For any $i \in \{1, \ldots, n\}$, we have $(f/n)^{1/\ell} \leq r_i \leq f^{1/\ell}$.

In case of uniform opening cost $f = 1$ and a metric exponent $\ell = 1$, Bădoiu et al. [14] discovered a useful relation between the value $\mathrm{weight}(\mathcal{B}(x_i, r_i))$ and the radius r_i. Their result can be generalized to any uniform opening cost $f \geq 0$ and any metric exponent $\ell \geq 1$. We obtain the following lemma:

Lemma 3.1.1. *For each $x_i \in X$, we have*

$$\mathrm{weight}\left(\mathcal{B}(x_i, r_i)\right) \geq \frac{f}{r_i^\ell} \ .$$

Proof. Due to the definition of r_i, we have

$$\sum_{x \in X \cap \mathcal{B}(x_i, r_i)} (r_i^\ell - \mathrm{D}(x_i, x)^\ell) = f \ ,$$

which implies

$$\sum_{x \in X \cap \mathcal{B}(x_i, r_i)} r_i^\ell \geq f \ .$$

Since $\mathrm{weight}(\mathcal{B}(x_i, r_i)) = |\{x \in X \mid x \in \mathcal{B}(x_i, r_i)\}|$, we obtain $\mathrm{weight}(\mathcal{B}(x_i, r_i)) \geq f / r_i^\ell$. □

Sum of the Radii. Bădoiu et al. [14] proved that the sum of the radii associated with the points in X is a good approximation of the optimal facility location cost for X. Again, their result can be generalized to any uniform opening cost $f \geq 0$ and any metric exponent $\ell \geq 1$.

In the proof of the generalized result, we use a modified version of the Mettu-Plaxton algorithm. More precisely, this version works exactly as Algorithm 2.3.1 except that, in the first step, it computes, for each point $x_i \in X$, the radius r_i that satisfies Equation (3.1), instead of the original radius proposed by Mettu and Plaxton [87]. We will first show that this modified Mettu-Plaxton algorithm is still a constant-factor approximation. Based on this result, we will then prove that the sum of the exponentiated radii approximates the optimal cost $\mathrm{FacLoc}^*(X, f, \ell)$ within a constant factor.

Let F_{MP} be the set of open facilities computed by the modified Mettu-Plaxton algorithm. In the following, we will show that $\mathrm{FacLoc}(X, F_{\mathrm{MP}}, f, \ell) \leq 3^\ell \cdot \mathrm{FacLoc}^*(X, f, \ell)$. The argumentation is basically the same as in [87]. Only a few minor adaptations to our scenario have been made.

Claim 3.1.2. *For any point $x_i \in X$, there exists an open facility $x_j \in F_{\mathrm{MP}}$ such that $r_j \leq r_i$ and $\mathrm{D}(x_i, x_j) \leq 2 \cdot r_i$.*

Proof. If there is no such open facility x_j with $r_j \leq r_i$ in $\mathcal{B}(x_i, 2 \cdot r_i)$, then we open a facility at x_i and x_i belongs to F_{MP}. □

Claim 3.1.3. *Let x_i and x_j be distinct open facilities in F_{MP}. Then, we have $\mathrm{D}(x_i, x_j) > 2 \cdot \max\{r_i, r_j\}$.*

Proof. Without loss of generality, we assume that $r_j \leq r_i$. It follows that $x_j \notin \mathcal{B}(x_i, 2 \cdot r_i)$. Otherwise, the point x_i would not be an open facility. Thus, we have

$$\mathrm{D}(x_i, x_j) > 2 \cdot r_i \geq 2 \cdot r_j \ .$$

□

For any point $x_j \in X$ and an arbitrary set of open facilities $F' \subseteq X$, let

$$\mathrm{charge}(x_j, F') := \mathrm{D}(x_j, F')^\ell + \sum_{x_i \in F'} \max\{0, r_i^\ell - \mathrm{D}(x_i, x_j)^\ell\} \ .$$

Claim 3.1.4. *For an arbitrary set of open facilities $F' \subseteq X$, we have*

$$\sum_{x_j \in X} \text{charge}(x_j, F') = \text{FacLoc}(X, F', f, \ell) \ .$$

Proof. Due to the definition of charge$(\cdot,\cdot)$ and Equation (3.1), we get

$$\begin{aligned}
&\sum_{x_j \in X} \text{charge}(x_j, F') \\
= &\sum_{x_j \in X} \mathrm{D}(x_j, F')^\ell + \sum_{x_j \in X} \sum_{x_i \in F'} \max\{0, r_i^\ell - \mathrm{D}(x_i, x_j)^\ell\} \\
= &\sum_{x_j \in X} \mathrm{D}(x_j, F')^\ell + \sum_{x_i \in F'} \sum_{x_j \in X \cap \mathcal{B}(x_i, r_i)} (r_i^\ell - \mathrm{D}(x_i, x_j)^\ell) \\
= &\sum_{x_j \in X} \mathrm{D}(x_j, F')^\ell + \sum_{x_i \in F'} f \\
= &\ \text{FacLoc}(X, F', f, \ell) \ .
\end{aligned}$$

□

Claim 3.1.5. *Let $x_j \in X$ be any point, let $F' \subseteq X$ be an arbitrary set of open facilities, and let $x_i \in F'$ be any open facility. If we have $\mathrm{D}(x_j, x_i) = \mathrm{D}(x_j, F')$, then* charge$(x_j, F') \geq \max\{r_i^\ell, \mathrm{D}(x_j, x_i)^\ell\}$.

Proof. If $x_j \notin \mathcal{B}(x_i, r_i)$, then

$$\begin{aligned}
\text{charge}(x_j, F') &\geq \mathrm{D}(x_j, F')^\ell \\
&= \mathrm{D}(x_j, x_i)^\ell \\
&> r_i^\ell \ .
\end{aligned}$$

Otherwise, we have

$$\begin{aligned}
\text{charge}\left(x_j, F'\right) &\geq \mathrm{D}\left(x_j, F'\right)^\ell + \left(r_i^\ell - \mathrm{D}(x_j, x_i)^\ell\right) \\
&= \mathrm{D}\left(x_j, x_i\right)^\ell + \left(r_i^\ell - \mathrm{D}(x_j, x_i)^\ell\right) \\
&= r_i^\ell \\
&\geq \mathrm{D}(x_j, x_i)^\ell \ .
\end{aligned}$$

□

Claim 3.1.6. *Let $x_j \in X$ be any point, and let x_i be any open facility in F_{MP}. If $x_j \in \mathcal{B}(x_i, r_i)$, then* charge$(x_j, F_{\text{MP}}) \leq r_i^\ell$.

Proof. By Claim 3.1.3, there is no open point $x_m \in F_{\text{MP}}$ such that we have $i \neq m$ and $x_j \in \mathcal{B}(x_m, r_m)$. Since $\mathrm{D}(x_j, F_{\text{MP}}) \leq \mathrm{D}(x_j, x_i)$, we obtain

$$\begin{aligned}
\text{charge}\left(x_j, F_{\text{MP}}\right) &= \mathrm{D}\left(x_j, F_{\text{MP}}\right)^\ell + \left(r_i^\ell - \mathrm{D}\left(x_j, x_i\right)^\ell\right) \\
&\leq \mathrm{D}\left(x_j, x_i\right)^\ell + \left(r_i^\ell - \mathrm{D}\left(x_j, x_i\right)^\ell\right) \\
&= r_i^\ell \ .
\end{aligned}$$

□

Claim 3.1.7. *Let $x_j \in$ be any point, and let x_i be any open facility in F_{MP}. If $x_j \notin \mathcal{B}(x_i, r_i)$, then we have* $\mathrm{charge}(x_j, F_{\mathrm{MP}}) \leq \mathrm{D}(x_j, x_i)^\ell$.

Proof. The correctness of the claim follows immediately, unless there is an open facility $x_m \in F_{\mathrm{MP}}$ such that $x_j \in \mathcal{B}(x_m, r_m)$. If such an open facility x_m exists, then Claims 3.1.3 and 3.1.6 imply $\mathrm{D}(x_i, x_m) > 2 \cdot \max\{r_i, r_m\}$ and $\mathrm{charge}(x_j, F_{\mathrm{MP}}) \leq r_m^\ell$. Furthermore, by triangle inequality, we obtain

$$\begin{aligned} \mathrm{D}(x_j, x_i) &\geq \mathrm{D}(x_i, x_m) - \mathrm{D}(x_j, x_m) \\ &> 2r_m - r_m \\ &= r_m \ , \end{aligned}$$

which proves $\mathrm{charge}(x_j, F_{\mathrm{MP}}) \leq r_m^\ell \leq \mathrm{D}(x_j, x_i)^\ell$. □

Claim 3.1.8. *For any point $x_j \in X$ and an arbitrary set of open facilities $F' \subseteq X$, we have* $\mathrm{charge}(x_j, F_{\mathrm{MP}}) \leq 3^\ell \cdot \mathrm{charge}(x_j, F')$.

Proof. Let x_i be some open facility in F' such that we have $\mathrm{D}(x_j, x_i) = \mathrm{D}(x_j, F')$. By Claim 3.1.2, there exists a facility $x_m \in F_{\mathrm{MP}}$ such that we have $r_m \leq r_i$ and $\mathrm{D}(x_i, x_m) \leq 2 \cdot r_i$.

If $x_j \in \mathcal{B}(x_m, r_m)$, then we get $\mathrm{charge}(x_j, F_{\mathrm{MP}}) \leq r_m^\ell$ by Claim 3.1.6. Since Claim 3.1.5 implies $\mathrm{charge}(x_j, F') \geq r_i^\ell$, we can conclude

$$\begin{aligned} \mathrm{charge}(x_j, F_{\mathrm{MP}}) &\leq r_m^\ell \\ &\leq r_i^\ell \\ &\leq \mathrm{charge}(x_j, F') \ . \end{aligned}$$

This proves the assertion in case that we have $x_j \in \mathcal{B}(x_m, r_m)$.

If $x_j \notin \mathcal{B}(x_m, r_m)$, then $\mathrm{charge}(x_j, F_{\mathrm{MP}}) \leq \mathrm{D}(x_j, x_m)^\ell$ by Claim 3.1.7. Thus, by triangle inequality, we get

$$\begin{aligned} \mathrm{charge}(x_j, F_{\mathrm{MP}}) &\leq \mathrm{D}(x_j, x_m)^\ell \\ &\leq \left(\mathrm{D}(x_j, x_i) + \mathrm{D}(x_i, x_m)\right)^\ell \\ &\leq \left(\mathrm{D}(x_j, x_i) + 2 \cdot r_i\right)^\ell \\ &\leq 3^\ell \cdot \max\{\mathrm{D}(x_j, x_i)^\ell, r_i^\ell\} \ . \end{aligned}$$

Now, the assertion follows by Claim 3.1.5. □

Lemma 3.1.9. $\mathrm{FacLoc}(X, F_{\mathrm{MP}}, f, \ell) \leq 3^\ell \cdot \mathrm{FacLoc}^*(X, f, \ell)$

Proof. The assertion follows from Lemmas 3.1.4 and 3.1.8. □

Based on the results above, we can prove the following lemma:

Lemma 3.1.10.

$$\frac{1}{2^{\ell+1}} \cdot \mathrm{FacLoc}^*(X, f, \ell) \leq \sum_{x_i \in X} r_i^\ell \leq 6^\ell \cdot \mathrm{FacLoc}^*(X, f, \ell)$$

Proof. We first prove the lower bound and then the upper bound. The argumentation is basically the same as in [14]. Only a few minor adaptations to our scenario have been made.

Lower bound: Let F_{MP} be the set of open facilities computed by the modified Mettu-Plaxton algorithm. Then, it follows from Claim 3.1.2 that

$$2^\ell \cdot \sum_{x_i \in X} r_i^\ell \geq \sum_{x_i \in X} \mathrm{D}(x_i, F_{\mathrm{MP}})^\ell \ . \tag{3.2}$$

Next, we show that we also have

$$2^\ell \cdot \sum_{x_i \in X} r_i^\ell \geq f \cdot |F_{\mathrm{MP}}| \ . \tag{3.3}$$

Due to Claim 3.1.3, each point $x_i \in X$ is contained in at most one ball $\mathcal{B}(x_j, r_j)$ for some open facility $x_j \in F_{\mathrm{MP}}$. Furthermore, observe that, for any point $x_m \in \mathcal{B}(x_j, r_j)$, we must have $r_j \leq 2 \cdot r_m$. Otherwise, we would have

$$x_m \in \mathcal{B}(x_m, 2 \cdot r_m) \subseteq \mathcal{B}(x_m, r_j) \subseteq \mathcal{B}(x_m, r_j + \mathrm{D}(x_j, x_m)) \subseteq \mathcal{B}(x_j, 2 \cdot r_j) \ ,$$

and the modified Mettu-Plaxton algorithm would not open a facility at x_j, which is a contradiction. Hence, we obtain

$$\begin{aligned}
\sum_{x_i \in X} r_i^\ell &\geq \sum_{x_j \in F_{\mathrm{MP}}} \sum_{x_m \in X \cap \mathcal{B}(x_j, r_j)} r_m^\ell \\
&\geq \sum_{x_j \in F_{\mathrm{MP}}} \sum_{x_m \in X \cap \mathcal{B}(x_j, r_j)} \left(\frac{r_j}{2}\right)^\ell \\
&= \frac{1}{2^\ell} \cdot \sum_{x_j \in F_{\mathrm{MP}}} \sum_{x_m \in X \cap \mathcal{B}(x_j, r_j)} r_j^\ell \\
&\geq \frac{1}{2^\ell} \cdot \sum_{x_j \in F_{\mathrm{MP}}} f \\
&= \frac{1}{2^\ell} \cdot f \cdot |F_{\mathrm{MP}}| \ ,
\end{aligned}$$

which proves Inequality (3.3). Due to Inequalities (3.2) and (3.3), we get

$$\begin{aligned}
2^{\ell+1} \cdot \sum_{x_i \in X} r_i^\ell &\geq f \cdot |F_{\mathrm{MP}}| + \sum_{x_i \in X} \mathrm{D}(x_i, F_{\mathrm{MP}})^\ell \\
&= \mathrm{FacLoc}(X, F_{\mathrm{MP}}, f, \ell) \\
&\geq \mathrm{FacLoc}^*(X, f, \ell) \ .
\end{aligned}$$

Upper bound: Due to Lemma 3.1.9, we know that

$$\mathrm{FacLoc}(X, F_{\mathrm{MP}}, f, \ell) \leq 3^{\ell} \cdot \mathrm{FacLoc}^{*}(X, f, \ell) \ .$$

Thus, to prove the upper bound, it remains to show that

$$\sum_{x_i \in X} r_i^{\ell} \ \leq \ 2^{\ell} \cdot \mathrm{FacLoc}(X, F_{\mathrm{MP}}, f, \ell) \ .$$

Due to Claim 3.1.4, we have

$$\begin{aligned} 2^{\ell} \cdot \mathrm{FacLoc}(X, F_{\mathrm{MP}}, f, \ell) \ &= \ 2^{\ell} \cdot \sum_{x_i \in X} \mathrm{charge}(x_i, F_{\mathrm{MP}}) \\ &\geq \ 2^{\ell} \cdot \left(\sum_{x_i \in F_{\mathrm{MP}}} r_i^{\ell} + \sum_{x_j \in X \setminus F_{\mathrm{MP}}} \max\{r_{\delta(j)}^{\ell}, \mathrm{D}(x_j, x_{\delta(j)})^{\ell}\} \right) \ , \end{aligned}$$

where $\delta(j)$ denotes the index of the facility in F_{MP} that is closest to x_j. Thus, if we can show that

$$2^{\ell} \cdot \left(\sum_{x_i \in F_{\mathrm{MP}}} r_i^{\ell} + \sum_{x_j \in X \setminus F_{\mathrm{MP}}} \max\{r_{\delta(j)}^{\ell}, \mathrm{D}(x_j, x_{\delta(j)})^{\ell}\} \right) \geq \sum_{x_i \in X} r_i^{\ell} \ , \tag{3.4}$$

then we are done. It is sufficient to prove

$$r_j^{\ell} \ \leq \ 2^{\ell-1} \cdot \left(\mathrm{D}(x_j, x_{\delta(j)})^{\ell} + r_{\delta(j)}^{\ell} \right) \tag{3.5}$$

because this implies $\max\{r_{\delta(j)}^{\ell}, \mathrm{D}(x_j, x_{\delta(j)})^{\ell}\} \geq r_j^{\ell}/2^{\ell}$ and Inequality (3.4) follows. We prove the correctness of Inequality (3.5) by contradiction. Hence, we assume that

$$r_j^{\ell} > 2^{\ell-1} \cdot \left(\mathrm{D}(x_j, x_{\delta(j)})^{\ell} + r_{\delta(j)}^{\ell}) \right) \ .$$

We can easily prove by induction that $2^{\ell-1} \cdot (a^{\ell} + b^{\ell}) \geq (a+b)^{\ell}$ for any $a, b \geq 0$. Thus, we obtain

$$r_j^{\ell} > \left(\mathrm{D}(x_j, x_{\delta(j)}) + r_{\delta(j)}) \right)^{\ell} \ ,$$

which, in turn, would imply $\mathcal{B}(x_{\delta(j)}, r_{\delta(j)}) \subseteq \mathcal{B}(x_j, r_j)$. Furthermore, by applying triangle inequality and $2^{\ell-1} \cdot (a^{\ell} + b^{\ell}) \geq (a+b)^{\ell}$ for an $a, b \geq 0$, we get

$$\begin{aligned} \mathrm{D}(x_j, x_m)^{\ell} \ &\leq \ \left(\mathrm{D}(x_j, x_{\delta(j)}) + \mathrm{D}(x_{\delta(j)}, x_m) \right)^{\ell} \\ &\leq \ 2^{\ell-1} \cdot \left(\mathrm{D}(x_j, x_{\delta(j)})^{\ell} + \mathrm{D}(x_{\delta(j)}, x_m)^{\ell} \right) \end{aligned}$$

as upper bound on the exponentiated distance between x_j and any point $x_m \in \mathcal{B}(x_{\delta(j)}, r_{\delta(j)})$. Now, we obtain

$$\begin{aligned}
& \sum_{x_m \in X \cap \mathcal{B}(x_j, r_j)} r_j^\ell - \mathrm{D}(x_j, x_m)^\ell \\
\geq & \sum_{x_m \in X \cap \mathcal{B}(x_{\delta(j)}, r_{\delta(j)})} r_j^\ell - \mathrm{D}(x_j, x_m)^\ell \\
> & \sum_{x_m \in X \cap \mathcal{B}(x_{\delta(j)}, r_{\delta(j)})} 2^{\ell-1} \cdot \left(\mathrm{D}(x_j, x_{\delta(j)})^\ell + r_{\delta(j)}^\ell\right) - 2^{\ell-1} \cdot \left(\mathrm{D}(x_j, x_{\delta(j)})^\ell + \mathrm{D}(x_{\delta(j)}, x_m)^\ell\right) \\
= & \ 2^{\ell-1} \cdot \sum_{x_m \in X \cap \mathcal{B}(x_{\delta(j)}, r_{\delta(j)})} r_{\delta(j)}^\ell - \mathrm{D}(x_{\delta(j)}, x_m)^\ell \\
= & \ 2^{\ell-1} \cdot f \\
\geq & \ f \ ,
\end{aligned}$$

which is a contradiction because the definition of r_j requires

$$\sum_{x_m \in X \cap \mathcal{B}(x_j, r_j)} r_j^\ell - \mathrm{D}(x_j, x_m)^\ell \ = \ f \ .$$

It follows that Inequality (3.5) is true, which was the only thing left to prove the assertion of the lemma. □

3.2 Distributed Algorithm for Metric Spaces

Our distributed algorithm consists of three parts (see Algorithm 3.2.1 for a description in pseudocode). Recall that we assume that each point knows its distance to all the other points. At the beginning of the first part, each point $x_i \in X$ creates a $(\lceil \log(n) \rceil + 1)$-bit array. These bits are used to decide whether a point should open a facility or not. In the following, we will call these bits *phase bits*. The values of these phase bits are chosen at random so that, for each $k \in \{0, 1, \ldots, \lceil \log(n) \rceil\}$, the k-th phase bit is 1 with probability $\min\{2^k/n, 1\}$ and 0 otherwise. Finally, every point sends its $\lceil \log(n) \rceil + 1$ phase bits to all the other points.

The second part of the algorithm is organized in $\lceil \log(n) \rceil + 1$ phases. During these phases, each point decides locally, based on the phase bits, if it should open a facility or connect itself to another open facility. This is accomplished as follows: Consider the k-th phase of point x_i. The algorithm opens a facility at this point if the k-th phase bit is 1 and the first $k-1$ phase bits of all the other points at a distance of at most $2^k \cdot f/n$ from x_i are 0. Otherwise, if there exists a point x_j at a distance of at most $2^k \cdot f/n$ from x_i which has a 1 among the first $k-1$ phase bits, the algorithm *tentatively connects* x_i to the point x_j. In the final solution, x_i will be connected to the nearest open facility (which might differ from x_j). Note that if neither the k-th phase bit of x_i is 1 nor there exists a point x_j at a distance of at most $2^k \cdot f/n$ from x_i which has a 1 among the first $k-1$ phase bits, x_i

does nothing in phase k. At the end of the last phase, every point knows whether it is an open facility or not because the $\lceil \log(n) \rceil$-th phase bit of every point is 1 with probability $\min\{2^{\lceil \log(n) \rceil}/n, 1\} = 1$. Finally, each point broadcasts whether it is an open facility or not.

In the last part of the algorithm, every point that is not an open facility sends a request of connection to the nearest open facility.

We will show in the next section that, with high constant probability, the total opening cost for the facilities is at most a constant factor larger than the sum of the radii, and any client $x_i \in X$ has at least one open facility in the ball $\mathcal{B}(x_i, cr_i)$, where c is some small constant. Since the sum of the radii is a constant-factor approximation of the optimal facility location cost (see Lemma 3.1.10), this implies that, with high constant probability, our distributed algorithm computes a constant factor-approximation for the uniform metric facility location problem.

Algorithm 3.2.1 Local Algorithm for Point x_i

1: open$[i] \leftarrow$ FALSE
2: **for** $k \leftarrow 0$ **to** $\lceil \log(n) \rceil$ **do**
3: $\quad \varphi_i[k] \leftarrow \begin{cases} 1 & \text{, with probability } \min\{2^k/n, 1\} \\ 0 & \text{, otherwise} \end{cases}$
4: **send** φ_i to all $x_j \in X,\ j \neq i$
5: **receive** φ_j from all $x_j \in X,\ j \neq i$
6: **for** $k \leftarrow 0$ **to** $\lceil \log(n) \rceil$ **do**
7: $\quad$ **if** $\varphi_i[k] = 1$ **and** for each point $x_j \in \mathcal{B}(x_i, 2^k \cdot f/n)$, $x_j \neq x_i$, we have $\varphi_j[m] = 0$ for all $m < k$ **then**
8: $\quad\quad$ open$[i] \leftarrow$ TRUE
9: **send** open$[i]$ to all $x_j \in X,\ j \neq i$
10: **receive** open$[j]$ from all $j \in X,\ j \neq i$
11: **if** open$[i]$ = FALSE **then**
12: $\quad$ **connect** to the nearest open facility

3.2.1 Analysis of the Algorithm

In this section, we show that our distributed algorithm produces a solution for the uniform metric facility location problem whose cost are with high constant probability at most a constant factor larger than the optimal cost.

To simplify the analysis, we do not use the exact value r_i satisfying Equation (3.1) for $\ell = 1$ but the value $\tilde{r}_i := 2^j \cdot f/n$ where j is the smallest integer that satisfies the inequality

$$\sum_{x \in X \cap \mathcal{B}(x_i, \tilde{r}_i)} (\tilde{r}_i - \mathrm{D}(x_i, x)) \geq f \ .$$

First, we give an upper bound on the expected opening cost of any point $x_i \in X$.

Lemma 3.2.1. *Let $x_i \in X$ be any point. Then, the expected opening cost of x_i is $\mathcal{O}(r_i)$.*

Proof. At first, we estimate the probability that the algorithm opens a facility at x_i in any phase $k \in \{0, 1, \ldots, \lceil \log(n) \rceil\}$. Recall that this happens if the k-th phase bit of x_i is 1 and the first $k-1$ phase bits of all the other points at a distance of at most $2^k \cdot f/n$ are 0. Let $Y_{i,k}$ be the indicator random variable for the event that the algorithm opens a facility at x_i in phase k. We now consider the two cases $k < j$ and $j \leq k \leq \lceil \log(n) \rceil$ with $j = \log(\tilde{r}_i \cdot n/f)$.

Case $j \leq k \leq \lceil \log(n) \rceil$: The k-th phase bit of x_i is 1 with probability $\min\{2^k/n, 1\} \leq 2^k/n$. Furthermore, for any phase $m < k$, the m-th phase bit of an arbitrary point in $\mathcal{B}(x_i, 2^k \cdot f/n)$ is 0 with probability $1 - 2^m/n$. Hence, the probability that all of the first $k-1$ phase bits of this point in $\mathcal{B}(x_i, 2^k \cdot f/n)$ are 0 is $\prod_{m=0}^{k-1} 1 - 2^m/n$. Thus, we have

$$\mathbf{Pr}\left[Y_{i,k} = 1\right] \leq \frac{2^k}{n} \cdot \left[\left(1 - \frac{2^0}{n}\right) \cdot \left(1 - \frac{2^1}{n}\right) \cdot \ldots \cdot \left(1 - \frac{2^{k-1}}{n}\right)\right]^{\text{weight}\left(\mathcal{B}\left(x_i, 2^k \cdot \frac{f}{n}\right)\right)} .$$

Observe that $\tilde{r}_i \geq r_i$. By applying Lemma 3.1.1 with $\ell = 1$, we obtain that

$$\begin{aligned} \text{weight}\left(\mathcal{B}\left(x_i, 2^k \cdot \frac{f}{n}\right)\right) &\geq \text{weight}\left(\mathcal{B}\left(x_i, 2^j \cdot \frac{f}{n}\right)\right) \\ &= \text{weight}\left(\mathcal{B}(x_i, \tilde{r}_i)\right) \\ &\geq \text{weight}\left(\mathcal{B}(x_i, r_i)\right) \\ &\geq \frac{f}{r_i} \\ &\geq \frac{f}{\tilde{r}_i} . \end{aligned}$$

Thus, we get

$$\begin{aligned} \mathbf{Pr}\left[Y_{i,k} = 1\right] &\leq \frac{2^k}{n} \cdot \left[\left(1 - \frac{2^0}{n}\right) \cdot \left(1 - \frac{2^1}{n}\right) \cdot \ldots \cdot \left(1 - \frac{2^{k-1}}{n}\right)\right]^{\frac{f}{\tilde{r}_i}} \\ &= \frac{2^k}{n} \cdot \left[\left(1 - \frac{2^0}{n}\right) \cdot \left(1 - \frac{2^1}{n}\right) \cdot \ldots \cdot \left(1 - \frac{2^{k-1}}{n}\right)\right]^{\frac{n}{2^j}} \\ &\leq \frac{2^k}{n} \cdot \left(1 - \frac{2^{k-1}}{n}\right)^{\frac{n}{2^j}} . \end{aligned}$$

Now, let m denote the non-negative integer $k - j$. Then, we obtain

$$\begin{aligned} \mathbf{Pr}\left[Y_{i,k} = 1\right] &\leq \frac{2^{j+m}}{n} \cdot \left(1 - \frac{2^{j+m-1}}{n}\right)^{\frac{n}{2^j}} \\ &= \frac{2^{j+m}}{n} \cdot \left(1 - \frac{2^{j+m-1}}{n}\right)^{\frac{n}{2^{j+m-1}} \cdot 2^{m-1}} \\ &\leq \frac{2^{j+m}}{n} \cdot \left(\frac{1}{e}\right)^{2^{m-1}} , \end{aligned}$$

where the last inequality is due to a bound on Euler's number (see Inequality (B.2)).

Case $k < j$: An upper bound on the probability that the algorithm opens a facility at x_i in a phase $k < j$ is $2^k/n$. Hence, we have

$$\mathbf{Pr}\left[Y_{i,k} = 1\right] \leq \frac{2^k}{n} \ .$$

Let Y_i be the indicator random variable for the event that the algorithm opens a facility at x_i. Then, the expected opening cost of point x_i are upper bounded by

$$\begin{aligned}
f \cdot \mathbf{E}\left[Y_i\right] &= f \cdot \mathbf{E}\left[\sum_{k=0}^{\lceil \log(n) \rceil} Y_{i,k}\right] \\
&= f \cdot \sum_{k=0}^{\lceil \log(n) \rceil} \mathbf{E}\left[Y_{i,k}\right] \\
&= f \cdot \sum_{k=0}^{\lceil \log(n) \rceil} \mathbf{Pr}\left[Y_{i,k} = 1\right] \\
&\leq f \cdot \sum_{k=0}^{j-1} \frac{2^k}{n} + f \cdot \sum_{m=0}^{\lceil \log(n) \rceil - j} \frac{2^{j+m}}{n} \cdot \left(\frac{1}{e}\right)^{2^{m-1}} \\
&= f \cdot \frac{2^j - 1}{n} + f \cdot \frac{2^{j+1}}{n} \cdot \sum_{m=0}^{\lceil \log(n) \rceil - j} \left(\frac{1}{e}\right)^{2^{m-1}} \cdot 2^{m-1} \\
&\leq f \cdot \frac{2^j - 1}{n} + f \cdot \frac{2^{j+1}}{n} \cdot \sum_{m=0}^{\lceil \log(n) \rceil - j} 2^{-m+1} \\
&\in \mathcal{O}(\tilde{r}_i) \ ,
\end{aligned}$$

where the last inequality follows from the easily provable fact that

$$\left(\frac{1}{e}\right)^{2^{m-1}} \cdot 2^{m-1} \leq 2^{-m+1}$$

for all $m \geq 0$. Finally, due to the definition of $\tilde{r}_i$, the expected opening cost of the point x_i is $\mathcal{O}(r_i)$. □

The proof of our upper bound on the final connection cost of any point $x_i \in X$ utilizes the following lemma:

Lemma 3.2.2. *Let $x_i \in X$ be any point that has been chosen as open facility or that has tentatively been connected in any phase $k \in \{0, \ldots, \lceil \log(n) \rceil\}$. Then, the distance of x_i to the nearest open facility is at most $2^{k+1} \cdot f/n$.*

Proof. Obviously, if we open a facility at x_i in phase k, then the distance of x_i to the nearest open facility is $0 \leq 2^{k+1} \cdot f/n$. Next, we consider the case that x_i has been connected tentatively. Note that, due to our construction, a point cannot be connected tentatively

in phase 0. Thus, in the following, we will assume that x_i has tentatively been connected to a point $x_j \in X$ in a phase $k \in \{1, \ldots, \lceil \log(n) \rceil\}$. It follows that the distance from x_i to x_j is at most $2^k \cdot f/n$. Let m denote the smallest number of a phase bit of x_j whose value is 1. Since x_i has tentatively been connected to x_j in phase k, we have $m \leq k-1$. Now, either x_j is open or tentatively connected. If it is open, then the distance from x_i to the nearest open facility is at most $2^k \cdot f/n$ and we are done. Otherwise, x_j has tentatively been connected to another point within a distance of at most $2^m \cdot f/n \leq 2^{k-1} \cdot f/n$. Recursively applying this argument (see also Figure 3.1) yields that there must be an open facility within a distance of at most

$$2^k \cdot \frac{f}{n} + \sum_{m=0}^{k-1} 2^m \cdot \frac{f}{n} \leq 2^{k+1} \cdot \frac{f}{n}$$

from x_i. □

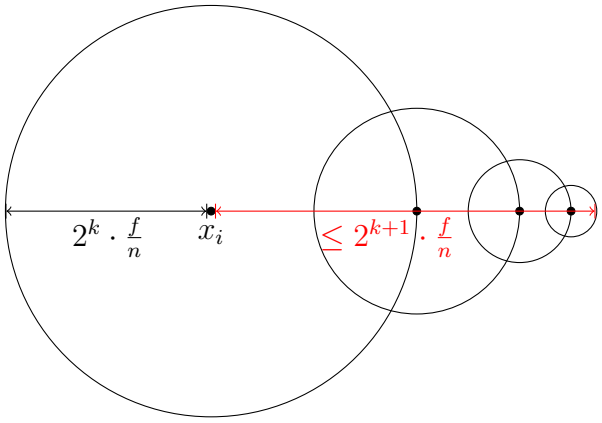


Figure 3.1: Connecting x_i to an open facility over a chain of tentatively connected points.

Lemma 3.2.3. *Let $x_i \in X$ be any point. Then, the expected final connection cost of x_i is $\mathcal{O}(r_i)$.*

Proof. Due to our construction, a point cannot be connected tentatively in phase 0. Thus, in phase 0, the algorithm either opens a facility at x_i or does nothing with x_i. If it opens a facility at x_i, then the connection cost of x_i is obviously 0. Due to Lemma 3.2.2, if the point x_i has been chosen as an open facility or has tentatively been connected in a phase $k \in \{1, \ldots, \lceil \log(n) \rceil\}$, then its final connection cost is at most $2^{k+1} \cdot f/n$. Now, for each $k \in \{0, \ldots, \lceil \log(n) \rceil\}$, let $Z_{i,k}$ be the indicator random variable for the event that the algorithm has not opened a facility at x_i and has not tentatively connected x_i up to and

including phase k. Then, we can upper bound the expected final connection cost of x_i by

$$\begin{aligned}
& \sum_{k=1}^{\lceil \log(n) \rceil} 2^{k+1} \cdot \frac{f}{n} \cdot \mathbf{Pr}\left[x_i \text{ is opened or tentatively connected in phase } k\right] \\
= & \sum_{k=1}^{\lceil \log(n) \rceil} 2^{k+1} \cdot \frac{f}{n} \cdot \left(\mathbf{Pr}\left[Z_{i,k-1} = 1\right] - \mathbf{Pr}\left[Z_{i,k} = 1\right]\right) \\
= & \; 2 \cdot \frac{f}{n} \cdot \mathbf{Pr}\left[Z_{i,0} = 1\right] - 2^{\lceil \log(n) \rceil + 1} \cdot \frac{f}{n} \cdot \mathbf{Pr}\left[Z_{i,\lceil \log(n) \rceil} = 1\right] \\
& + \sum_{k=0}^{\lceil \log(n) \rceil - 1} 2^{k+1} \cdot \frac{f}{n} \cdot \mathbf{Pr}\left[Z_{i,k} = 1\right] \\
= & \; 2 \cdot \frac{f}{n} \cdot \mathbf{Pr}\left[Z_{i,0} = 1\right] + \sum_{k=0}^{\lceil \log(n) \rceil - 1} 2^{k+1} \cdot \frac{f}{n} \cdot \mathbf{Pr}\left[Z_{i,k} = 1\right] \quad ,
\end{aligned}$$

where the last equality follows from $\mathbf{Pr}\left[Z_{i,\lceil \log(n) \rceil} = 1\right] = 0$. Thus, to upper bound the expected final connection cost of x_i, we have to upper bound the probabilities $\mathbf{Pr}\left[Z_{i,k} = 1\right]$. We consider the two cases $k < j$ and $j \leq k < \lceil \log(n) \rceil$ with $j = \log(\tilde{r}_i \cdot n/f)$.

Case $j \leq k < \lceil \log(n) \rceil$: Observe that $Z_{i,k} = 1$ if the first k phase bits of x_i are 0, and the first $k-1$ phase bits of all the other points at a distance of at most $2^k \cdot f/n$ are also 0. For any phase $m \leq k$, the m-th phase bit of x_i is 0 with probability $1 - 2^m/n$. Hence, the probability that all of the first k phase bits of x_i are 0 is $\prod_{m=0}^{k} 1 - 2^m/n$. Similarly, the probability that all of the first $k-1$ phase bits of any point in $\mathcal{B}(x_i, 2^k \cdot f/n)$ are 0 is $\prod_{m=0}^{k-1} 1 - 2^m/n$. As proven in Lemma 3.2.1, the number of points in $\mathcal{B}(x_i, 2^k \cdot f/n)$ is lower bounded by

$$\text{weight}\left(\mathcal{B}\left(x_i, 2^k \cdot \frac{f}{n}\right)\right) \geq \frac{f}{\tilde{r}_i} \quad .$$

It follows that

$$\begin{aligned}
\mathbf{Pr}\left[Z_{i,k} = 1\right] & \leq \left[\left(1 - \frac{2^0}{n}\right) \cdot \ldots \cdot \left(1 - \frac{2^k}{n}\right)\right] \cdot \left[\left(1 - \frac{2^0}{n}\right) \cdot \ldots \cdot \left(1 - \frac{2^{k-1}}{n}\right)\right]^{\frac{f}{\tilde{r}_i}} \\
& \leq \left(1 - \frac{2^k}{n}\right) \cdot \left(1 - \frac{2^{k-1}}{n}\right)^{\frac{n}{2^j}} \quad .
\end{aligned}$$

Let m denote the non-negative integer $k - j$. Then, we have

$$\begin{aligned}
\mathbf{Pr}\left[Z_{i,k} = 1\right] & \leq \left(1 - \frac{2^{j+m}}{n}\right) \cdot \left(1 - \frac{2^{j+m-1}}{n}\right)^{\frac{n}{2^{j+m-1}} \cdot 2^{m-1}} \\
& \leq \left(1 - \frac{2^{j+m}}{n}\right) \cdot \left(\frac{1}{e}\right)^{2^{m-1}} \leq \left(\frac{1}{e}\right)^{2^{m-1}} \quad ,
\end{aligned}$$

where the second inequality is due to a bound on Euler's number (see Inequality (B.2)).

Case $k < j$: Obviously, an upper bound on the probability that the algorithm does not open a facility at x_i or tentatively connect x_i up to and including phase k is 1. Hence, we get

$$\mathbf{Pr}\left[Z_{i,k}\right] \leq 1 \ .$$

Based on the above two cases, we can upper bound the expected final connection cost of x_i by

$$\begin{aligned}
& 2 \cdot \frac{f}{n} \cdot \mathbf{Pr}\left[Z_{i,0} = 1\right] + \sum_{k=0}^{\lceil \log(n) \rceil - 1} 2^{k+1} \cdot \frac{f}{n} \cdot \mathbf{Pr}\left[Z_{i,k} = 1\right] \\
\leq \ & 2 \cdot \frac{f}{n} \cdot 1 + \sum_{k=0}^{j-1} 2^{k+1} \cdot \frac{f}{n} \cdot 1 + \sum_{m=0}^{\lceil \log(n) \rceil - j - 1} 2^{j+m+1} \cdot \frac{f}{n} \cdot \left(\frac{1}{e}\right)^{2^{m-1}} \\
\leq \ & 2 \cdot \frac{f}{n} + 2^{j+1} \cdot \frac{f}{n} + 2^{j+2} \cdot \frac{f}{n} \cdot \sum_{m=0}^{\lceil \log(n) \rceil - j - 1} 2^{m-1} \cdot \left(\frac{1}{e}\right)^{2^{m-1}} \\
\leq \ & 2 \cdot \frac{f}{n} + 2^{j+1} \cdot \frac{f}{n} + 2^{j+2} \cdot \frac{f}{n} \cdot \sum_{m=0}^{\lceil \log(n) \rceil - j - 1} 2^{-m+1} \\
\in \ & \mathcal{O}(\tilde{r}_i) \ ,
\end{aligned}$$

where, as in the proof of Lemma 3.2.1, the last inequality follows from the easily provable fact that

$$2^{m-1} \cdot \left(\frac{1}{e}\right)^{2^{m-1}} \leq 2^{-m+1}$$

for all $m \geq 0$. Finally, due to the definition of $\tilde{r}_i$, the expected final connection cost of the point x_i is $\mathcal{O}(r_i)$. □

Now, we can prove that our distributed algorithm for the uniform metric facility location problem produces a solution whose total cost is with high constant probability at most a constant factor larger than the optimal cost.

Lemma 3.2.4. *The facility location cost for X is $\mathcal{O}(\mathrm{FacLoc}^*(X, f))$ with high constant probability.*

Proof. Due to Lemmas 3.2.1 and 3.2.3, the expected opening cost as well as the expected final connection cost of any point $x_i \in X$ is $\mathcal{O}(r_i)$. Thus, the algorithm computes a set of open facilities F that leads to an expected total cost of $\sum_{x_i \in X} \mathcal{O}(r_i)$. By applying Lemma 3.1.10 with $\ell = 1$, we have that the expected value of $\mathrm{FacLoc}(X, F, f)$ is $\mathcal{O}(\mathrm{FacLoc}^*(X, f))$. Now, the assertion of the lemma follows by applying Markov's inequality. □

We summarize our results in the following theorem:

Theorem 2. *Given any n-point metric space (X, D), there is a randomized distributed algorithm working in the synchronous message passing model that computes with high constant probability a constant-factor approximation of the uniform metric facility location problem for X. The algorithm uses three rounds of all-to-all communication where the message sizes are bounded to $\mathcal{O}(\log(n))$ bits.*

3.3 Distributed Algorithm for Powers of Metric Spaces

In this section, we extend the distributed algorithm given in Section 3.2 to the uniform facility location problem for powers of metric spaces. Let $\ell \in \mathbb{R}$ with $\ell \geq 1$ be the (constant) metric exponent. Then, we only have to make the following three adaptations to Algorithm 3.2.1:

1. The total number of phases is $\lceil \log(n)/\ell \rceil + 1$.
2. The k-th phase bit is set to 1 with probability $\min\{2^{k\ell}/n, 1\}$ and 0 otherwise.
3. In the k-th phase, we check the first $k-1$ phase bits of all the points in a distance of at most $2^k \cdot (f/n)^{1/\ell}$ from x_i.

The rest of the local algorithm for x_i remains unchanged. A complete pseudocode listing of the adapted algorithm is given by Algorithm 3.3.1.

Algorithm 3.3.1 Local Algorithm for Point x_i

1: open$[i] \leftarrow$ FALSE
2: **for** $k \leftarrow 0$ **to** $\lceil \log(n)/\ell \rceil$ **do**
3: $\quad \varphi_i[k] \leftarrow \begin{cases} 1 & \text{, with probability } \min\{2^{k\ell}/n, 1\} \\ 0 & \text{, otherwise} \end{cases}$
4: **send** φ_i to all $x_j \in X,\ j \neq i$
5: **receive** φ_j from all $x_j \in X,\ j \neq i$
6: **for** $k \leftarrow 0$ **to** $\lceil \log(n)/\ell \rceil$ **do**
7: $\quad$ **if** $\varphi_i[k] = 1$ **and** for each point $x_j \in \mathcal{B}(x_i, 2^k \cdot (f/n)^{1/\ell})$, $x_j \neq x_i$, we have $\varphi_j[m] = 0$ for all $m < k$ **then**
8: $\quad\quad$ open$[i] \leftarrow$ TRUE
9: **send** open$[i]$ to all $x_j \in X,\ j \neq i$
10: **receive** open$[j]$ from all $j \in X,\ j \neq i$
11: **if** open$[i] =$ FALSE **then**
12: $\quad$ **connect** to the nearest open facility

3.3.1 Analysis of the Algorithm

Let $\tilde{r}_i := 2^j \cdot (f/n)^{1/\ell}$ where j is the smallest integer that satisfies the inequality

$$\sum_{x \in X \cap \mathcal{B}(x_i, \tilde{r}_i)} (\tilde{r}_i^\ell - \mathrm{D}(x_i, x)^\ell) \geq f$$

be an approximation of the radius r_i defined by Equation (3.1). Then, using this approximation $\tilde{r}_i$ and bearing the three adaptations mentioned above in mind, the analysis of our distributed algorithm given in Section 3.2.1 can be easily transferred to the uniform facility location problem for powers of metric spaces. We obtain the following lemmas:

Lemma 3.3.1. *Let $x_i \in X$ be any point. Then, the expected opening cost of x_i is $\mathcal{O}(4^\ell \cdot r_i^\ell)$.*

Proof. We prove this lemma by using Lemma 3.1.10 and reusing the techniques given in the proof of Lemma 3.2.1.

First, we compute an upper bound on the probability that the algorithm opens a facility at x_i in any phase $k \in \{0, 1, \ldots, \lceil \log(n)/\ell \rceil\}$. Recall that this happens if the k-th phase bit of x_i is 1 and the first $k-1$ bits of all the other points at a distance of at most $2^k \cdot (f/n)^{1/\ell}$ are 0. Let $Y_{i,k}$ be the indicator random variable for the event that the algorithm opens a facility at x_i in phase k. We examine the two cases $k < j$ and $j \leq k \leq \lceil \log(n)/\ell \rceil$ with $j = \log(\tilde{r}_i \cdot (n/f)^{1/\ell})$.

Case $j \leq k \leq \lceil \log(n)/\ell \rceil$: The k-th phase bit of x_i is 1 with probability $\min\{2^{k\ell}/n, 1\} \leq 2^{k\ell}/n$. Furthermore, for any phase $m < k$, the m-th phase bit of an arbitrary point located in $\mathcal{B}(x_i, 2^k \cdot (f/n)^{1/\ell})$ is 0 with probability $1 - 2^{m\ell}/n$. Thus, the probability that all of the first $k-1$ phase bits of this point are 0 is $\prod_{m=0}^{k-1} 1 - 2^{m\ell}/n$. Hence, we get

$$\mathbf{Pr}\left[Y_{i,k} = 1\right] \leq \frac{2^{k\ell}}{n} \cdot \left[\left(1 - \frac{2^{0\ell}}{n}\right) \cdot \left(1 - \frac{2^{1\ell}}{n}\right) \cdot \ldots \cdot \left(1 - \frac{2^{(k-1)\ell}}{n}\right)\right]^{\mathrm{weight}\left(\mathcal{B}\left(x_i, 2^k \cdot (f/n)^{1/\ell}\right)\right)} .$$

Due to Lemma 3.1.1 and $\tilde{r}_i \geq r_i$, we have

$$\begin{aligned}
\mathrm{weight}\left(\mathcal{B}\left(x_i, 2^k \cdot \left(\frac{f}{n}\right)^{1/\ell}\right)\right) &\geq \mathrm{weight}\left(\mathcal{B}\left(x_i, 2^j \cdot \left(\frac{f}{n}\right)^{1/\ell}\right)\right) \\
&\geq \mathrm{weight}\left(\mathcal{B}(x_i, \tilde{r}_i)\right) \\
&\geq \mathrm{weight}\left(\mathcal{B}(x_i, r_i)\right) \\
&\geq \frac{f}{r_i^\ell} \\
&\geq \frac{f}{\tilde{r}_i^\ell} .
\end{aligned}$$

It follows that

$$\begin{aligned}
\mathbf{Pr}\left[Y_{i,k}=1\right] &\leq \frac{2^{k\ell}}{n}\cdot\left[\left(1-\frac{2^{0\ell}}{n}\right)\cdot\left(1-\frac{2^{1\ell}}{n}\right)\cdot\ldots\cdot\left(1-\frac{2^{(k-1)\ell}}{n}\right)\right]^{\frac{f}{\tilde{r}_i^{\ell}}} \\
&= \frac{2^{k\ell}}{n}\cdot\left[\left(1-\frac{2^{0\ell}}{n}\right)\cdot\left(1-\frac{2^{1\ell}}{n}\right)\cdot\ldots\cdot\left(1-\frac{2^{(k-1)\ell}}{n}\right)\right]^{\frac{n}{2^{j\ell}}} \\
&\leq \frac{2^{k\ell}}{n}\cdot\left(1-\frac{2^{(k-1)\ell}}{n}\right)^{\frac{n}{2^{j\ell}}} \quad .
\end{aligned}$$

Now, let m be the non-negative integer $k-j$. Then, we obtain

$$\begin{aligned}
\mathbf{Pr}\left[Y_{i,k}=1\right] &\leq \frac{2^{(j+m)\ell}}{n}\cdot\left(1-\frac{2^{(j+m-1)\ell}}{n}\right)^{\frac{n}{2^{j\ell}}} \\
&= \frac{2^{(j+m)\ell}}{n}\cdot\left(1-\frac{2^{(j+m-1)\ell}}{n}\right)^{\frac{n}{2^{(j+m-1)\ell}}\cdot 2^{(m-1)\ell}} \\
&\leq \frac{2^{(j+m)\ell}}{n}\cdot\left(\frac{1}{e}\right)^{2^{(m-1)\ell}} \quad ,
\end{aligned}$$

where the last inequality is due to a bound on Euler's number (see Inequality (B.2)).

Case $k < j$: Obviously, an upper bound on the probability that the algorithm opens a facility at x_i in a phase $k < j$ is $2^{k\ell}/n$. It follows that

$$\mathbf{Pr}\left[Y_{i,k}=1\right] \leq \frac{2^{k\ell}}{n} \quad .$$

Let Y_i be the indicator random variable for the event that the algorithm opens a facility at x_i. Then, the expected opening cost of the point x_i are upper bounded by

$$\begin{aligned}
f\cdot\mathbf{E}\left[Y_i\right] &= f\cdot\mathbf{E}\left[\sum_{k=0}^{\lceil\log(n)/\ell\rceil} Y_{i,k}\right] \\
&= f\cdot\sum_{k=0}^{\lceil\log(n)/\ell\rceil}\mathbf{E}\left[Y_{i,k}\right] \\
&= f\cdot\sum_{k=0}^{\lceil\log(n)/\ell\rceil}\mathbf{Pr}\left[Y_{i,k}=1\right]
\end{aligned}$$

Based on the above two cases, we obtain

$$\begin{aligned} f \cdot \mathbf{E}[Y_i] &\leq f \cdot \sum_{k=0}^{j-1} \frac{2^{k\ell}}{n} + f \cdot \sum_{m=0}^{\lceil \log(n)/\ell \rceil - j} \frac{2^{(j+m)\ell}}{n} \cdot \left(\frac{1}{e}\right)^{2^{(m-1)\ell}} \\ &= \frac{f}{n} \cdot \frac{2^{j\ell}-1}{2^\ell - 1} + \frac{f}{n} \cdot 2^{(j+1)\ell} \cdot \sum_{m=0}^{\lceil \log(n)/\ell \rceil - j} \left(\frac{1}{e}\right)^{2^{(m-1)\ell}} \cdot 2^{(m-1)\ell} \\ &\leq \frac{f}{n} \cdot 2^{j\ell} + \frac{f}{n} \cdot 2^{(j+1)\ell} \cdot \sum_{m=0}^{\lceil \log(n)/\ell \rceil - j} 2^{-m+1} \\ &\in \mathcal{O}(2^\ell \cdot \tilde{r}_i^\ell) \ , \end{aligned}$$

where the last inequality follows from the easily provable fact that

$$\left(\frac{1}{e}\right)^{2^{(m-1)\ell}} \cdot 2^{(m-1)\ell} \leq 2^{-m+1}$$

for all $m \geq 0$ and any $\ell \geq 1$. Finally, due to the definition of $\tilde{r}_i$, the expected opening cost of the point x_i is $\mathcal{O}(4^\ell \cdot r_i^\ell)$. □

The proof of our upper bound on the final connection cost of any point $x_i \in X$ utilizes the following lemma:

Lemma 3.3.2. *Let $x_i \in X$ be any point that has been chosen as open facility or that has tentatively been connected in any phase $k \in \{0, \ldots, \lceil \log(n)/\ell \rceil\}$. Then, the distance of x_i to the nearest open facility is at most $2^{k+1} \cdot (f/n)^{1/\ell}$.*

Proof. To prove this lemma, we use the same approach as in the proof of Lemma 3.2.2.

In case that the algorithm opens a facility at x_i in phase k, the distance of x_i to the nearest open facility is $0 \leq 2^{k+1} \cdot (f/n)^{1/\ell}$. Next, we consider the case that x_i has been connected tentatively. The algorithm does not tentatively connect any point in phase 0. Hence, in the following, we will assume that x_i has tentatively been connected to a point $x_j \in X$ in a phase $k \in \{1, \ldots, \lceil \log(n)/\ell \rceil\}$. Then, the distance from x_i to x_j is at most $2^k \cdot (f/n)^{1/\ell}$. Let m denote the smallest number of a phase bit of x_j whose value is 1. Since x_i has tentatively been connected to x_j in phase k, we have $m \leq k-1$. Now, we have to consider the two cases that either x_j is open or x_j has been connected tentatively as well. Obviously, if x_j is an open facility, then the distance from x_i to the nearest open facility is at most $2^k \cdot (f/n)^{1/\ell}$, so we are done. Otherwise, x_j has tentatively been connected to another point within a distance of at most $2^m \cdot (f/n)^{1/\ell} \leq 2^{k-1} \cdot (f/n)^{1/\ell}$. By recursively applying this argument, we obtain that there must be an open facility within a distance of at most

$$2^k \cdot \left(\frac{f}{n}\right)^{1/\ell} + \sum_{m=0}^{k-1} 2^m \cdot \left(\frac{f}{n}\right)^{1/\ell} \leq 2^{k+1} \cdot \left(\frac{f}{n}\right)^{1/\ell}$$

from x_i. □

Lemma 3.3.3. *Let $x_i \in X$ be any point. Then, the expected final connection cost of x_i is $\mathcal{O}(16^\ell \cdot r_i^\ell)$.*

Proof. We prove this lemma by using Lemma 3.3.2 and reusing the techniques given in the proof of Lemma 3.2.3.

The algorithm does not tentatively connect any point in phase 0. Hence, in phase 0, it either opens a facility at x_i or does nothing with x_i, which obviously results in 0 connection cost for x_i in phase 0. Due to Lemma 3.3.2, if the point x_i has been chosen as an open facility or has tentatively been connected in any other phase $k \in \{1, \ldots, \lceil \log(n)/\ell \rceil\}$, then its final connection cost is at most $2^{(k+1)\ell} \cdot f/n$. Now, for each $k \in \{0, \ldots, \lceil \log(n)/\ell \rceil\}$, let $Z_{i,k}$ be the indicator random variable for the event that the algorithm has not opened a facility at x_i and has not tentatively connected x_i up to and including phase k. Then, we can upper bound the expected final connection cost of x_i by

$$\begin{aligned}
& \sum_{k=1}^{\lceil \log(n)/\ell \rceil} 2^{(k+1)\ell} \cdot \frac{f}{n} \cdot \mathbf{Pr}\left[x_i \text{ is opened or tentatively connected in phase } k\right] \\
= & \sum_{k=1}^{\lceil \log(n)/\ell \rceil} 2^{(k+1)\ell} \cdot \frac{f}{n} \cdot \left(\mathbf{Pr}\left[Z_{i,k-1} = 1\right] - \mathbf{Pr}\left[Z_{i,k} = 1\right]\right) \\
\leq & \sum_{k=0}^{\lceil \log(n)/\ell \rceil - 1} 2^{(k+2)\ell} \cdot \frac{f}{n} \cdot \mathbf{Pr}\left[Z_{i,k} = 1\right]
\end{aligned}$$

In order to upper bound the expected final connection cost of x_i, we upper bound the probabilities $\mathbf{Pr}\left[Z_{i,k} = 1\right]$. Therefore, we examine the two cases $k < j$ and $j \leq k < \lceil \log(n)/\ell \rceil$ with $j = \log(\tilde{r}_i \cdot (n/f)^{1/\ell})$.

Case $j \leq k < \lceil \log(n)/\ell \rceil$: Observe that we have $Z_{i,k} = 1$ only in the case that the first k phase bits of x_i are 0 and the first $k-1$ phase bits of all the other points at a distance of at most $2^k \cdot (f/n)^{1/\ell}$ are 0 as well. For any phase $m \leq k$, the m-th phase bit of x_i is 0 with probability $1 - 2^{m\ell}/n$. Thus, the probability that all of the first k phase bits of x_i are 0 is $\prod_{m=0}^{k} 1 - 2^{m\ell}/n$. Similarly, the probability that all of the first $k-1$ phase bits of any point in $\mathcal{B}(x_i, 2^k \cdot (f/n)^{1/\ell})$ are 0 is $\prod_{m=0}^{k-1} 1 - 2^{m\ell}/n$. As proven in Lemma 3.3.1, the number of points in $\mathcal{B}(x_i, 2^k \cdot (f/n)^{1/\ell})$ is lower bounded by

$$\text{weight}\left(\mathcal{B}\left(x_i, 2^k \cdot \left(\frac{f}{n}\right)^{1/\ell}\right)\right) \geq \frac{f}{\tilde{r}_i^\ell} \ .$$

Hence, we have

$$\begin{aligned}
\mathbf{Pr}\left[Z_{i,k} = 1\right] &\leq \left[\left(1 - \frac{2^{0\ell}}{n}\right) \cdot \ldots \cdot \left(1 - \frac{2^{k\ell}}{n}\right)\right] \cdot \left[\left(1 - \frac{2^{0\ell}}{n}\right) \cdot \ldots \cdot \left(1 - \frac{2^{(k-1)\ell}}{n}\right)\right]^{\frac{f}{\tilde{r}_i^\ell}} \\
&\leq \left(1 - \frac{2^{k\ell}}{n}\right) \cdot \left(1 - \frac{2^{(k-1)\ell}}{n}\right)^{\frac{n}{2^{j\ell}}} \ .
\end{aligned}$$

Let m denote the non-negative integer $k - j$. Then, we get

$$\begin{aligned}\mathbf{Pr}\left[Z_{i,k} = 1\right] &\leq \left(1 - \frac{2^{(j+m)\ell}}{n}\right) \cdot \left(1 - \frac{2^{(j+m-1)\ell}}{n}\right)^{\frac{n}{2^{(j+m-1)\ell}} \cdot 2^{(m-1)\ell}} \\ &\leq \left(1 - \frac{2^{(j+m)\ell}}{n}\right) \cdot \left(\frac{1}{e}\right)^{2^{(m-1)\ell}} \leq \left(\frac{1}{e}\right)^{2^{(m-1)\ell}} ,\end{aligned}$$

where the second inequality is due to a bound on Euler's number (see Inequality (B.2)).

Case $k < j$: An obvious upper bound on the probability that the algorithm does not open a facility at x_i or tentatively connect x_i up to and including phase k is 1, so we have

$$\mathbf{Pr}\left[Z_{i,k}\right] \leq 1 .$$

Now, we can upper bound the expected final connection cost x_i by

$$\begin{aligned}&\sum_{k=0}^{\lceil \log(n)/\ell \rceil - 1} 2^{(k+2)\ell} \cdot \frac{f}{n} \cdot \mathbf{Pr}\left[Z_{i,k} = 1\right] \\ \leq\ & \sum_{k=0}^{j-1} 2^{(k+2)\ell} \cdot \frac{f}{n} \cdot 1 + \sum_{m=0}^{\lceil \log(n)/\ell \rceil - j - 1} 2^{(j+m+2)\ell} \cdot \frac{f}{n} \cdot \left(\frac{1}{e}\right)^{2^{(m-1)\ell}} \\ =\ & \frac{2^{j\ell} - 1}{2^\ell - 1} \cdot 2^{2\ell} \cdot \frac{f}{n} + 2^{(j+3)\ell} \cdot \frac{f}{n} \cdot \sum_{m=0}^{\lceil \log(n)/\ell \rceil - j - 1} 2^{(m-1)\ell} \cdot \left(\frac{1}{e}\right)^{2^{(m-1)\ell}} \\ \leq\ & 2^{(j+2)\ell} \cdot \frac{f}{n} + 2^{(j+3)\ell} \cdot \frac{f}{n} \cdot \sum_{m=0}^{\lceil \log(n)/\ell \rceil - j - 1} 2^{-m+1} \\ \in\ & \mathcal{O}(8^\ell \cdot \tilde{r}_i^\ell) ,\end{aligned}$$

where, as in the proof of Lemma 3.3.1, the last inequality follows from the easily provable fact that

$$2^{(m-1)\ell} \cdot \left(\frac{1}{e}\right)^{2^{(m-1)\ell}} \leq 2^{-m+1}$$

for all $m \geq 0$ and any $\ell \geq 1$. Finally, due to the definition of $\tilde{r}_i$, the expected final connection cost of x_i is $\mathcal{O}(16^\ell \cdot r_i^\ell)$. □

Lemma 3.3.4. *The facility location cost for X is $\mathcal{O}(\mathrm{FacLoc}^*(X, f, \ell))$ with high constant probability.*

Proof. It follows from Lemmas 3.3.1 and 3.3.3 and ℓ being a constant metric exponent that the expected opening cost as well as the expected final connection cost of any point $x_i \in X$ is $\mathcal{O}(r_i^\ell)$. Hence, the algorithm computes a set of open facilities F that leads to an expected total cost of $\sum_{x_i \in X} \mathcal{O}(r_i^\ell)$. Due to Lemma 3.1.10, we obtain that the expected value of $\mathrm{FacLoc}(X, F, f, \ell)$ is $\mathcal{O}(\mathrm{FacLoc}^*(X, f, \ell))$. Finally, the assertion of the lemma follows by applying Markov's inequality. □

We summarize our results in the following theorem:

Theorem 3. *Given any n-point metric space (X, D) and a constant metric exponent $\ell \geq 1$, there is a randomized distributed algorithm working in the synchronous message passing model that computes for X with high constant probability a constant-factor approximation of the uniform facility location problem for powers of metric spaces. The algorithm uses three rounds of all-to-all communication where the message sizes are bounded to $\mathcal{O}(\log(n))$ bits.*

4 A Kinetic Data Structure for Facility Location

In this chapter, we investigate a facility location problem under motion. The input is a set of continuously moving objects. Each object moves along a known trajectory and can change its status between open facility and client at any time. The goal is to maintain a subset of the given objects as open facilities such that, at any time, the current facility location cost induced by the chosen open facilities is as close to the current optimal cost as possible, and also some side condition is satisfied. Observe that minimizing the mobile facility location cost at any time, without considering any side condition, can result in many status changes of the objects. Depending on the tasks of an open facility, such a status change can be expensive. Hence, the side condition we consider is to change the status of an object rather seldom so that the total number of status changes is below some appropriate threshold.

Since the kinetic data structure (KDS) framework is well-suited to maintain a combinatorial structure of continuously moving objects and common in the field of computational geometry [2, 15, 54], we developed a KDS for the facility location problem described above. Our KDS applies a counting argument of Bădoiu et al. [14] to kinetize a modified version of the Mettu-Plaxton algorithm. The counting argument asserts that the radius of a facility can be approximated well by just counting the number of points in exponentially growing balls centered at this particular facility.

Note that we cannot apply the original Mettu-Plaxton algorithm to obtain a responsive KDS, i.e., a KDS with polylogarithmic update time. The reason is that similar to maintaining an exact solution for the mobile facility location problem, maintaining the solution provided by Algorithm 2.3.1 is not stable. That means, a slight perturbation of the input might result in a number of status changes that is linear in the number of input points, whereas we are looking for stable solutions, where only a polylogarithmic number of changes occur upon an event.

In Section 4.1, we present the essential ideas and some notations used throughout this chapter. A detailed description of the KDS can be found in Section 4.2. We analyze our KDS in Section 4.3. First, we prove that, at any time, it is guaranteed that our current set of open facilities leads to a total cost which is at most a constant factor larger than the current optimal cost. Afterwards, we analyze our KDS in terms of its complexity.

4.1 The Special Radii

The input of the considered mobile facility location problem is a set $P = \{p_1, p_2, \ldots, p_n\}$ of n independently moving points in $\mathbb{R}^d$, where d is a constant. For any point $p_i \in P$, we denote its opening cost by f_i and its demand by d_i. Furthermore, let $p_i(t)$ denote

the position of p_i at the point of time t, and let $P(t) := \{p_1(t), p_2(t), \ldots, p_n(t)\}$. Then, the mobile facility location problem is to maintain, at each point of time t, a set of open facilities $F(t)$ such that $\mathrm{FacLoc}(P(t), F(t))$ is minimized (see Section 2.2 for a definition of $\mathrm{FacLoc}(P(t), F(t))$). We let $F^*(t)$ denote an optimal set of open facilities at the point of time t.

To approach the mobile facility location problem, we kinetize a modified version of the Mettu-Plaxton algorithm. One essential modification affects the radius associated with a point. According to Equation (2.1), we let $r_i(t)$ be the radius of a point $p_i \in P$ at the point of time t. More precisely, $r_i(t)$ is the radius of the ball with center $p_i(t)$ that satisfies

$$\sum_{p_j(t) \in P(t) \cap \mathcal{B}(p_i(t), r_i(t))} d_j \cdot (r_i(t) - \mathrm{D}(p_i(t), p_j(t))) = f_i \ . \tag{4.1}$$

Let $r_{\min}$ denote the lower limit of the range of $r_i(t)$, and let $r_{\max}$ denote the upper limit. Then, as observed in Section 2.3, we have

$$r_{\min} = \frac{\min_{p_j \in P} f_j}{n \cdot \max_{p_j \in P} d_j} \quad \text{and} \quad r_{\max} = \frac{\max_{p_j \in P} f_j}{\min_{p_j \in P} d_j} \ . \tag{4.2}$$

Based on this definition of a radius, we introduce a new radius associated with a point. This new radius is much easier to maintain than the original radius when the points move. Compared to the original radii, the new radii of the points depend on cubes instead of balls. The key idea of our KDS is to use a set of nested cubes around each point and to update the KDS each time a point enters or leaves a cube of another point.

4.1.1 Definition of the Special Radii

Cubes. Similar to the definition of balls, for a point $p_i(t) \in P(t)$ and a non-negative value r, we define $\mathcal{C}(p_i(t), r)$ to be the axis-parallel cube whose center is the point $p_i(t)$ and whose side length is $2r$. Given such a cube $\mathcal{C}(p_i(t), r)$, we let $\mathrm{weight}(\mathcal{C}(p_i(t), r))$ denote the sum of the demands of all the points in $P(t)$ that are located in the cube $\mathcal{C}(p_i(t), r)$, i.e., we define

$$\mathrm{weight}(\mathcal{C}(p_i(t), r)) := \sum_{p_j(t) \in P(t) \cap \mathcal{C}(p_i(t), r)} d_j \, .$$

Note that the cube $\mathcal{C}(p_i(t), r)$ is a ball with radius r with respect to the L_∞-metric. According to this and for sake of simplicity, we will refer to the value r of a cube $\mathcal{C}(p_i(t), r)$ as the radius of the cube, i.e., the double radius of a cube is equal to its side length.

Special Radius Associated with a Point. Our KDS maintains for each point $p_i \in P$ an approximation of $r_i(t)$, called the *special radius* $\tilde{r}_i(t)$, which is defined as follows:

Definition 4.1.1 (Special Radius)**.** *At any point of time t, the* special radius $\tilde{r}_i(t)$ *of any point $p_i \in P$ is the value $2^{\tilde{k}}$ such that $\tilde{k} = k_0 + \lceil \log(4\sqrt{d}) \rceil$ and k_0 is the minimum integer k with $\lceil \log(r_{\min}) \rceil \leq k \leq \lceil \log(r_{\max}) \rceil$ for which $\mathrm{weight}(\mathcal{C}(p_i(t), 2^{k_0})) \geq f_i \cdot 2^{-k_0}$ holds.*

In the following, we will prove the existence of the special radius $\tilde{r}_i(t)$ of any point $p_i(t) \in P(t)$ at any point of time t. Moreover, we will show that the special radius $\tilde{r}_i(t)$ is a constant-factor approximation of the value $r_i(t)$.

The proof of the existence of the special radius is based on a result obtained in [14]. More precisely, for the uniform metric facility location problem, the authors in [14] gave lower and upper bounds on the value $r_i(t)$ (confer also Lemma 3.1.1 with $\ell = 1$). We generalize their result to the non-uniform case:

Lemma 4.1.2. *At any point of time t and for each $p_i \in P$, we have*

$$\frac{f_i}{\text{weight}(\mathcal{B}(p_i(t), r_i(t)))} \leq r_i(t) \leq \frac{2 \cdot f_i}{\text{weight}(\mathcal{B}(p_i(t), r_i(t)/2))} \ .$$

Proof. It follows from the definition of $r_i(t)$ given in Equation (4.1) that

$$\sum_{p_j(t) \in P(t) \cap \mathcal{B}(p_i(t), r_i(t))} d_j \cdot r_i(t) \geq f_i \ ,$$

so we have

$$r_i(t) \geq \frac{f_i}{\sum_{p_j(t) \in P(t) \cap \mathcal{B}(p_i(t), r_i(t))} d_j} = \frac{f_i}{\text{weight}(\mathcal{B}(p_i(t), r_i(t)))} \ .$$

This proves the first inequality of the lemma.

Furthermore, we get

$$\begin{aligned} f_i &= \sum_{p_j(t) \in P(t) \cap \mathcal{B}(p_i(t), r_i(t))} d_j \cdot (r_i(t) - \mathrm{D}(p_i(t), p_j(t))) \\ &\geq \sum_{p_j(t) \in P(t) \cap \mathcal{B}(p_i(t), r_i(t)/2)} d_j \cdot (r_i(t) - \mathrm{D}(p_i(t), p_j(t))) \\ &\geq \frac{r_i(t)}{2} \cdot \sum_{p_j(t) \in P(t) \cap \mathcal{B}(p_i(t), r_i(t)/2)} d_j \\ &= \frac{r_i(t)}{2} \cdot \text{weight}(\mathcal{B}(p_i(t), r_i(t)/2)) \, , \end{aligned}$$

where the second inequality follows from the fact that $r_i(t) - \mathrm{D}(p_i(t), p_j(t)) \geq r_i(t)/2$ for all $p_j(t) \in P(t) \cap \mathcal{B}(p_i(t), r_i(t)/2)$ and $\mathcal{B}(p_i(t), r_i(t)/2) \subseteq \mathcal{B}(p_i(t), r_i(t))$. This proves the second equality of the lemma. □

Lemma 4.1.3. *Let t be any point of time, and let $p_i \in P$ be any point. Then, there exists an integer k with $\lceil \log(r_{\min}) \rceil \leq k \leq \lceil \log(r_{\max}) \rceil$ such that*

$$\text{weight}(\mathcal{B}(p_i(t), 2^k)) \geq f_i \cdot 2^{-k} \ .$$

Proof. Due to Lemma 4.1.2, we have

$$\text{weight}(\mathcal{B}(p_i(t), 2^{\log(r_i(t))})) = \text{weight}(\mathcal{B}(p_i(t), r_i(t))) \geq \frac{f_i}{r_i(t)} = \frac{f_i}{2^{\log(r_i(t))}} \ .$$

Since $2^{\lceil \log(r_i(t)) \rceil} \geq 2^{\log(r_i(t))}$, it follows that

$$\text{weight}(\mathcal{B}(p_i(t), 2^{\lceil \log(r_i(t)) \rceil})) \geq \frac{f_i}{2^{\lceil \log(r_i(t)) \rceil}} \ .$$

Now, the existence of an integer k with $\lceil \log(r_{\min}) \rceil \leq k \leq \lceil \log(r_{\max}) \rceil$ such that

$$\text{weight}(\mathcal{B}(p_i(t), 2^k)) \geq f_i \cdot 2^{-k}$$

follows from $r_{\min} \leq r_i(t) \leq r_{\max}$. □

Due to Lemma 4.1.3 and the fact that a ball with a certain radius is completely covered by the cube having the same center and the same radius as the ball, we obtain the following result:

Corollary 4.1.4. *Let t be any point of time, and let $p_i \in P$ be any point. Then, there exists an integer k with $\lceil \log(r_{\min}) \rceil \leq k \leq \lceil \log(r_{\max}) \rceil$ such that*

$$\text{weight}(\mathcal{C}(p_i(t), 2^k)) \geq f_i \cdot 2^{-k} \ .$$

It follows from Corollary 4.1.4 that, at each point of time t, the special radius $\tilde{r}_i(t)$ of each point $p_i \in P$ exists. Next, we use a modified version of a counting argument given in [14] to prove that $\tilde{r}_i(t)$ is a constant-factor approximation of $r_i(t)$. More precisely, for the uniform metric facility location problem, Bădoiu et al. [14] showed how to approximate $r_i(t)$ by counting the number of points in exponentially growing balls around $p_i(t)$. We generalize their result to the non-uniform case:

Lemma 4.1.5. *Let t be any point of time, let $p_i \in P$ be any point, and let k_1 be the minimum integer k with $\lceil \log(r_{\min}) \rceil \leq k \leq \lceil \log(r_{\max}) \rceil$ such that* $\text{weight}(\mathcal{B}(p_i(t), 2^k)) \geq f_i \cdot 2^{-k}$. *Then, it holds that*

$$\frac{1}{2} \cdot r_i(t) \leq 2^{k_1} \leq 2 \cdot r_i(t) \ .$$

Proof. The existence of the integer k_1 is due to Lemma 4.1.3. Furthermore, due to the choice of k_1, we have

$$\text{weight}\left(\mathcal{B}\left(p_i(t), 2^{k_1-1}\right)\right) < f_i \cdot 2^{-(k_1-1)} \ .$$

It follows that, for any $r_i(t) < 2^{k_1-1}$, we get

$$\begin{aligned} \text{weight}\left(\mathcal{B}\left(p_i(t), r_i(t)\right)\right) &\leq \text{weight}\left(\mathcal{B}\left(p_i(t), 2^{k_1-1}\right)\right) \\ &< f_i \cdot 2^{-(k_1-1)} \\ &< f_i \cdot \frac{1}{r_i(t)} \ . \end{aligned}$$

Now, we obtain

$$r_i(t) < \frac{f_i}{\text{weight}(\mathcal{B}(p_i(t), r_i(t)))} \ ,$$

which is a contradiction to Lemma 4.1.2. Hence, $r_i(t) \geq 2^{k_1-1}$ must be true, which proves the second inequality of the assertion.

Furthermore, for any $r_i(t) > 2^{k_1+1}$, we have

$$\begin{aligned} \text{weight}(\mathcal{B}(p_i(t), r_i(t)/2)) &\geq \text{weight}\left(\mathcal{B}\left(p_i(t), 2^{k_1}\right)\right) \\ &\geq f_i \cdot 2^{-k_1} \\ &> f_i \cdot \frac{2}{r_i(t)} \ . \end{aligned}$$

In this case, it follows that

$$r_i(t) > \frac{2f_i}{\text{weight}(\mathcal{B}(p_i(t), r_i(t)/2))} \ ,$$

which is again a contradiction to Lemma 4.1.2. Thus, we have $r_i(t) \leq 2^{k_1+1}$, which proves the first inequality of the assertion. □

Our algorithm uses the approach of [14], but, for any integer k, we approximate the sum of the demands of all the points in a ball with radius 2^k by the sum of the demands of all the points in a cube with radius 2^k. This leads to the following result:

Lemma 4.1.6. *Let t be any point of time, let $p_i \in P$ be any point, and let k_0 be the minimum integer k with $\lceil \log(r_{\min}) \rceil \leq k \leq \lceil \log(r_{\max}) \rceil$ such that* $\text{weight}(\mathcal{C}(p_i(t), 2^k)) \geq f_i \cdot 2^{-k}$. *Then, it holds that*

$$\frac{1}{4\sqrt{d}} \cdot r_i(t) \leq 2^{k_0} \leq 2 \cdot r_i(t) \ .$$

Proof. Let k_1, $\lceil \log(r_{\min}) \rceil \leq k_1 \leq \lceil \log(r_{\max}) \rceil$, be defined as in Lemma 4.1.5. Then, the radius of $\mathcal{C}(p_i(t), 2^{k_0})$ is at most 2^{k_1} since each point in $P(t)$ that is located in $\mathcal{B}(p_i(t), 2^{k_1})$ is also located in $\mathcal{C}(p_i(t), 2^{k_1})$. Thus, we get

$$\text{weight}\left(\mathcal{C}\left(p_i(t), 2^{k_1}\right)\right) \geq f_i \cdot 2^{-k_1} \ .$$

The maximum radius of $\mathcal{C}(p_i(t), 2^{k_0})$ is illustrated on the left hand side of Figure 4.1.

Furthermore, the radius of $\mathcal{C}(p_i(t), 2^{k_0})$ is larger than $1/\sqrt{d} \cdot 2^{k_1-1}$. The reason is that

$$\text{weight}\left(\mathcal{B}\left(p_i(t), 2^{k_1-1}\right)\right) < f_i \cdot 2^{-(k_1-1)}$$

and

$$\text{weight}\left(\mathcal{C}\left(p_i(t), \frac{1}{\sqrt{d}} \cdot 2^{k_1-1}\right)\right) \leq \text{weight}\left(\mathcal{B}\left(p_i(t), 2^{k_1-1}\right)\right) \ ,$$

so we have

$$\begin{aligned} \text{weight}\left(\mathcal{C}\left(p_i(t), 2^{k_1-1-\log(\sqrt{d})}\right)\right) &= \text{weight}\left(\mathcal{C}\left(p_i(t), \frac{1}{\sqrt{d}} \cdot 2^{k_1-1}\right)\right) \\ &< f_i \cdot 2^{-(k_1-1)} \\ &< f_i \cdot 2^{-\left(k_1-1-\log\left(\sqrt{d}\right)\right)} \ . \end{aligned}$$

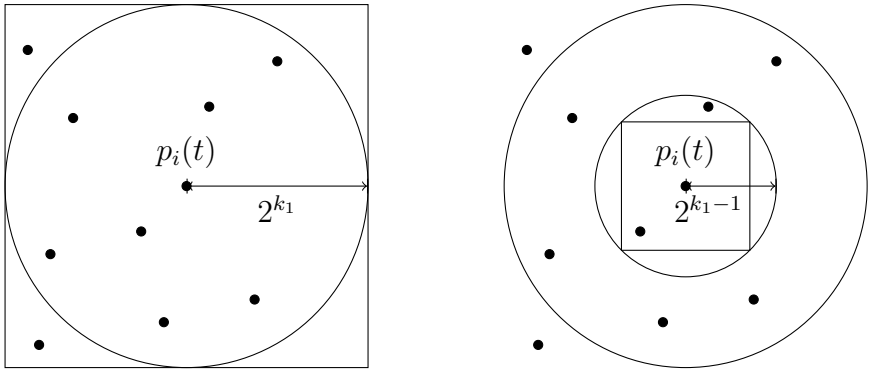


Figure 4.1: Illustration of the maximum and minimum radius of $\mathcal{C}(p_i(t), 2^{k_0})$.

The minimum radius of $\mathcal{C}(p_i(t), 2^{k_0})$ is illustrated on the right hand side of Figure 4.1.

Now, the lemma follows from $1/\sqrt{d} \cdot 2^{k_1-1} < 2^{k_0} \leq 2^{k_1}$ and Lemma 4.1.5. □

Based on Lemma 4.1.6, we can now show that the special radius associated with a point is always a constant-factor approximation of the original radius defined by Mettu and Plaxton [87]. Furthermore, we prove that the number of possible values of a special radius is only logarithmic in nR where

$$R := \frac{\max_{p_i \in P} f_i \cdot \max_{p_i \in P} d_i}{\min_{p_i \in P} f_i \cdot \min_{p_i \in P} d_i} \ .$$

Lemma 4.1.7. *Let t be any point of time, and let $p_i \in P$ be any point. Then, we have*

$$r_i(t) \leq \tilde{r}_i(t) \leq 2^{3+\lceil \log(\sqrt{d}) \rceil} \cdot r_i(t) \ .$$

The number of possible values for $\tilde{r}_i(t)$ is upper bounded by $\mathcal{O}(\log(nR))$.

Proof. Due to Lemma 4.1.6, we have

$$2^{-\log(4\sqrt{d})} \cdot r_i(t) \leq 2^{k_0} \leq 2 \cdot r_i(t) \ .$$

According to Definition 4.1.1, we set the special radius to

$$\tilde{r}_i(t) = 2^{\tilde{k}} = 2^{k_0 + \lceil \log(4\sqrt{d}) \rceil)} \ ,$$

so we obtain $r_i(t) \leq \tilde{r}_i(t) \leq 2^{3+\lceil \log(\sqrt{d}) \rceil} \cdot r_i(t)$. Due to $r_{\min} \leq r_i(t) \leq r_{\max}$, Equation (4.2), and the fact that $\tilde{r}_i(t)$ is a power of 2, there are $\mathcal{O}(\log(nR))$ possible values for $\tilde{r}_i(t)$. □

Walls around a Point. We consider a set of $\mathcal{O}(\log(nR))$ nested cubes for each point $p_i(t) \in P(t)$. More precisely, there is the cube $\mathcal{C}(p_i(t), 2^k)$ with radius 2^k for each $k \in \{\lceil \log(r_{\min}) \rceil + \lceil \log(4\sqrt{d}) \rceil, \lceil \log(r_{\min}) \rceil + 1 + \lceil \log(4\sqrt{d}) \rceil, \ldots, \lceil \log(r_{\max}) \rceil + \lceil \log(4\sqrt{d}) \rceil\}$. The side faces of the cube defined by $\mathcal{C}(p_i(t), 2^k)$ form a wall around $p_i(t)$, which we call $\mathcal{W}_{i,k}(t)$. Hence, there exists a set of $\mathcal{O}(\log(nR))$ walls for $p_i(t)$. We use this set of walls to determine the points of time when an update of p_i in our KDS is required. In general, an event occurs each time when any point crosses any wall of another point.

4.1.2 Computation of the Special Radii

In order to compute the special radius associated with any point efficiently at any time, we maintain two $(d+1)$-dimensional dynamic range trees denoted by T_1 and T_2. At any time, range tree T_1 is used to manage the current set of open facilities (which we call *open points*), and T_2 stores the current set of clients (which we call *closed points*). Apart from the fact that the two data structures contain different point sets, they are constructed in the same way. In the first d levels of the range trees, the points are handled according to their coordinates and, in the $(d+1)$-st level, the points are handled according to their special radii. Additionally, with each node v in every binary search tree of the $(d+1)$-st level, we store the sum of the demands of all the points contained in the subtree rooted at v.

At any point of time t, the range trees rely on the relative position of the points in $P(t)$. More precisely, the leaves of any binary search tree of any level ℓ, $1 \leq \ell \leq d$, in T_1 and T_2 store the points sorted according to their ranks based on dimension ℓ, i.e., sorted according to their ℓ-th coordinate. Now, the movement of the points in P is reflected by insert and delete operations on T_1 and T_2. At each point of time t, when any two points $p_i(t), p_j(t) \in P(t)$ change their ranks based on any dimension ℓ, we delete p_i and p_j from T_1 and T_2 and reinsert them according to their position at time t.

By applying a technique proposed by Willard and Lueker in [109], we are able to support all required properties of T_1 and T_2 efficiently. More precisely, T_1 and T_2 have the following complexity:

Lemma 4.1.8 ([109])**.** *The range trees T_1 and T_2 have a space requirement of $\mathcal{O}(n \log^d(n))$ and can be initialized in $\mathcal{O}(n \log^{d+1}(n))$ time. The worst-case time per insertion and deletion is $\mathcal{O}(\log^{d+1}(n))$. Given any orthogonal range $[x_1, x_1'] \times [x_2, x_2'] \times \ldots \times [x_{d+1}, x_{d+1}'] \subset \mathbb{R}^{d+1}$ at any time t, the set*

$$Q := \{p_i(t) \in P(t) \mid p_i(t) \in [x_1, x_1'] \times [x_2, x_2'] \times \ldots \times [x_d, x_d'] \text{ and } \tilde{r}_i(t) \in [x_{d+1}, x_{d+1}']\}$$

can be computed in $\mathcal{O}(\log^{d+1}(n) + |Q|)$ time and the value $\sum_{p_i(t) \in Q} d_i$ in $\mathcal{O}(\log^{d+1}(n))$ time.

Besides the two range trees, we maintain a binary search tree T that contains for each point in P a pair consisting of the point's index and its current status (which is either open or closed). T is sorted according to the indices. Thus, we can output the status of a given point in $\mathcal{O}(\log(n))$ time by querying T.

4.1.3 The Invariant

The key idea of our KDS is to keep up one invariant consisting of the following conditions:

(a) for each closed point $p_i(t) \in P(t) \backslash F(t)$, there is an open point $p_j(t) \in F(t)$ with $\tilde{r}_j(t) \leq \tilde{r}_i(t)$ in $\mathcal{C}(p_i(t), 4 \cdot \tilde{r}_i(t))$ and

(b) for each open point $p_i(t) \in F(t)$, there is no other open point $p_j(t) \in F(t)$ with $\tilde{r}_j(t) \leq \tilde{r}_i(t)$ in $\mathcal{C}(p_i(t), 2 \cdot \tilde{r}_i(t))$.

The choice of Conditions (a) and (b) enables our KDS to be stable. Moreover, we will show that, by keeping up Conditions (a) and (b), we maintain, at any point of time t, a set of open facilities $F(t)$ that leads to a total cost which is at most a constant factor larger than the optimal cost. The following argumentation for proving that our KDS maintains a constant-factor approximation is basically the same as in [87] and the proof of Lemma 3.1.9. Only a few minor adaptations to the kinetic setting have been made.

Claim 4.1.9. *Let t be any point of time, and let $p_i(t)$ be any point in $P(t)$. If the invariant is satisfied at the point of time t, then there exists a point $p_j(t) \in F(t)$ such that $\tilde{r}_j(t) \leq \tilde{r}_i(t)$ and $\mathrm{D}(p_i(t), p_j(t)) \leq 64d \cdot r_i(t)$.*

Proof. Since the invariant is satisfied, there is an open facility $p_j(t) \in F(t)$ with radius $\tilde{r}_j(t) \leq \tilde{r}_i(t)$ in $\mathcal{C}(p_i(t), 4 \cdot \tilde{r}_i(t))$ for each point $p_i(t) \in P(t)$. Thus, we get $\mathrm{D}(p_i(t), p_j(t)) \leq \sqrt{d} \cdot 4 \cdot \tilde{r}_i(t)$. Now, due to Lemma 4.1.7, we have

$$\begin{aligned} \mathrm{D}(p_i(t), p_j(t)) &\leq \sqrt{d} \cdot 4 \cdot 2^{3+\lceil \log(\sqrt{d}) \rceil} \cdot r_i(t) \\ &\leq 64d \cdot r_i(t) \ . \end{aligned}$$

□

Claim 4.1.10. *Let t be any point of time, and let $p_i(t)$ and $p_j(t)$ be distinct points in $F(t)$. If the invariant is satisfied at the point of time t, then we have*

$$\mathrm{D}(p_i(t), p_j(t)) > 2 \cdot \max\{r_i(t), r_j(t)\} \ .$$

Proof. Without loss of generality, we assume that $\tilde{r}_j(t) \leq \tilde{r}_i(t)$. From the fact that the invariant is satisfied, it follows that $p_j(t) \notin \mathcal{C}(p_i(t), 2 \cdot \tilde{r}_i(t))$. Otherwise, the point p_i would be closed at the point of time t. Thus, we have

$$\mathrm{D}(p_i(t), p_j(t)) > 2 \cdot \tilde{r}_i(t) \geq 2 \cdot r_i(t)$$

and

$$\mathrm{D}(p_i(t), p_j(t)) > 2 \cdot \tilde{r}_i(t) \geq 2 \cdot \tilde{r}_j(t) \geq 2 \cdot r_j(t) \ ,$$

where $\tilde{r}_i(t) \geq r_i(t)$ and $\tilde{r}_j(t) \geq r_j(t)$ follow from Lemma 4.1.7. □

For any point $p_j(t) \in P(t)$ and an arbitrary set of open facilities $X(t) \subseteq P(t)$, let

$$\mathrm{charge}(p_j(t), X(t)) := \mathrm{D}(p_j(t), X(t)) + \sum_{p_i(t) \in X(t)} \max\{0, r_i(t) - \mathrm{D}(p_i(t), p_j(t))\} \ .$$

Claim 4.1.11. *Let t be any point of time. For an arbitrary set of open facilities $X(t) \subseteq P(t)$, we get*

$$\sum_{p_j(t) \in P(t)} \mathrm{charge}(p_j(t), X(t)) \cdot d_j = \mathrm{FacLoc}(P(t), X(t)) \ .$$

Proof. Due to the definition of charge$(\cdot,\cdot)$ and FacLoc$(\cdot,\cdot)$ and due to Equation (4.1), we get

$$\begin{aligned}
& \sum_{p_j(t)\in P(t)} \text{charge}(p_j(t), X(t)) \cdot d_j \\
= & \sum_{p_j(t)\in P(t)} \mathrm{D}(p_j(t), X(t)) \cdot d_j + \sum_{p_i(t)\in X(t)} \sum_{p_j(t)\in P(t)\cap\mathcal{B}(p_i(t),r_i(t))} (r_i(t) - \mathrm{D}(p_i(t), p_j(t))) \cdot d_j \\
= & \sum_{p_j(t)\in P(t)} \mathrm{D}(p_j(t), X(t)) \cdot d_j + \sum_{p_i(t)\in X(t)} f_i \\
= & \ \text{FacLoc}(P(t), X(t)) \ .
\end{aligned}$$

□

Claim 4.1.12. *Let t be any point of time, let $p_j(t) \in P(t)$ be any point, let $X(t) \subseteq P(t)$ be an arbitrary set of open facilities, and let $p_i(t) \in X(t)$ be any open facility. If we have* $\mathrm{D}(p_j(t), p_i(t)) = \mathrm{D}(p_j(t), X(t))$*, then* $\text{charge}(p_j(t), X(t)) \geq \max\{r_i(t), \mathrm{D}(p_j(t), p_i(t))\}$.

Proof. If $p_j(t) \notin \mathcal{B}(p_i(t), r_i(t))$, then

$$\begin{aligned}
\text{charge}(p_j(t), X(t)) &\geq \mathrm{D}(p_j(t), X(t)) \\
&= \mathrm{D}(p_j(t), p_i(t)) \\
&> r_i(t) \ .
\end{aligned}$$

Otherwise, we have

$$\begin{aligned}
\text{charge}(p_j(t), X(t)) &\geq \mathrm{D}(p_j(t), X(t)) + (r_i(t) - \mathrm{D}(p_j(t), p_i(t))) \\
&= \mathrm{D}(p_j(t), p_i(t)) + (r_i(t) - \mathrm{D}(p_j(t), p_i(t))) \\
&= r_i(t) \\
&\geq \mathrm{D}(p_j(t), p_i(t)) \ .
\end{aligned}$$

□

Claim 4.1.13. *Let t be any point of time, let $p_j(t) \in P(t)$ be any point, and let $p_i(t)$ be any open facility in $F(t)$. If the invariant is satisfied at the point of time t and we have $p_j(t) \in \mathcal{B}(p_i(t), r_i(t))$, then* $\text{charge}(p_j(t), F(t)) \leq r_i(t)$.

Proof. By Claim 4.1.10, there is no open point $p_\ell(t) \in F(t)$ such that we have $i \neq \ell$ and $p_j(t) \in \mathcal{B}(p_\ell(t), r_\ell(t))$. Since $\mathrm{D}(p_j(t), F(t)) \leq \mathrm{D}(p_j(t), p_i(t))$, we obtain

$$\begin{aligned}
\text{charge}(p_j(t), F(t)) &= \mathrm{D}(p_j(t), F(t)) + (r_i(t) - \mathrm{D}(p_j(t), p_i(t))) \\
&\leq \mathrm{D}(p_j(t), p_i(t)) + (r_i(t) - \mathrm{D}(p_j(t), p_i(t))) \\
&= r_i(t) \ .
\end{aligned}$$

□

Claim 4.1.14. *Let t be any point of time, let $p_j(t) \in P(t)$ be any point, and let $p_i(t)$ be any open facility in $F(t)$. If the invariant is satisfied at the point of time t and we have $p_j(t) \notin \mathcal{B}(p_i(t), r_i(t))$, then* $\text{charge}(p_j(t), F(t)) < \text{D}(p_j(t), p_i(t))$.

Proof. The correctness of the claim follows immediately, unless there is a point $p_\ell(t) \in F(t)$ such that $p_j(t) \in \mathcal{B}(p_\ell(t), r_\ell(t))$. If such a point $p_\ell(t)$ exists, then Claims 4.1.10 and 4.1.13 imply $\text{D}(p_i(t), p_\ell(t)) > 2 \cdot \max\{r_i(t), r_\ell(t)\}$ and $\text{charge}(p_j(t), F(t)) \leq r_\ell(t)$. Furthermore, by triangle inequality, we obtain

$$\begin{aligned} \text{D}(p_j(t), p_i(t)) &\geq \text{D}(p_i(t), p_\ell(t)) - \text{D}(p_j(t), p_\ell(t)) \\ &> 2r_\ell(t) - r_\ell(t) \\ &= r_\ell(t) \ , \end{aligned}$$

which proves $\text{charge}(p_j(t), F(t)) \leq r_\ell(t) < \text{D}(p_j(t), p_i(t))$. □

Claim 4.1.15. *Let t be any point of time, let $p_j(t) \in P(t)$ be any point, and let $X(t) \subseteq P(t)$ be an arbitrary set of open facilities. If the invariant is satisfied at the point of time t, then*

$$\text{charge}(p_j(t), F(t)) < (64d + 1) \cdot \text{charge}(p_j(t), X(t)) \ .$$

Proof. Let $p_i(t)$ be some point in $X(t)$ such that we have $\text{D}(p_j(t), p_i(t)) = \text{D}(p_j(t), X(t))$. By Claim 4.1.9, there exists a point $p_\ell(t) \in F(t)$ such that we have $\tilde{r}_\ell(t) \leq \tilde{r}_i(t)$ and $\text{D}(p_i(t), p_\ell(t)) \leq 64d \cdot r_i(t)$.

If $p_j(t) \in \mathcal{B}(p_\ell(t), r_\ell(t))$, then we obtain $\text{charge}(p_j(t), F(t)) \leq r_\ell(t)$ by Claim 4.1.13. Then, we get $r_\ell(t) \leq \tilde{r}_\ell(t) \leq \tilde{r}_i(t) \leq \sqrt{d} \cdot 4 \cdot 2^{3+\lceil \log(\sqrt{d}) \rceil} \cdot r_i(t) \leq 64d \cdot r_i(t)$ due to the arguments above and Lemma 4.1.7. Since Claim 4.1.12 implies $\text{charge}(p_j(t), X(t)) \geq r_i(t)$, we can conclude

$$\begin{aligned} \text{charge}(p_j(t), F(t)) &\leq r_\ell(t) \\ &\leq 64d \cdot r_i(t) \\ &\leq 64d \cdot \text{charge}(p_j(t), X(t)) \ . \end{aligned}$$

This proves the assertion in case that we have $p_j(t) \in \mathcal{B}(p_\ell(t), r_\ell(t))$.

If $p_j(t) \notin \mathcal{B}(p_\ell(t), r_\ell(t))$, then $\text{charge}(p_j(t), F(t)) < \text{D}(p_j(t), p_\ell(t))$ by Claim 4.1.14. Thus, by triangle inequality, we get

$$\begin{aligned} \text{charge}(p_j(t), F(t)) &< \text{D}(p_j(t), p_i(t)) + \text{D}(p_i(t), p_\ell(t)) \\ &\leq \text{D}(p_j(t), p_i(t)) + 64d \cdot r_i(t) \, . \end{aligned}$$

Since the ratio of $\text{D}(p_j(t), p_i(t)) + 64d \cdot r_i(t)$ to the maximum of $r_i(t)$ and $\text{D}(p_j(t), p_i(t))$ is at most $64d + 1$, we obtain $\text{charge}(p_j(t), F(t)) < (64d + 1) \cdot \max\{\text{D}(p_j(t), p_i(t)), r_i(t)\}$. Now, the assertion follows by Claim 4.1.12. □

Lemma 4.1.16. *Let t be any point of time. If the invariant is satisfied at the point of time t, then we have*

$$\text{FacLoc}(P(t), F(t)) < (64d + 1) \cdot \text{FacLoc}(P(t), F^*(t)) \, .$$

Proof. Due to Claims 4.1.11 and 4.1.15, we have

$$\begin{aligned}\mathrm{FacLoc}(P(t), F(t)) &= \sum_{p_j(t)\in P(t)} \mathrm{charge}(p_j(t), F(t)) \cdot d_j \\ &< \sum_{p_j(t)\in P(t)} (64d+1) \cdot \mathrm{charge}(p_j(t), X(t)) \cdot d_j \\ &= (64d+1) \cdot \mathrm{FacLoc}(P(t), X(t))\end{aligned}$$

for an arbitrary set of open facilities $X(t) \subseteq P(t)$. Thus, the approximation factor is also true for an optimal set of open facilities $F^*(t)$, which completes the proof of the lemma. □

4.2 The Kinetic Data Structure

This section addresses the design of our KDS for the mobile facility location problem. After describing how to compute an initial set of open facilities, we describe how the event queue is structured and how an update of the KDS is processed.

4.2.1 Initial Set of Open Facilities

Let $p_i(t_0)$ denote the initial position of the point $p_i \in P$. To compute an initial set of open facilities, we apply Algorithm 4.2.1, which is a modified version of Algorithm 2.3.1, on the point set $P(t_0)$. The modification is that, instead of considering exactly the sorted sequence of the $r_i(t_0)$ values, we round each $r_i(t_0)$ to one of the $\mathcal{O}(\log(nR))$ possible values for the special radii (i.e., compute its corresponding $\tilde{r}_i(t_0)$ value) and use the sorted sequence of the rounded values.

Algorithm 4.2.1 MODIFIED-METTU-PLAXTON-FACLOC(P, t_0)

```
1: calculate the radius r̃_i(t_0) for each point p_i(t_0) ∈ P(t_0)
2: for k ← ⌈log(r_min)⌉ + ⌈log(4√d)⌉ to ⌈log(r_max)⌉ + ⌈log(4√d)⌉ do
3:    let I_k be the set of indices of all the points with radius 2^k
4:    for each i ∈ I_k do
5:       if there is no open facility in C(p_i(t_0), 2 · 2^k) then
6:          open facility at p_i(t_0)
```

4.2.2 Event Queue

In order to maintain the invariant defined in Section 4.1.3, we have to update our KDS at certain points of time. More precisely, we perform an update at each point of time when a point $p_j(t)$ crosses a wall $\mathcal{W}_{i,k}(t)$, $\lceil\log(r_{\min})\rceil + \lceil\log(4\sqrt{d})\rceil \le k \le \lceil\log(r_{\max})\rceil + \lceil\log(4\sqrt{d})\rceil$, of another point $p_i(t)$.

To keep track of these events, we use the following data structure: For each dimension ℓ, $1 \le \ell \le d$, we store all n points and all $\mathcal{O}(n \cdot \log(nR))$ wall faces that are orthogonal to

the ℓ-th coordinate axis in a list sorted by the ℓ-th coordinate. For each consecutive pair in each of the d lists, we keep up one certificate to certify the sorted order of the lists. We define the failure time of the certificate for any pair of consecutive objects to be the first future point of time when these objects change their ranks in their sorted list. The failure times of all certificates are maintained in one event queue.

In case that more than one event occurs at the same time, we handle them in an arbitrary order. Certainly, it is not the case that each event implicates that a point crosses a wall of another point (as, e.g., the change of the rank of two wall faces also causes an event), but definitely every crossing of a wall is discovered by a failure of at least one certificate. The event queue has the following complexity:

Lemma 4.2.1. *The event queue has size $\mathcal{O}(n \log(nR))$, can be initialized in $\mathcal{O}(n \log^2(nR))$ time, and can be updated in $\mathcal{O}(\log(nR))$ time. Provided that the trajectories can be described by bounded-degree polynomials, the total number of events is $\mathcal{O}(n^2 \log^2(nR))$. A flight plan update involves $\mathcal{O}(\log(nR))$ certificates and requires $\mathcal{O}(\log^2(nR))$ time.*

Proof. Each of the d lists stores n points and $\mathcal{O}(n \log(nR))$ wall faces. It follows that the event queue holds $\mathcal{O}(n \log(nR))$ events. Thus, the upper bound on the space requirement is as claimed.

The initialization of the d lists and the event queue can be done by simple sorting operations in $\mathcal{O}(n \log(nR) \log(n \log(nR))) \subset \mathcal{O}(n \log^2(nR))$ time. In each following update, we have to re-calculate the points of time when the two objects involved in the current event change their ranks with their two neighbors in the corresponding list. Thus, a constant number of events have to be updated in the event queue. Since the event queue contains $\mathcal{O}(n \log(nR))$ elements and we can use a min-heap to realize it, an update of an event requires $\mathcal{O}(\log(n \log(nR))) \subset \mathcal{O}(\log(nR))$ time. Furthermore, a flight plan update of a point causes a re-calculation of the points of time when the point and all its wall faces change their ranks with the associated neighbors in all d lists. Afterwards, the involved certificates are updated in the event queue. Since a point has $\mathcal{O}(\log(nR))$ wall faces, the number of involved certificates is $\mathcal{O}(\log(nR))$. Their update in the event queue can be accomplished in $\mathcal{O}(\log^2(nR))$ time.

In case that the trajectories can be described by bounded-degree polynomials and no flight plan update occurs, the upper bound on the total number of events is given as follows. For each pair of elements, an event occurs when the trajectories of the two elements cross each other. The number of cuts of two polynomials is bounded by the maximum degree of both polynomials. Hence, the total number of cuts of $\mathcal{O}(n \log(nR))$ bounded-degree polynomials is $\mathcal{O}(cn^2 \log^2(nR))$, where the constant c is the maximum degree of the polynomials. □

4.2.3 Handling an Update

In this section, we describe how an event E, occurring at any point of time t, is handled (confer Algorithm 4.2.2, ll. 5). As the first step, the event queue is updated as explained in Section 4.2.2. Then, we have to distinguish between the following three cases:

(*i*) Both objects involved in the considered certificate are faces of walls.

(*ii*) Both objects involved in the considered certificate are points.

(*iii*) One object involved in the considered certificate is a point and the other object is a face of a wall.

The handling of the three cases mainly depends on whether the invariant is violated or not. We say that a point $p_i(t) \in P(t)$ violates the invariant at a point of time t if either (a) $p_i(t)$ is closed, but there is no open facility with radius smaller than or equal to $r_i(t)$ in the cube $\mathcal{C}(p_i(t), 4 \cdot \tilde{r}_i(t))$ or (b) $p_i(t)$ is open, but there is another open facility with radius smaller than or equal to $r_i(t)$ in the cube $\mathcal{C}(p_i(t), 2 \cdot \tilde{r}_i(t))$. We assume that the invariant is satisfied by the time when E occurs.

In Case (*i*), no point crosses the wall of another point. As a result, the invariant is still satisfied, so handling E is completed.

In Case (*ii*), the event indicates that a point $p_i(t)$ and another point $p_j(t)$ change their ranks based on a dimension ℓ, $1 \leq \ell \leq d$. This means that we have to update the position of p_i and p_j in the range trees T_1 and T_2. Since no point crosses a wall of another point, handling E is then completed.

In Case (*iii*), it might be that the invariant is violated. Let $p_j(t)$ be the first object involved in the considered certificate, and let $p_i(t)$ be the point whose wall is the second object involved in the considered certificate. In case that $p_j(t)$ does not cross a wall of $p_i(t)$, handling E is completed. Otherwise, we update the radius $\tilde{r}_i(t)$ according to Definition 4.1.1, i.e., we set $\tilde{r}_i(t) = 2^{\tilde{k}}$ such that $\tilde{k} = k_0 + \lceil \log(4\sqrt{d}) \rceil$ and k_0 is the minimum integer k, $\lceil \log(r_{\min}) \rceil \leq k \leq \lceil \log(r_{\max}) \rceil$, with $\text{weight}(\mathcal{C}(p_i(t), 2^{k_0})) \geq f_i \cdot 2^{-k_0}$. We will show that the new value of k_0 differs from its old value (before event E occurred) by at most 1. Thus, there are three possible values for k_0. Each of these values can be tested by one range query on both T_1 and T_2. Afterwards, we test if $p_i(t)$ violates the invariant by using a range query on T_1. If this is the case, we change the status of $p_i(t)$. As an effect of changing the radius or the status of one point, the invariant may be violated by many other points (e.g., their open facility has been closed). In the following, we will show how to deal with this problem (confer Algorithm 4.2.3).

Algorithm Restore. Suppose that $p_i(t)$ is a point that triggered an event E at a point of time t and whose radius or status changed due to E. Let $\tilde{r}_i(t) = 2^{\tilde{k}}$ be its updated radius. First, we restore the invariant at all points with radius $2^{\tilde{k}-1}$ to ensure that no point with radius less than or equal to $2^{\tilde{k}-1}$ violates the invariant. Then, we handle all points with radius $2^{\tilde{k}}$ that violates the invariant, then the points with radius $2^{\tilde{k}+1}$, ..., up to the biggest possible radius. Now, we describe the procedure in general for any radius 2^k.

We define two cubes $S_1 := \mathcal{C}(p_i(t), 4 \cdot 2^{k+1})$ and $S_2 := \mathcal{C}(p_i(t), 6 \cdot 2^{k+1})$. Both cubes are divided into equally sized cubelets with radius 2^k. The left hand side of Figure 4.2 illustrates this decomposition in the plane.

To guarantee that no open point with radius 2^k violates the invariant, we proceed as follows with each cubelet in S_1: Let m be the center point of the considered cubelet. If

Algorithm 4.2.2 KineticFL(P, t_0)

```
1: Modified-Mettu-Plaxton-FacLoc(P, t_0)
2: initialize event queue Q
3: while Q is not empty do
4:   E ← dequeue(Q)
5:   update Q
6:   if E indicates that p_i(t) and p_j(t) change their ranks in any list for any i, j then
7:     update position of p_i and p_j in T_1 and T_2
8:   else
9:     if E indicates that p_j(t) crosses a wall of p_i(t) for any i, j then
10:      update r̃_i(t) ← 2^k̃ in T_1 and T_2
11:      if p_i(t) violates the invariant then
12:        change status of p_i(t)
13:      if radius or status of p_i(t) changed then
14:        Restore(p_i(t), k̃)
```

Algorithm 4.2.3 Restore$(p_i(t), \tilde{k})$

```
1: for k ← k̃ − 1 to ⌈log(r_max)⌉ + ⌈log(4√d)⌉ do
2:   define cubes S_1 ← C(p_i(t), 4 · 2^{k+1}) and S_2 ← C(p_i(t), 6 · 2^{k+1})
3:   for each cubelet C with center m_C and radius 2^k in S_1 do
4:     if ∃ open facility with radius < 2^k in C(m_C, 3 · 2^k) then
5:       close all facilities with radius 2^k in C
6:   for each cubelet C with center m_C and radius 2^k in S_2 do
7:     if ∄ open facility with radius ≤ 2^k in C(m_C, 3 · 2^k) then
8:       open one point with radius 2^k in C (if existing)
```

there is an open facility with radius less than 2^k in $\mathcal{C}(m, 3 \cdot 2^k)$, then we close all facilities with radius 2^k in $\mathcal{C}(m, 2^k)$. Note that there is at most one such facility. The considered area around a cubelet is illustrated in Figure 4.2.

In order to ensure that no closed point with radius 2^k violates the invariant neither, we proceed as follows with each cubelet in S_2: Let m be the center point of the considered cubelet. If there does not exist an open facility with radius less than or equal to 2^k in $\mathcal{C}(m, 3 \cdot 2^k)$, then we open a point with radius 2^k in the cubelet (if there is such a point). No matter, whether we opened a point or not, it is guaranteed that, for each closed point $p_j(t)$ with $\tilde{r}_j(t) = 2^k$ in the cubelet, there is an open facility in $\mathcal{C}(p_j(t), 4 \cdot \tilde{r}_j(t))$.

4.3 Quality and Complexity of the Kinetic Data Structure

At first, we prove that our KDS maintains a subset of the moving input points as open facilities such that, at any time, the associated total cost is at most a constant factor larger

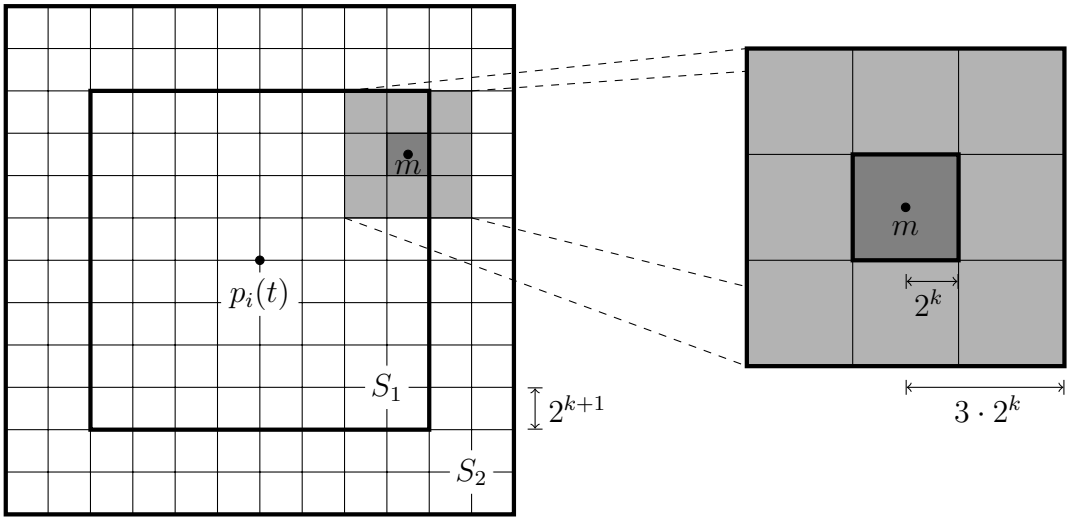


Figure 4.2: Illustration of the decomposition into cubelets and the tested area for a cubelet. The shown decomposition is used during the iteration of algorithm Restore that restores the invariant at all points with radius 2^k. The cubes S_1 and S_2 are indicated by thick lines. For each cubelet in S_1 and S_2, we perform a test. The shaded area indicates the tested area $\mathcal{C}(m, 3 \cdot 2^k)$ for one cubelet in S_1. This area is magnified on the right hand side of the figure, where the dark shaded area corresponds to the tested cubelet $\mathcal{C}(m, 2^k)$.

than the current optimal cost. For that purpose, we show that we restore the invariant each time it is violated. Finally, we analyze the complexity of our KDS.

4.3.1 Maintenance of the Invariant

To simplify the description of the following proofs, we assume that at most one event occurs at the same time. Assuming this, we can show that the invariant is always satisfied after our KDS has handled an event. In case that more than one event occurs at the same time, the following proofs would differ in the sense that the fulfillment of the invariant can be guaranteed only after our KDS has handled all of these events.

First, we prove that the invariant is satisfied as long as algorithm KineticFL does not call algorithm Restore.

Lemma 4.3.1. *The invariant is satisfied after the first step of algorithm* KineticFL.

Proof. Since algorithm Modified-Mettu-Plaxton-FacLoc treats the points in non-decreasing order according to their special radii and opens a point $p_i(t_0)$ with radius $\tilde{r}_i(t_0)$ if and only if there is no other open point in $\mathcal{C}(p_i(t_0), 2 \cdot \tilde{r}_i(t_0))$, no open point violates the invariant.

Furthermore, algorithm Modified-Mettu-Plaxton-FacLoc does not open a point $p_i(t_0)$ with radius $\tilde{r}_i(t_0)$ if and only if there is another open point in $\mathcal{C}(p_i(t_0), 2 \cdot \tilde{r}_i(t_0)) \subseteq$

$\mathcal{C}(p_i(t_0), 4 \cdot \tilde{r}_i(t_0))$. Because this point has been treated earlier than $p_i(t_0)$, its radius is less than or equal to $\tilde{r}_i(t_0)$. Thus, there exists an open point with radius less than or equal to $\tilde{r}_i(t_0)$ in $\mathcal{C}(p_i(t_0), 4 \cdot \tilde{r}_i(t_0))$. Hence, no closed point violates the invariant. □

Claim 4.3.2. *Let E be any event such that algorithm* KineticFL *does not change the radius or the status of any point. If the invariant is satisfied before E, then it holds after E as well.*

Proof. We have to consider two cases. In the first case, no point crosses a wall of another point. This implies that no point enters or leaves any cube of another point and no point changes its radius. Hence, the invariant is still valid and the claim holds.

Let t be the point of time when event E occurs. Then, in the second case, we have that a wall $\mathcal{W}_{i,k}(t)$ of a point $p_i(t)$ is crossed by another point $p_j(t)$, but our algorithm does not change the radius or the status of $p_i(t)$. It follows that neither $p_i(t)$ changed its radius nor $p_i(t)$ violates the invariant because otherwise our algorithm would have changed the radius and the status of $p_i(t)$, respectively. Due to the fact that $p_i(t)$ is unchanged and only the wall $\mathcal{W}_{i,k}(t)$ is crossed at the point of time t, no point in $P(t)\backslash\{p_i(t)\}$ violates the invariant neither. This completes the proof. □

Next, we prove that the updated radius of a point that triggered an event E differs at most by a factor of 2 from its value before E.

Claim 4.3.3. *Let E be an event at any point of time t where any point $p_j(t) \in P(t)$ crosses any wall of any other point $p_i(t) \in P(t)$. Let $t' < t$ be any point of time after the latest point of time when p_i has been involved in one event. We get $1/2 \cdot \tilde{r}_i(t') \leq \tilde{r}_i(t) \leq 2 \cdot \tilde{r}_i(t')$.*

Proof. Let k'_0 and k_0 be the minimum integers k with $\lceil \log(r_{\min}) \rceil \leq k \leq \lceil \log(r_{\max}) \rceil$ for which we have $\text{weight}(\mathcal{C}(p_i(t'), 2^{k'_0})) \geq f_i \cdot 2^{-k'_0}$ and $\text{weight}(\mathcal{C}(p_i(t), 2^{k_0})) \geq f_i \cdot 2^{-k_0}$, respectively. Note that the existence of k'_0 and k_0 is due to Corollary 4.1.4. Furthermore, let $\mathcal{W}_{i,\ell}(t)$ be the wall that is crossed by $p_j(t)$. We have to consider the cases (i) $p_j(t)$ leaves the cube $\mathcal{C}(p_i(t), 2^\ell)$ and (ii) $p_j(t)$ enters the cube $\mathcal{C}(p_i(t), 2^\ell)$.

Case (*i*). Since the point of time t', p_j is the only point that has crossed a wall of p_i. It follows that $\text{weight}(\mathcal{C}(p_i(t), 2^m)) < f_i \cdot 2^{-m}$, for any $m < k'_0$, and $\text{weight}(\mathcal{C}(p_i(t), 2^{k'_0})) \leq \text{weight}(\mathcal{C}(p_i(t'), 2^{k'_0}))$. This implies $k_0 \geq k'_0$.

Since $p_j(t)$ has only crossed one wall of $p_i(t)$, we get

$$\text{weight}(\mathcal{C}(p_i(t), 2^{k'_0+1})) \geq \text{weight}(\mathcal{C}(p_i(t'), 2^{k'_0})) \geq f_i \cdot 2^{-k'_0} \geq f_i \cdot 2^{-(k'_0+1)} \ ,$$

where the second inequality is given by the definition of k'_0. Thus, we have $k_0 \leq k'_0 + 1$. Overall, we obtain $k'_0 \leq k_0 \leq k'_0 + 1$ in Case (i).

Case (*ii*). Due to the fact that $p_j(t)$ is the only point that has crossed a wall of $p_i(t)$ and $p_j(t)$ enters a cube with center $p_i(t)$, we have weight$(\mathcal{C}(p_i(t), 2^m)) \geq$ weight$(\mathcal{C}(p_i(t'), 2^m))$, for all possible values of m. Hence, we get $k_0 \leq k_0'$.

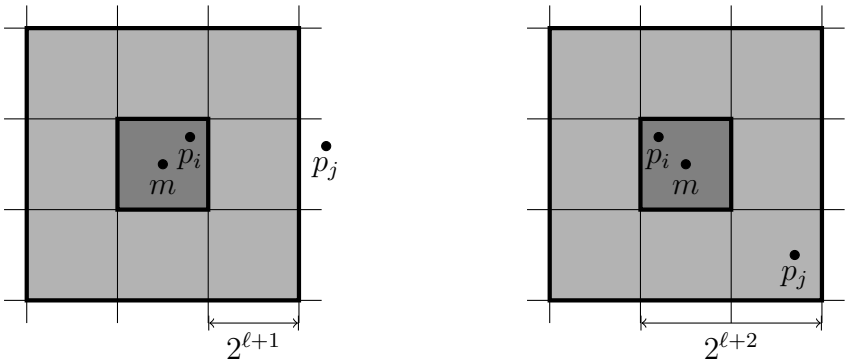
Recall that $p_j(t)$ crosses the wall $\mathcal{W}_{i,\ell}(t)$. If $\ell \geq k_0' - 1$, then $k_0 \geq k_0' - 1$ follows obviously. Now, let us assume that $\ell < k_0' - 1$ and $k_0 = \ell$. Due to this assumption, we obtain that weight$(\mathcal{C}(p_i(t), 2^\ell)) \geq f_i \cdot 2^{-\ell}$. Since p_j is the only point that has crossed a wall of p_i, we also have weight$(\mathcal{C}(p_i(t'), 2^{\ell+1})) \geq f_i \cdot 2^{-\ell} \geq f_i \cdot 2^{-(\ell+1)}$. This implies $k_0' \leq \ell + 1$, which is a contradiction. Hence, we get $k_0' - 1 \leq k_0 \leq k_0'$ in Case (ii).

Considering both cases, we get $k_0' - 1 \leq k_0 \leq k_0' + 1$. Now, the claim follows due to the definition of the special radii. □

The following claims show that the invariant is restored after each call of algorithm Restore.

Claim 4.3.4. *Let $p_h(t)$ be a point that triggered an event E and whose radius or status changed due to E. Let $\tilde{r}_h(t) = 2^{\tilde{k}}$ be the updated radius of $p_h(t)$. If no point with radius less than or equal to $2^{\tilde{k}-2}$ violates the invariant before E, then this holds after E as well.*

Proof. Due to Claim 4.3.3, the radius of p_h has been at least $2^{\tilde{k}-1}$ before E. While processing event E, we only change the status of points with radius larger than or equal to $2^{\tilde{k}-1}$. These status changes cannot affect the invariant at points with radius less than or equal to $2^{\tilde{k}-2}$. Thus, the assertion follows. □



(a) (b)

Figure 4.3: The dark gray area indicates the cube $\mathcal{C}(m, 2^\ell)$ in S_2 that contains $p_i(t)$ during running the outer **for**-loop of algorithm Restore for $k = \ell$. The light gray area indicates the cube $\mathcal{C}(m, 3 \cdot 2^\ell)$. (a) Arrangement of points that leads to the desired contradiction in the proof of Case (i) in Claim 4.3.5. (b) Arrangement of points that leads to the desired contradiction in the proof of Case (i) in Claim 4.3.6.

Claim 4.3.5. *Let $p_h(t)$ be a point that triggered an event E and whose radius or status changed due to E. Let $\tilde{r}_h(t) = 2^{\tilde{k}}$ be the updated radius of $p_h(t)$. If the invariant is satisfied before E and no open point with radius less than or equal to $2^{\ell-1}$ violates the invariant before running the outer* ***for****-loop of algorithm* RESTORE *for $k = \ell$, $\tilde{k} - 1 \leq \ell \leq \lceil \log(r_{\max}) \rceil + \lceil \log(4\sqrt{d}) \rceil$, then, after running this* ***for****-loop, no open point with radius 2^{ℓ} violates the invariant.*

Proof. The proof is by contradiction. Let us assume that, after running the outer **for**-loop of algorithm RESTORE for $k = \ell$, there is an open point $p_i(t)$ with radius $\tilde{r}_i(t) = 2^{\ell}$ that has another open point $p_j(t)$ with radius $\tilde{r}_j(t) \leq \tilde{r}_i(t)$ in $\mathcal{C}(p_i(t), 2 \cdot \tilde{r}_i(t))$. We have to consider the cases (i) $p_i(t) \in S_2$ and (ii) $p_i(t) \notin S_2$.

Case (*i*). Subcase $\tilde{r}_j(t) < \tilde{r}_i(t)$: Due to the fact that $\tilde{r}_j(t) < 2^{\ell}$, we have opened p_j before running the outer **for**-loop for $k = \ell$. It follows that $p_i(t) \in \mathcal{C}(m, 2^{\ell})$ and $p_j(t) \notin \mathcal{C}(m, 3 \cdot 2^{\ell})$ for one center m of a considered cubelet (see Figure 4.3 (a)) because otherwise we either would have closed $p_i(t)$ or would not have opened $p_i(t)$. Thus, we have $p_j(t) \notin \mathcal{C}(p_i(t), 2^{\ell+1}) = \mathcal{C}(p_i(t), 2 \cdot \tilde{r}_i(t))$, which is a contradiction to the assumption made above.

Subcase $\tilde{r}_j(t) = \tilde{r}_i(t)$: We have to consider the case that neither p_i nor p_j is opened while running the outer **for**-loop for $k = \ell$ and the case that at least one of p_i and p_j is opened during this **for**-loop. In the first case, it follows that p_i and p_j must have been open before running the outer **for**-loop for $k = \ell$. It follows that both points have been open before E or one point is p_h. Then, either the invariant has been violated before E, which is a contradiction to the precondition of the claim, or changing the status of p_h violated the invariant, which means that a rule of the algorithm has been broken. In the latter case, we have opened p_i or p_j or both while running the outer **for**-loop for $k = \ell$. Without loss of generality, let us assume that we have opened p_j before we have opened p_i. Then, we must have that $p_i(t) \in \mathcal{C}(m, 2^{\ell})$ and $p_j(t) \notin \mathcal{C}(m, 3 \cdot 2^{\ell})$ for one center m of a considered cubelet (see Figure 4.3 (a)). It follows that $p_j(t) \notin \mathcal{C}(p_i(t), 2^{\ell+1}) = \mathcal{C}(p_i(t), 2 \cdot \tilde{r}_i(t))$, which is a contradiction to the assumption made above.

Case (*ii*). Subcase $\tilde{r}_j(t) < \tilde{r}_i(t)$: Due to the fact that $\tilde{r}_j(t) < 2^{\ell}$, we have opened p_j before running the outer **for**-loop for $k = \ell$. Furthermore, it follows from $p_i(t) \notin S_2$ that we must have opened p_i before running the outer **for**-loop for $k = \ell$ as well. Hence, both p_i and p_j have been open before running this **for**-loop. Thus, the invariant must have been violated at point $p_j(t)$ with $\tilde{r}_j(t) \leq 2^{\ell-1}$ before running the outer **for**-loop for $k = \ell$, which is a contradiction to the precondition of the claim.

Subcase $\tilde{r}_j(t) = \tilde{r}_i(t)$: We can use the same argumentation as in subcase $\tilde{r}_j(t) = \tilde{r}_i(t)$ of Case (i) with the modification that we know that p_i has been opened before running the outer **for**-loop for $k = \ell$. The reason is that $p_i(t) \notin S_2$, so we do not change its status while running this **for**-loop. □

Claim 4.3.6. *Let $p_h(t)$ be a point that triggered an event E and whose radius or status changed due to E. Let $\tilde{r}_h(t) = 2^{\tilde{k}}$ be the updated radius of $p_h(t)$. If the invariant is satisfied*

*before E and no closed point with radius less than or equal to $2^{\ell-1}$ violates the invariant before running the outer **for**-loop of algorithm* RESTORE *for $k = \ell$, where $\tilde{k} - 1 \leq \ell \leq \lceil \log(r_{\max}) \rceil + \lceil \log(4\sqrt{d}) \rceil$, then, after running this **for**-loop, no closed point with radius 2^{ℓ} violates the invariant.*

Proof. The proof is by contradiction. Let us assume that, after running the outer **for**-loop of algorithm RESTORE for $k = \ell$, there is a closed point $p_i(t)$ with radius $\tilde{r}_i(t) = 2^{\ell}$ that has no open point with radius less than or equal to $\tilde{r}_i(t)$ in $\mathcal{C}(p_i(t), 4 \cdot \tilde{r}_i(t))$. We have to consider the cases *(i)* $p_i(t) \in S_2$ and *(ii)* $p_i(t) \notin S_2$.

Case (*i*). Due to our construction, we have $p_i(t) \in \mathcal{C}(m, 2^{\ell})$ and there is an open point $p_j(t)$ with radius at most 2^{ℓ} in $\mathcal{C}(m, 3 \cdot 2^{\ell})$ for any center m of a considered cubelet (see Figure 4.3 (b)) because otherwise we would have opened a point with radius 2^{ℓ} in $\mathcal{C}(m, 2^{\ell})$. Note that, in case there is no other point with radius at most 2^{ℓ} in $\mathcal{C}(m, 2^{\ell})$ except $p_i(t)$, we would have opened p_i and $p_j = p_i$. Thus, we have $p_j(t) \in \mathcal{C}(p_i(t), 2^{\ell+2}) = \mathcal{C}(p_i(t), 4 \cdot \tilde{r}_i(t))$, which is a contradiction to the assumption made above.

Case (*ii*). Let t' be any point of time between the occurrence of E and the latest event before. Then, there was an open point $p_j(t')$ with radius less than or equal to $\tilde{r}_i(t')$ in the cube $\mathcal{C}(p_i(t'), 4 \cdot \tilde{r}_i(t'))$ because otherwise the invariant was violated before E. Since E had no influence on the radius of p_i, we have $\tilde{r}_i(t') = \tilde{r}_i(t) = 2^{\ell}$.

First, let us assume that $p_j = p_h$. Since $p_i(t) \notin S_2 = \mathcal{C}(p_h(t), 6 \cdot 2^{\ell+1})$, we have $p_j(t) \notin \mathcal{C}(p_i(t), 6 \cdot 2^{\ell+1})$. From $p_j(t') \in \mathcal{C}(p_i(t'), 4 \cdot \tilde{r}_i(t')) = \mathcal{C}(p_i(t'), 4 \cdot 2^{\ell})$ and $p_j(t) \notin \mathcal{C}(p_i(t), 6 \cdot 2^{\ell+1})$ follows that p_j must have crossed the wall $\mathcal{W}_{i,\ell+3}(t'')$ at a time t'' with $t' < t'' < t$. This implies an event at time t'', which is a contradiction to the definition of t'. Thus, we have $p_j \neq p_h$.

Due to $p_i \neq p_h$, $p_j \neq p_h$, and $p_j(t') \in \mathcal{C}(p_i(t'), 4 \cdot \tilde{r}_i(t'))$, $p_j(t) \in \mathcal{C}(p_i(t), 4 \cdot \tilde{r}_i(t))$ must also be true. Thus, if p_i violates the invariant after E, then we must have closed p_j during processing E. We only close points with radius less than or equal to $\tilde{r}_i(t)$ in S_1, so we must have $p_j(t) \in S_1$. Since $p_i(t) \notin S_2$ and $p_j(t) \in S_1$, we get $p_j(t) \notin \mathcal{C}(p_i(t), 2 \cdot 2^{\ell+1}) = \mathcal{C}(p_i(t), 4 \cdot \tilde{r}_i(t))$, which is a contradiction. □

Now, we can combine the obtained results to get the following lemma:

Lemma 4.3.7. *The invariant is satisfied after algorithm* KINETICFL *has handled an event.*

Proof. Due to Lemma 4.3.1 and Claim 4.3.2, the invariant is satisfied as long as we do not call algorithm RESTORE. Now, we show by induction that the invariant is also satisfied after running algorithm RESTORE.

Let $p_h(t)$ be the point whose radius or status changed due to an event E, and let $\tilde{r}_h(t) = 2^{\tilde{k}}$ be its updated radius. Due to the precondition given above and Claim 4.3.4, the assertion is true for all points with radius at most $2^{\tilde{k}-2}$. This proves the base case. By induction hypothesis, the preconditions of Claims 4.3.5 and 4.3.6 hold for any ℓ with

$\tilde{k} - 1 \leq \ell \leq \lceil \log(r_{\max}) \rceil + \lceil \log(4\sqrt{d}) \rceil$. This means that the assertion holds for all points with radius at most $2^{\ell-1}$. It follows from Claims 4.3.5 and 4.3.6 that the assertion also holds for all points with radius at most 2^{ℓ}, which completes the proof of the lemma. □

Due to Lemmas 4.3.1, 4.3.7 and 4.1.16, we get the following result:

Lemma 4.3.8. *The KDS for the mobile facility location problem in $\mathbb{R}^d$ maintains at each point of time t a subset of open facilities $F(t) \subseteq P(t)$ such that we have*

$$\text{FacLoc}(P(t), F(t)) < (64d + 1) \cdot \text{FacLoc}(P(t), F^*(t)) \ .$$

4.3.2 Complexity

In the remainder of this chapter, we analyze our KDS in terms of its compactness, locality, responsiveness, and efficiency (see Section 2.4.3 for definitions of these attributes). Lemma 4.2.1 already implies that our KDS is compact and local. Next, we prove that the requirement for being responsive and efficient is also fulfilled.

Lemma 4.3.9. *Each update operation requires $\mathcal{O}(\log^{d+1}(n) \cdot \log(nR))$ time and $\mathcal{O}(\log(nR))$ status changes.*

Proof. Due to Lemma 4.2.1, the time to update the event queue is $\mathcal{O}(\log(nR))$. Except for algorithm RESTORE, all further steps require a constant number of range queries on T_1 and T_2. Due to Lemma 4.1.8, this requires $\mathcal{O}(\log^{d+1}(n))$ time. Next, we examine the time needed for algorithm RESTORE. We consider the running time resulting for restoring the invariant at points with radius 2^k. The number of cubelets with radius 2^k in $\mathcal{C}(p_h(t), 6 \cdot 2^{k+1})$ is 12^d, where $p_h(t)$ is the point that triggered the event. The query of open or closed points for one cubelet can be answered by one range query on T_1 or T_2. Due to Lemma 4.1.8, this requires $\mathcal{O}(\log^{d+1}(n))$ time. Afterwards, there has to be at most one point inserted and deleted in T_1 and T_2, which can be done in $\mathcal{O}(\log^{d+1}(n))$ time according to Lemma 4.1.8. By summation over all radii, we get a total running time of $\mathcal{O}(\log^{d+1}(n) \cdot \log(nR))$.

There can exist at most one open facility with radius 2^k in a cubelet with radius 2^k because otherwise at least one open facility would violate the invariant. Hence, the number of open facilities with radius 2^k that are closed while running algorithm RESTORE is constant. Furthermore, we open at most one facility in each cubelet, so the number of opened facilities with radius 2^k is also constant. Due to the fact that we handle $\mathcal{O}(\log(nR))$ radii, there are $\mathcal{O}(\log(nR))$ status changes per event. □

Since our KDS processes a total number of $\mathcal{O}(n^2 \log^2(nR))$ events (see Lemma 4.2.1), the total processing time is bounded by $\mathcal{O}(n^2 \log^{d+1}(n) \cdot \log^3(nR))$. To measure the efficiency as defined in Section 2.4.3, we use a result from [46]. In [46], Gao et al. investigated a problem in the KDS framework which is closely related to the mobile facility location problem. In particular, they provided a randomized KDS to maintain a set of centers among moving points in the plane such that, given a specified radius, all the points are covered by balls of the given radius centered at the chosen center points. Gao et al. showed that the size

of the center set is at most a constant factor larger than the minimum one. To prove the efficiency of their KDS, they showed that there is a set of n points moving linearly on the real line that forces any c-approximate cover to change $\Omega(n^2/c^2)$ times. With some minor modifications, their result can be transferred to the facility location problem.

Lemma 4.3.10. *For any constant $c > 1$, there exists a set P of n points moving linearly on the real line such that any c-approximate solution to the mobile facility location problem for P undergoes $\Omega(n^2/c^2)$ status changes.*

Proof. We assume that c is an integer and $n = 2cm$ with $m \geq 12c^2$ being also an integer. Let P be the set of n moving points which is defined as follows. We partition P into m groups, each containing $2c$ points. Let the j-th point in the i-th group be denoted by $p_{i,j}$, where $0 \leq i < m$ and $0 \leq j < 2c$. The initial position of all the points in the i-th group is $i \cdot 2m$. Now, we let the point $p_{i,j}$ move with speed $j \cdot 2m$. Let $p_{i,j}(t)$ be the position of $p_{i,j}$ at the point of time t. Then, we have

$$p_{i,j}(t) = (i + jt) \cdot 2m \ ,$$

for $0 \leq i < m$, $0 \leq j < 2c$, and $t \geq 0$. Note that, in the time period from 0 to m, the points often change their ranks on the line. Afterwards, no two points will change their rank any more.

Let us consider the configuration of P at any point of time $t_1 := k + 3c/m$, for some integer $k < m$. At the point of time t_1, the location of the point $p_{i,j}$ is

$$p_{i,j}(t_1) = \left(i + jk + \frac{3cj}{m}\right) \cdot 2m = 2(i + jk)m + 6cj \ .$$

Let $p_{i,j}$ and $p_{i',j'}$ be any two distinct points. In case that $i + jk \neq i' + j'k$, the distance between $p_{i,j}$ and $p_{i',j'}$ is

$$|p_{i,j}(t_1) - p_{i',j'}(t_1)| > 2m - 12c^2 \geq 4c$$

at the point of time t_1. In case that $i + jk = i' + j'k$, we have $j' \neq j$ since $p_{i,j}$ and $p_{i',j'}$ are distinct. Then, it follows that the distance between $p_{i,j}$ and $p_{i',j'}$ is

$$|p_{i,j}(t_1) - p_{i',j'}(t_1)| \geq 6c$$

at the point of time t_1. Thus, at the point of time t_1, no two points are within distance $4c$ of each other.

Assuming that the opening costs as well as the demands of all the points in P are 1, we next analyze an optimal solution for P at the point of time t_1. Since the distance between any two points in P is greater then $4c$ at the point of time t_1, the only existing optimal solution is to open a facility at each input point. This leads to a total cost of n. It follows that any c-approximate solution can have a cost of at most cn. Let us now consider an approximate solution in which only a $(1-\alpha)$-fraction of the input points are open facilities. Then, the cost for this solution is more than $n - \alpha n + 4c \cdot \alpha n$ since the distance between

any two points is greater than $4c$ at the point of time t_1. To ensure that the cost is at most cn, α must be smaller than $(c-1)/(4c-1)$. Since $1/4 > (c-1)/(4c-1)$ for $c > 1$, we obtain that any c-approximate solution must open more than $3n/4$ facilities.

Next, we consider the configuration of P at any point of time $t_2 := k$, for some integer $k < m$. Since $p_{i,j}(t_2) = (i + jk) \cdot 2m$, where $0 \leq i < m$, $0 \leq j < 2c$, and $k < m$, each point is located at a position $2sm$ for some $s \in \{0, \ldots, m + 2ck\}$. It follows that, at the point of time t_2, there exist at most $m + 2ck$ open facilities in an optimal solution, and the optimal facility location cost is at most $m + 2ck$. Thus, a c-approximate solution may have at most $c(m + 2ck)$ open facilities.

Hence, between the points of time t_1 and t_2, any c-approximate solution undergoes at least $3n/4 - c(m + 2ck) = n/4 - 2c^2k$ status changes. Summing up over all $k \in \{0, \ldots, K-1\}$, the number of status changes is at least

$$\sum_{k=0}^{K-1} \frac{n}{4} - 2c^2k > \frac{Kn}{4} - c^2K^2 \ .$$

Setting $K = n/(8c^2) < m$, we have established that the total number of changes is $\Omega(n^2/c^2)$. □

Due to Lemma 4.3.10 and the fact that we process a total number of $\mathcal{O}(n^2 \log^2(nR))$ events, our KDS has an efficiency value of $\mathcal{O}(\log^2(nR))$. Hence, the KDS for the mobile facility location problem is efficient.

We summarize our results in the following theorem:

Theorem 4. *Let P be a set of n independently moving points in $\mathbb{R}^d$, where d is a constant dimension. Then, there exists a deterministic KDS for the mobile facility location problem that maintains at any point of time t a set $F(t) \subseteq P(t)$ such that we have*

$$\mathrm{FacLoc}(P(t), F(t)) < (64d + 1) \cdot \mathrm{FacLoc}(P(t), F^*(t)) \ .$$

Let $R = \max_{p_i \in P} f_i \cdot \max_{p_i \in P} d_i / (\min_{p_i \in P} f_i \cdot \min_{p_i \in P} d_i)$, where f_i and d_i are the opening cost and the demand of a point p_i, respectively. Then, the KDS has a space requirement of $\mathcal{O}(n(\log^d(n) + \log(nR)))$ and each event requires $\mathcal{O}(\log(nR))$ status changes and $\mathcal{O}(\log^{d+1}(n) \cdot \log(nR))$ update time. In case that the trajectories can be described by bounded-degree polynomials, the total number of updates is $\mathcal{O}(n^2 \log^2(nR))$, which results in a total processing time of $\mathcal{O}(n^2 \log^{d+1}(n) \cdot \log^3(nR))$. A flight plan update involves $\mathcal{O}(\log(nR))$ certificates and requires $\mathcal{O}(\log^2(nR))$ time.

5 Facility Location in Data Streams

This chapter deals with a constant-factor approximation algorithm for the cost of the uniform facility location problem over dynamic geometric data streams in a discrete Euclidean space $\{1, \ldots, \Delta\}^d$, where d is a constant. The starting point of our algorithm is the work of Indyk [64]. It gives the best previous approach for approximating the cost of the uniform facility location problem over dynamic geometric data streams and guarantees an approximation factor of $\mathcal{O}(\log^2(\Delta))$. In [64], Indyk defines a certain partition of the space into nested square grids and a set of cells in this partition such that the number of these cells gives an $\mathcal{O}(\log(\Delta))$-approximation. During the approximation process to estimate the number of these cells, the algorithm of [64] looses another $\mathcal{O}(\log(\Delta))$ factor.

In Section 5.1, we use a similar partition of the space into nested square grids, and we show that opening a facility in each cell of a subset of the cells defined in [64] leads to a constant-factor approximation of the facility location cost. Moreover, in Section 5.3, we propose an algorithm that maintains this cost sufficiently well in the dynamic geometric data stream model. In this way, we obtain a streaming algorithm for approximating the cost of the uniform facility location problem that strongly improves the best previous one.

5.1 Definition of a Good Estimator

Let $P := \{p_1, \ldots, p_n\}$ be a set of n points from a discrete Euclidean space $\{1, \ldots, \Delta\}^d$, where d is a constant. In the streaming context, P will refer to the current point set, i.e., the set of points obtained after having applied an input sequence of insertions and deletions.

In this section, we will define a good estimator for the uniform Euclidean facility location problem (see Section 2.2 for a definition). Before we derive our estimator for the general case, we show how to deal with some special cases.

5.1.1 Estimator for Special Cases

We consider the following four special cases:

(i) The point set P is empty.

(ii) The point set P is non-empty and contains $\mathcal{O}(\lceil f/\Delta \rceil)$ points.

(iii) The opening cost f is at most 1.

(iv) The opening cost f is at least Δ^{d+1}.

In Case (i), there are no points that have to be served by an open facility. Hence, the facility location cost is obviously 0. Thus, our estimator is 0.

We distinguish two subcases of Case (ii), namely $f/\Delta < 1$ and $f/\Delta \geq 1$. In the first subcase, we have $|P| \geq 1$ and $|P| \in \mathcal{O}(1)$. Thus, there exists at least one open facility in an optimal solution, so the optimal facility location cost is at least f. If each point opens a facility, then the facility location cost are $f \cdot |P| \in \mathcal{O}(f)$. Hence, the optimal facility location cost is $\Theta(f)$. In the second subcase, we have $|P| \geq 1$ and $|P| \in \mathcal{O}(f/\Delta)$. Again, there exists at least one open facility in an optimal solution, so the optimal facility location cost is at least f. Furthermore, the total connection cost of the points in P is $\mathcal{O}(f)$ since the longest pairwise distance in P is upper bounded by $\sqrt{d} \cdot \Delta$ and there are $\mathcal{O}(f/\Delta)$ points in P. Thus, if we open one facility and connect the remaining points in P to this facility, then the resulting facility location cost is $\mathcal{O}(f)$. It follows that the optimal facility location cost is again $\Theta(f)$. Hence, in both subcases of Case (ii), we set our estimator to f.

In Case (iii), it is optimal to open a facility at each point in P since the opening cost f is at most as big as the minimum pairwise distance in P. Hence, our estimator is $f \cdot |P|$.

Case (iv) is similar to Case (ii). We can assume that P is not empty because otherwise we have Case (i). It follows that there has to be at least one open facility in an optimal solution. Thus, the optimal facility location cost is at least f. Furthermore, since the maximum pairwise distance of the points in P is at most $\sqrt{d} \cdot \Delta$ and there are at most Δ^d points in P, the cost to connect all the points in P to the same facility is $\mathcal{O}(\Delta^{d+1})$. Thus, the optimal facility location cost is $\Theta(f)$, so we can always safely output f as constant-factor approximation of the optimal facility location cost.

Distinct Elements Data Structure

To be able to transfer the computation of our estimators to the dynamic geometric data stream model, we have to be able to compute a good estimator for the size of P. To obtain this estimator, we use the data structure for counting the number of distinct elements in a data stream, under insertions and deletions, that has been proposed by Kane et al. [72]. This data structure has the following properties:

Lemma 5.1.1 ([72])**.** *Let ε, $0 < \varepsilon < 1$, be a precision parameter. There is a data structure that computes a $(1 \pm \varepsilon)$-approximation of the number of distinct elements in a data stream under insertions and deletions with probability at least $2/3$. The space requirement of the data structure is upper bounded by $\mathcal{O}(1/\varepsilon^2 \cdot \log(N) \cdot (\log(1/\varepsilon) + \log(\log(M))))$ bits, where N is the size of the domain of the elements and M is the multiplicity of single elements. The update time of an element is $\mathcal{O}(1)$.*

Corollary 5.1.2. *Let ε, $0 < \varepsilon < 1$, be a precision parameter, and let δ, $0 < \delta < 1$, be an error probability parameter. There is a data structure that computes a $(1 \pm \varepsilon)$-approximation of the number of distinct elements in a data stream under insertions and deletions with probability at least $1 - \delta$. The data structure has a space requirement of $\mathcal{O}(1/\varepsilon^2 \cdot \log(N) \cdot (\log(1/\varepsilon) + \log(\log(M))) \cdot \log(1/\delta))$ bits, where N is the size of the domain*

of the elements and M is the multiplicity of single elements. The update time of an element is $\mathcal{O}(\log(1/\delta))$.

Proof. The data structure from Lemma 5.1.1 outputs a $(1\pm\varepsilon)$-approximation of the number of distinct elements in a data stream under insertions and deletions with an error probability of at most $1/3$. This error probability can be reduced by using a standard amplification technique. More precisely, we run $\lceil 75\ln(1/\delta)\rceil$ copies of the algorithm in parallel and output their median value. For each $j \in \{1,\ldots,\lceil 75\ln(1/\delta)\rceil\}$, let Z_j be the indicator random variable for the event that the j-th run of the algorithm outputs a $(1\pm\varepsilon)$-approximation of the number of distinct elements. By a Chernoff bound, we get

$$\mathbf{Pr}\left[\sum_{j=1}^{\lceil 75\ln(1/\delta)\rceil} Z_j \leq \left(1-\frac{1}{5}\right)\cdot\mathbf{E}\left[\sum_{j=1}^{\lceil 75\ln(1/\delta)\rceil} Z_j\right]\right] \leq \exp\left(-\frac{1}{2\cdot 5^2}\cdot\mathbf{E}\left[\sum_{j=1}^{\lceil 75\ln(1/\delta)\rceil} Z_j\right]\right) \leq \delta \ .$$

Thus, the probability that more than a fraction of $8/15$-th of the copies computes a $(1\pm\varepsilon)$-approximation is at least $1-\delta$. This implies that the median value of the copies is a $(1\pm\varepsilon)$-approximation with probability at least $1-\delta$. Now, the assertion follows from Lemma 5.1.1. □

Given a stream of insert and delete operations of points from a discrete Euclidean space $\{1,\ldots,\Delta\}^d$ and an error probability parameter δ, we apply the data structure from Corollary 5.1.2 with precision parameter $\varepsilon := 1/2$. Since we assume that the input stream is consistent, i.e., no point is removed which is not present in the current point set and no point is added twice, the multiplicity M is constant. Furthermore, the size N of the domain is Δ^d. Thus, with probability at least $1-\delta$, we can compute a constant-factor approximation of $|P|$ and, hence, a constant-factor approximation of the facility location cost in the four special cases using $\mathcal{O}(d\cdot\log(\Delta)\cdot\log(1/\delta))$ space. An insertion or deletion of a point requires $\mathcal{O}(\log(1/\delta))$ time.

5.1.2 Estimator Based on a Space Partition

In the remainder of this chapter, we will always assume that the size of P is $\Omega(f/\Delta)$ and $1 < f < \Delta^{d+1}$. Furthermore, we assume that the value f is a power of 2. Note that, by rounding the opening cost up to the next power of 2, the facility location cost and also our estimator is increased by a factor of at most 2.

In order to deal with the general case, we define a certain partition of the input space and relate this partition to the cost for the uniform Euclidean facility location problem. In particular, if we assign to each cell in this partition a weight that corresponds to the number of points inside the cell multiplied by the side length of the cell, the sum of these weights is a constant-factor approximation of the cost for the uniform facility location problem. We will use this fact in Section 5.3 to develop an approximation algorithm in the dynamic geometric data stream model.

To compute the above mentioned space partition for P, we impose $\lceil\log(\Delta)\rceil+1$ nested square grids over the point space denoted by $\mathcal{G}(0), \mathcal{G}(1), \ldots, \mathcal{G}(\lceil\log(\Delta)\rceil)$. The side length

of each cell in grid $\mathcal{G}(i)$ is 2^i. We say that the grid cells in $\mathcal{G}(i)$ are in level i. The set of *neighbors* $\Gamma(\mathcal{C})$ of a cell $\mathcal{C} \in \mathcal{G}(i)$ is the set of all the cells in grid $\mathcal{G}(i)$ that share some part of their boundary with $\mathcal{C}$. Note that all the cells located at the border of some grid have less than $3^d - 1$ neighbors. For example, there is only one cell in grid $\mathcal{G}(\lceil \log(\Delta) \rceil)$, and this cell has no neighbors. All remaining cells have exactly $3^d - 1$ neighbors. Furthermore, we need the definition of a *parent cell* and a *subcell*. The parent cell of a cell $\mathcal{C} \in \mathcal{G}(i)$ in any level $i \in [\lceil \log(\Delta) \rceil]$ is the cell in $\mathcal{G}(i+1)$ that contains $\mathcal{C}$. The subcells of a cell $\mathcal{C} \in \mathcal{G}(i)$ in any level $i \in \{1, 2, \ldots, \lceil \log(\Delta) \rceil\}$ are all the cells in $\mathcal{G}(i-1)$ that are contained in $\mathcal{C}$.

In each grid $\mathcal{G}(i)$, the *active* and *maximal-useful* cells will play a decisive role in the space partitioning. They are defined as follows:

Definition 5.1.3 (Active Cell)**.** *A cell in any level $i \in [\lceil \log(\Delta) \rceil + 1]$ is called* active *if it contains at least $a(i) := f/2^i$ points of P. A grid cell that is not active is* inactive*.*

Observe that if a cell $\mathcal{C}$ is active, then all the cells that contain $\mathcal{C}$ are active as well.

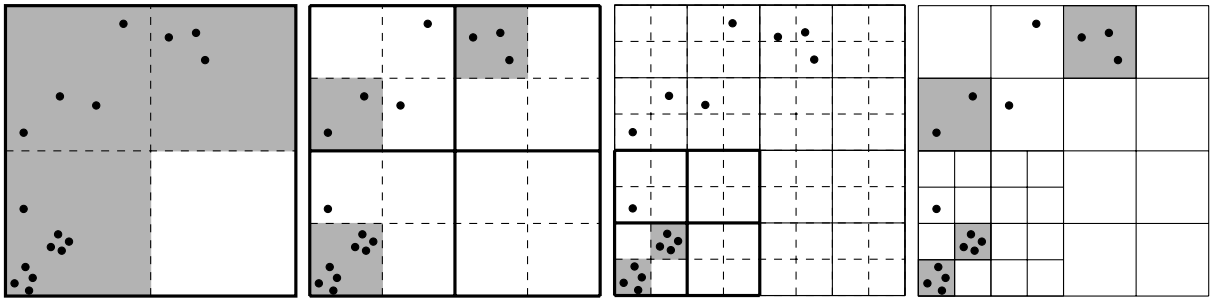
Definition 5.1.4 (Useful and Maximal-Useful Cell)**.** *A cell $\mathcal{C}$ in any level $i \in [\lceil \log(\Delta) \rceil + 1]$ is called* useful *if it neither contains an active subcell nor any of its neighbors $\Gamma(\mathcal{C})$ in grid $\mathcal{G}(i)$ contains an active subcell. A grid cell that is not useful is* useless*.*

A cell in any level $i \in [\lceil \log(\Delta) \rceil]$ is maximal-useful *if it is useful but its parent cell is useless. The cell in level $\mathcal{G}(\lceil \log(\Delta) \rceil)$ is maximal-useful if it is useful.*

Our space partition consists of all maximal-useful cells. Let $\mathcal{SP}(i)$ be the set of all maximal-useful cells in grid $\mathcal{G}(i)$, and let $\mathcal{SP} := \bigcup_i \mathcal{SP}(i)$ be the set of all maximal-useful cells. The cells in $\mathcal{SP}$ form a partition of the input space. This follows from the fact that we can simply construct $\mathcal{SP}$ in a process similar to that of building a quadtree. In general, a quadtree for a d-dimensional point set is a rooted tree in which every node corresponds to a squared cell. Each internal node v has 2^d children whose corresponding cells build a partition of v. Hence, the cells corresponding to the leaf nodes of the quadtree form a partition of the space, which is called a *quadtree partition*. Following this definition, we can construct our space partition by starting from the cell in the coarsest grid $\mathcal{G}(\lceil \log(\Delta) \rceil)$ and recursively splitting each useless cell into 2^d equal sized, squared subcells. The final space partition consists of only useful cells whose parent cells are useless. Hence, we obtain $\mathcal{SP}$ as desired. An illustration of a space partitioning is given in Figure 5.1.

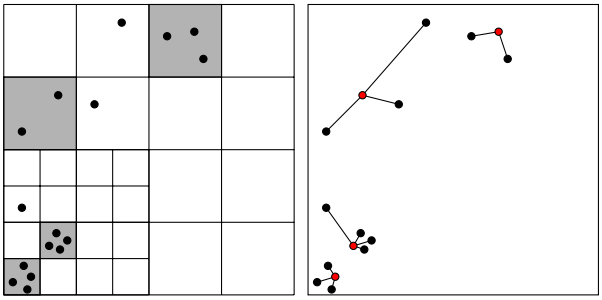
The key idea is now to place an open facility in each active cell in $\mathcal{SP}$. Figure 5.2 illustrates how this is related to a solution for the uniform facility location problem. We remark that our strategy of choosing the set of open facilities is a refinement of the strategy proposed in [64]. More precisely, the open facilities in [64] are chosen from all active cells in $\bigcup_{i=0}^{\lceil \log(\Delta) \rceil} \mathcal{G}(i)$, whereas we choose the open facilities from a subset of these cells.

Next, we define a value $\mathrm{FL}(P, f)$ that is based on the space partition $\mathcal{SP}$ and yields a constant-factor approximation of the cost of an optimal solution for the uniform Euclidean



(a) (b) (c) (d)

Figure 5.1: Example illustrating the quadtree partition for a set of points from $\{1, \ldots, 128\}^2$ and for the opening-cost value $f = 64$. Active cells are colored in gray. Useless cells are indicated by thick borders. Subcells of a cell are indicated by dashed borders. (a)-(c) The quadtree partition for subsequent depths of the recursion. (d) The final quadtree partition and its active cells.



(a) (b)

Figure 5.2: (a) The final quadtree partition for a set of points from $\{1, \ldots, 128\}^2$ and for the opening-cost value $f = 64$. Active cells are colored in gray. (b) Solution for the uniform facility location problem whose cost is approximated by the algorithm. The red points are the open facilities. Connections between points are indicated by line segments.

facility location problem. Let $n_P(\mathcal{C})$ be the number of points in the set P that are contained in the cell $\mathcal{C}$. Then, the estimator for the facility location cost is defined as

$$\mathrm{FL}(P, f) := \sum_{i=0}^{\lceil \log(\Delta) \rceil} \sum_{\mathcal{C} \in \mathcal{SP}(i)} n_P(\mathcal{C}) \cdot 2^i \ . \tag{5.1}$$

5.1.3 Properties of the Space Partition

Before we prove that $\mathrm{FL}(P, f)$ is indeed an $\mathcal{O}(1)$-approximation of the cost of the uniform Euclidean facility location problem, we discuss some properties of the space partition that are needed in the analysis. We say that two cells in a space partition are neighbors if they

share at least one point of their boundary. Furthermore, the distance between two cells is defined as the minimum distance between two points such that one point lies on the boundary of one cell and the other point lies on the boundary of the other cell. Now, we show that the space partition $\mathcal{SP}$ has the following properties:

Lemma 5.1.5. *The set $\mathcal{SP}$ of all maximal-useful cells has the following five properties:*

(*i*) *The side length of each cell in $\mathcal{SP}$ differs from the side length of each of its neighbors by a factor of at most 2, i.e., the space partition is balanced.*

(*ii*) *Let $i \in [\lceil \log(\Delta) \rceil + 1]$ be any level, and let $\mathcal{C}$ be any useless cell in $\mathcal{G}(i)$. Then, there exists an active cell with side length at most 2^{i-1} in $\mathcal{SP}$ that has a distance of at most $\sqrt{d} \cdot 2^{i+1}$ from $\mathcal{C}$.*

(*iii*) *Let $i \in [\lceil \log(\Delta) \rceil + 1]$ be any level, and let $\mathcal{C}$ be any inactive cell in $\mathcal{SP}(i)$. Then, there exists an active cell with side length at most 2^i in $\mathcal{SP}$ that has a distance of at most $5\sqrt{d} \cdot 2^i$ from $\mathcal{C}$.*

(*iv*) *Let $i \in [\lceil \log(\Delta) \rceil + 1]$ be any level, and let $\mathcal{C}$ be any active cell in $\mathcal{SP}(i)$. Then, we have*

$$\frac{f}{2^i} \leq n_P(\mathcal{C}) < 2^{d+1} \cdot \frac{f}{2^i} \ .$$

(*v*) *Let $i \in [\lceil \log(\Delta) \rceil + 1]$ be any level. Then, we have $0 < a(i) < \Delta^{d+1}$, and we have either $a(i) \geq 1$ or $\mathcal{SP}(i)$ contains no non-empty cell.*

Proof.

(*i*) Obviously, there cannot be a cell $\mathcal{C} \in \mathcal{SP}(0) \cup \mathcal{SP}(1)$ that has a neighbor in $\mathcal{SP}$ whose side length is less than half the side length of $\mathcal{C}$. We prove the assertion for any level $i \in \{2, \ldots, \lceil \log(\Delta) \rceil\}$ by contradiction. Assume that $\mathcal{C}_{\text{big}}$ is a cell from $\mathcal{SP}(i)$ that has a neighbor cell $\mathcal{C}_{\text{small}}$ in $\mathcal{SP}(j)$, $j \leq i-2$, i.e., $\mathcal{C}_{\text{small}}$ is a neighbor cell with side length $2^j \leq 2^{i-2}$. This situation is illustrated in Figure 5.3. Let $\mathcal{C}'_{\text{small}}$ be the parent cell of $\mathcal{C}_{\text{small}}$. Since $\mathcal{C}_{\text{small}}$ is maximal-useful, its parent $\mathcal{C}'_{\text{small}}$ is useless. Hence, $\mathcal{C}'_{\text{small}}$ or at least one neighbor in $\Gamma(\mathcal{C}'_{\text{small}})$ has an active subcell (the light gray area in Figure 5.3). This subcell is either contained in $\mathcal{C}_{\text{big}}$ or one of its neighbors $\Gamma(\mathcal{C}_{\text{big}})$. Hence, $\mathcal{C}_{\text{big}}$ is also a useless cell and cannot be a cell in $\mathcal{SP}(i)$, which is a contradiction.

(*ii*) The cells in level 0 are all useful per definition since they contain no subcells. To prove the assertion for the remaining levels, we proceed by induction. Let ℓ be the smallest level such that $\mathcal{SP}(\ell)$ is not empty. Let $\mathcal{C}$ be a useless cell in grid $\mathcal{G}(\ell+1)$. Since $\mathcal{C}$ is useless, either $\mathcal{C}$ or one of its neighbors in $\Gamma(\mathcal{C})$ contains an active subcell $\mathcal{A}$. By the choice of ℓ, we know that $\mathcal{A}$ is maximal-useful and in $\mathcal{SP}$. Furthermore, $\mathcal{A}$ has a side length of 2^ℓ and a distance of at most $\sqrt{d} \cdot 2^\ell$ from $\mathcal{C}$, which is less than $\sqrt{d} \cdot 2^{\ell+1}$. This proves the base case. Now, let $\mathcal{C}$ be a useless cell in grid $\mathcal{G}(i)$. By

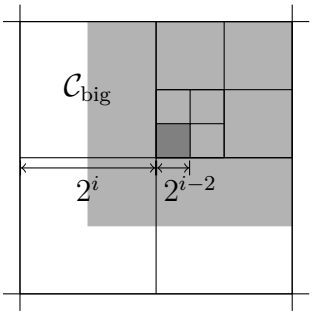


Figure 5.3: Arrangement of cells that leads to the desired contradiction in the proof of the first property stated in Lemma 5.1.5. The cell $\mathcal{C}_{\text{small}}$ is indicated by the dark gray square. The area containing an active subcell is colored in light gray.

definition, either $\mathcal{C}$ or one of its neighbors in $\Gamma(\mathcal{C})$ contains an active subcell. Let $\mathcal{A}$ be such a subcell. The cell $\mathcal{A}$ has side length 2^{i-1} and a distance of at most $\sqrt{d} \cdot 2^{i-1}$ from $\mathcal{C}$. If $\mathcal{A}$ is useful, it is maximal-useful and in $\mathcal{SP}$, so we are done. Otherwise, $\mathcal{A}$ is useless and in grid $\mathcal{G}(j)$, $j < i$. By induction hypothesis, we have an active cell $\mathcal{A}'$ with side length at most 2^{j-1} in $\mathcal{SP}$ which has a distance of at most $\sqrt{d} \cdot 2^{j+1} \leq \sqrt{d} \cdot 2^i$ from $\mathcal{A}$. Since $\mathcal{A}$ has a diagonal of length $\sqrt{d} \cdot 2^{i-1}$, we get that the distance from $\mathcal{C}$ to $\mathcal{A}'$ is at most $2 \cdot \sqrt{d} \cdot 2^{i-1} + \sqrt{d} \cdot 2^i = \sqrt{d} \cdot 2^{i+1}$.

(*iii*) Since we assume that $|P| \in \Omega(f/\Delta)$, the cell in $\mathcal{G}(\lceil \log(\Delta) \rceil)$ is active. According to this, let $\mathcal{C} \in \mathcal{SP}$ be an inactive cell in a level $i \in [\lceil \log(\Delta) \rceil]$. Let $\mathcal{C}'$ be the parent cell of $\mathcal{C}$. By ii), there is an active cell with side length at most 2^i in $\mathcal{SP}$ that has a distance of at most $\sqrt{d} \cdot 2^{i+2}$ from $\mathcal{C}'$. Hence, the distance from $\mathcal{C}$ is at most $\sqrt{d} \cdot 2^i + \sqrt{d} \cdot 2^{i+2} = 5\sqrt{d} \cdot 2^i$.

(*iv*) The first inequality of the assertion follows from our definition of an active cell. Since each cell in level 0 contains at most 2^d points from P and $f > 1$, the second inequality is satisfied for each cell in $\mathcal{SP}(0)$. Let $i \in \{1, \ldots, \lceil \log(\Delta) \rceil\}$ be any level and $\mathcal{C}$ be any cell in $\mathcal{SP}(i)$. The number of points in $\mathcal{C}$ is less than $2^{d+1} \cdot f/2^i$ because each of the 2^d subcells of $\mathcal{C}$ is inactive, i.e., there are less than $f/2^{i-1}$ points inside such a subcell.

(*v*) Recall that $a(i) = f/2^i$. Since $f > 1$, we have $a(i) > 0$. Furthermore, it follows from $f < \Delta^{d+1}$ and $i \geq 0$ that $a(i) = f/2^i < \Delta^{d+1}$.

Obviously, we get $a(0) = f/2^0 > 1$ since $f > 1$. Thus, in case $i = 0$, we always have $a(i) \geq 1$. For any level $i \in \{1, \ldots, \lceil \log(\Delta) \rceil\}$, the proof is by contradiction. Let us assume that there is a non-empty cell $\mathcal{C} \in \mathcal{SP}(i)$ with $a(i) \leq 1/2$. Then, we have $a(i-1) \leq 1$, so $\mathcal{C}$ contains an active subcell, which is a contradiction to the construction of the space partition $\mathcal{SP}$. Hence, we have $a(i) > 1/2$. In addition, since $a(i) = f/2^i > 1/2$ and we assume that f is a power of 2, we have $a(i) \geq 1$.

□

5.1.4 Analysis of the Estimator

In this section, we analyze our estimator $\mathrm{FL}(P, f)$. We separate the analysis into two parts. We give an appropriate lower bound in the first part and an appropriate upper bound in the second part. For this purpose, let $\mathrm{FacLoc}^*(P, f)$ be the cost of an optimal facility location solution for P.

Lemma 5.1.6. $\mathrm{FL}(P, f) \in \Omega(\mathrm{FacLoc}^*(P, f))$.

Proof. Our goal is to define a set of open facilities such that the induced facility location cost is $\mathcal{O}(\mathrm{FL}(P, f))$. This proves $\mathrm{FL}(P, f) \in \Omega(\mathrm{FacLoc}^*(P, f))$. We will show that it suffices to open one facility in each active cell in $\mathcal{SP}$.

We give an upper bound on the contribution of the points in each cell in $\mathcal{SP}$. For any level $i \in [\lceil \log(\Delta) \rceil + 1]$, each active cell $\mathcal{C} \in \mathcal{SP}(i)$ contributes at most $f + n_P(\mathcal{C}) \cdot \sqrt{d} \cdot 2^i$ because we open one facility in $\mathcal{C}$ and connect the points inside of $\mathcal{C}$ to this facility. Since $\mathcal{C}$ is active, it contains at least $f/2^i$ points. Thus, we have $f + n_P(\mathcal{C}) \cdot \sqrt{d} \cdot 2^i \in \mathcal{O}(n_P(\mathcal{C}) \cdot 2^i)$. The points in each inactive cell $\mathcal{C}$ in $\mathcal{SP}$ are connected to the nearest open facility. Due to Property (*iii*) in Lemma 5.1.5, for each inactive cell $\mathcal{C} \in \mathcal{SP}(i)$, there exists an active cell with side length at most 2^i in $\mathcal{SP}$ which has a distance of at most $5\sqrt{d} \cdot 2^i$ from $\mathcal{C}$. Thus, the connection cost for the points in $\mathcal{C}$ is at most $n_P(\mathcal{C}) \cdot (5\sqrt{d} \cdot 2^i + \sqrt{d} \cdot 2^i) \in \mathcal{O}(n_P(\mathcal{C}) \cdot 2^i)$. Summing up over all cells in $\mathcal{SP}$ gives that the cost of the defined solution is $\mathcal{O}(\mathrm{FL}(P, f))$. □

Lemma 5.1.7. $\mathrm{FL}(P, f) \in \mathcal{O}(\mathrm{FacLoc}^*(P, f))$.

Proof. Let F^* be a set of optimal open facilities. Since we assume that P is not empty, the set F^* is not empty and we have $\mathrm{FacLoc}^*(P, f) \in \Omega(f)$. Now, for any level $i \in [\lceil \log(\Delta) \rceil + 1]$, we partition the set $\mathcal{SP}(i)$ into two subsets $\mathcal{SP}_{\mathrm{near}}(i)$ and $\mathcal{SP}_{\mathrm{dist}}(i)$. The set $\mathcal{SP}_{\mathrm{near}}(i)$ contains every cell whose distance to its nearest open facility in F^* is less than 2^{i-1}, i.e.,

$$\mathcal{SP}_{\mathrm{near}}(i) := \{\mathcal{C} \in \mathcal{SP}(i) \mid \min_{q \in F^*} \mathrm{D}(q, \mathcal{C}) < 2^{i-1}\} \ .$$

The set $\mathcal{SP}_{\mathrm{dist}}(i)$ contains all remaining cells from $\mathcal{SP}(i)$, i.e.,

$$\mathcal{SP}_{\mathrm{dist}}(i) := \{\mathcal{C} \in \mathcal{SP}(i) \mid \min_{q \in F^*} \mathrm{D}(q, \mathcal{C}) \geq 2^{i-1}\} \ .$$

For each cell $\mathcal{C} \in \bigcup_{i=0}^{\lceil \log(\Delta) \rceil} \mathcal{SP}_{\mathrm{dist}}(i)$, the cost to connect the points inside of $\mathcal{C}$ to the nearest open facility in F^* is at least $n_P(\mathcal{C}) \cdot 2^{i-1}$. This is exactly half of the cost that we charge for the cell $\mathcal{C}$ by the definition of $\mathrm{FL}(P, f)$. Thus, the cost that we charge for points contained in $\bigcup_{i=0}^{\lceil \log(\Delta) \rceil} \mathcal{SP}_{\mathrm{dist}}(i)$ is upper bounded by twice the optimal connection cost.

Let $\mathcal{C} \in \mathcal{SP}(j)$ be a cell in any level $j \in [\lceil \log(\Delta) \rceil + 1]$ that contains an optimal facility $q \in F^*$. Furthermore, let $\mathcal{C}' \in \mathcal{SP}(i)$ be any cell in any level $i \in [\lceil \log(\Delta) \rceil + 1]$ such that $\mathcal{C}'$ is not a direct neighbor of $\mathcal{C}$. Due to Property (*i*) in Lemma 5.1.5, $\mathcal{SP}$ is a balanced space partition, so the neighbors of $\mathcal{C}'$ have a side length of at least 2^{i-1}. It follows that there is at least one cell with side length at least 2^{i-1} between $\mathcal{C}'$ and $\mathcal{C}$. Thus, $\mathcal{C}'$ is not within

distance of less than 2^{i-1} from q. Hence, we have $\mathcal{C}' \notin \mathcal{SP}_{\text{near}}(i)$. This implies that only direct neighbors of $\mathcal{C}$ can be in $\bigcup_{i=0}^{\lceil \log(\Delta) \rceil} \mathcal{SP}_{\text{near}}(i)$. Due to the fact that $\mathcal{SP}$ is a balanced space partition, the neighbors of $\mathcal{C}$ have a side length of at least 2^{j-1}. Thus, the number of neighbors of $\mathcal{C}$ is at most $4^d - 2^d$. It follows that less than 4^d cells in $\bigcup_{i=0}^{\lceil \log(\Delta) \rceil} \mathcal{SP}_{\text{near}}(i)$ are within distance less than half of their side length from q. Hence, we have

$$\sum_{i=0}^{\lceil \log(\Delta) \rceil} |\mathcal{SP}_{\text{near}}(i)| < 4^d \cdot |F^*| \ .$$

Now, for each cell in $\mathcal{SP}$, we charge a cost of $\mathcal{O}(f)$ by the definition of $\text{FL}(P, f)$. This is due to Property (iv) in Lemma 5.1.5, which implies that a cell in $\mathcal{SP}(i)$ contains at most $2^{d+1} \cdot f/2^i$ points. Thus, the cost that arises for all cells in $\bigcup_{i=0}^{\lceil \log(\Delta) \rceil} \mathcal{SP}_{\text{near}}(i)$ is $\mathcal{O}(f \cdot |F^*|)$, which is at most a constant factor larger than the optimal opening cost. □

5.2 Randomized Algorithm

In this section, we describe a randomized algorithm that implements the ideas of Section 5.1 and, with some modifications, can be transformed into a streaming algorithm, which we will do in Section 5.3.

The approach of the algorithm is closely related to performing the quadtree partition into maximal-useful cells. We try to identify all active cells in the grids. For that purpose, for each level $i \in [\lceil \log(\Delta) \rceil + 1]$, we maintain one random sample set and take each point into this set with probability $\alpha(i) := \min\{1/a(i), 1\}$. Recall that a cell in grid $\mathcal{G}(i)$ is active if it contains at least $a(i) = f/2^i$ points. Thus, in expectation, we will see at least one point in every active cell of grid $\mathcal{G}(i)$. Observe that some sample points will also end up in inactive cells. However, we will show in the analysis that this does not negatively affect our algorithm. We call a cell in grid $\mathcal{G}(i)$ *marked* if it contains at least one sample point. The key idea is to go through all levels $i \in [\lceil \log(\Delta) \rceil + 1]$ and to open one facility in every marked cell $\mathcal{C}$ in grid $\mathcal{G}(i)$ such that the following two conditions are satisfied:

(a) No subcell of $\mathcal{C}$ is marked.

(b) No smaller cell within a distance of less than 2^{i-1} from $\mathcal{C}$ is marked.

The motivation of Condition (b) is that, in our space partition $\mathcal{SP}$, the side lengths of neighbor cells differ at most by a factor of 2. Hence, a marked cell from $\mathcal{SP}$ prevents at most a constant number of other cells from $\mathcal{SP}$ to open a facility.

Finally, we obtain a new estimator for the cost of the uniform facility location problem based on our randomized algorithm. Let $\mathcal{F}$ denote the set of cells, where we open a facility. Then, the estimator is $\text{FL}_{\text{rand}}(P, f) := f \cdot |\mathcal{F}|$.

5.2.1 Random Sampling

In each level $i \in [\lceil \log(\Delta) \rceil + 1]$, we would like to sample each point from P independently at random with probability $\alpha(i) = \min\{1/a(i), 1\}$. Since we have insert as well as delete operations of points, the random experiments for the points must be reproducible. Therefore, for each level $i \in [\lceil \log(\Delta) \rceil + 1]$, we use a function $h_i : \{1, \ldots, \Delta\}^d \to \{0, 1\}$ that maps a point to the value 1 with probability $\alpha(i) = \min\{1/a(i), 1\}$. For any point $p \in P \subseteq \{1, \ldots, \Delta\}^d$, if $h_i(p) = 1$, then p is a sample point. Otherwise, p is not a sample point. We can construct a function $h_i(\cdot)$ with the following properties:

Lemma 5.2.1. *For each level $i \in [\lceil \log(\Delta) \rceil + 1]$, there is a function $h_i : \{1, \ldots, \Delta\}^d \to \{0, 1\}$ which maps each point $p \in \{1, \ldots, \Delta\}^d$ independently at random to a value in $\{0, 1\}$ such that*

$$\mathbf{Pr}\left[h_i(p) = 1\right] = \min\{1/a(i), 1\} \ .$$

The function $h_i(\cdot)$ uses $\mathcal{O}(\Delta^d \cdot \log(\Delta))$ random bits. For each point $p \in \{1, \ldots, \Delta\}^d$, the value of $h_i(p)$ can be computed in $\mathcal{O}(\log(\Delta))$ time.

Proof. Let $i \in [\lceil \log(\Delta) \rceil + 1]$ be any fixed level. In case that $\alpha(i) = 1$, the assertion of the lemma is obviously true. Hence, in the following, we will assume that $\alpha(i) < 1$ and, thus, $a(i) > 1$.

Since $i \geq 0$ and $f < \Delta^{d+1}$, we have $a(i) = f/2^i < \Delta^{d+1}$. In addition, since f is a power of 2 with positive exponent and $a(i) > 1$, the value $a(i)$ is also a power of 2 with positive exponent. Observe that $a(i)$ can be represented by $\ell := \lceil (d+1) \log(\Delta) \rceil$ bits.

Now, for each point $p \in \{1, \ldots, \Delta\}^d$, we generate a bit vector of length ℓ, where each bit is chosen independently and uniformly at random. Let

$$r_i(p) := \left(r_i^{(1)}, r_i^{(2)}, \ldots, r_i^{(\ell)}\right)$$

be the generated bit sequence for p. The function h_i maps p to 1 if $r_i^{(j)} = 0$ for $j < \log(a(i))$ and $r_i^{(\log(a(i)))} = 1$. For any $k \in \{1, \ldots, \ell\}$, the event that $r_i^{(j)} = 0$ for $j < k$ and $r_i^{(k)} = 1$ happens with probability 2^{-k}. Thus, the probability that h_i maps the point p to 1 is

$$\mathbf{Pr}\left[h_i(p) = 1\right] = 2^{-\log(a(i))} = \frac{1}{a(i)} = \alpha(i) \ .$$

Since we generate ℓ random bits for each point in $\{1, \ldots, \Delta\}^d$, the function $h_i(\cdot)$ uses $\ell \cdot \Delta^d \in \mathcal{O}(\Delta^d \cdot \log(\Delta))$ random bits in total. To compute $h_i(p)$ for any $p \in \{1, \ldots, \Delta\}^d$, we have at most one read and compare operation for each bit in $r_i(p)$. Thus, the time to compute $h_i(p)$ is $\mathcal{O}(\log(\Delta))$. □

The issue of full randomness will be discussed in Section 5.3.

5.2.2 Analysis of the Estimator

We will show that, with high constant probability, our randomized algorithm computes a facility location cost that is a constant-factor approximation of the estimator $\mathrm{FL}(P, f)$. For any level $i \in [\lceil \log(\Delta) \rceil + 1]$, let $\mathcal{F}(i)$ be the set of marked cells in $\mathcal{G}(i)$ that do not have a marked subcell and that do not have a smaller marked cell within a distance of less than 2^{i-1}. Then, the cells in the set $\bigcup_{i=0}^{\lceil \log(\Delta) \rceil} \mathcal{F}(i)$ are exactly the cells in which the algorithm opens its facilities, i.e., we have $\mathcal{F} = \bigcup_{i=0}^{\lceil \log(\Delta) \rceil} \mathcal{F}(i)$. Thus, the estimator of the randomized algorithm is given by

$$\mathrm{FL}_{\mathrm{rand}}(P, f) = f \cdot \sum_{i=0}^{\lceil \log(\Delta) \rceil} |\mathcal{F}(i)| \ . \tag{5.2}$$

Next, we derive appropriate lower and upper bounds of the estimator $\mathrm{FL}_{\mathrm{rand}}(P, f)$.

Lemma 5.2.2. $\mathrm{FL}_{\mathrm{rand}}(P, f) \in \Omega(\mathrm{FL}(P, f))$ *with probability at least* $15/16$.

Proof. Let us consider the space partition $\mathcal{SP}$ defined in Section 5.1. We are interested in the number of marked cells from $\mathcal{SP}$. However, f multiplied by the number of marked cells from $\mathcal{SP}$ does not immediately give a lower bound on $\mathrm{FL}_{\mathrm{rand}}(P, f)$. The reason is that, for any level $i \in [\lceil \log(\Delta) \rceil + 1]$, we do not open a facility in a marked cell in $\mathcal{SP}(i)$ if there is a smaller cell within a distance of less 2^{i-1} which is also marked. Since neighbor cells in $\mathcal{SP}$ differ by a factor of at most 2 in their side lengths, every marked cell in $\mathcal{SP}$ can prevent at most a constant number of other marked cells in $\mathcal{SP}$ from opening a facility. Thus, if we can show that the expected number of marked cells in $\mathcal{SP}$ is $\Omega(\mathrm{FL}(P, f)/f)$, then the assertion follows.

We say that a point $p \in P \cap \mathcal{SP}(i)$ is marked if it is sampled in level i. Let X_p denote the indicator random variable for the event that p is marked. Then, the expected number of marked points in any cell $\mathcal{C} \in \mathcal{SP}(i)$ is

$$\mathbf{E}\left[\sum_{p \in \mathcal{C}} X_p\right] = n_P(\mathcal{C}) \cdot \min\left\{\frac{1}{a(i)}, 1\right\} \ . \tag{5.3}$$

By the definition of $a(i)$, we obtain that

$$\mathbf{E}\left[\sum_{p \in \mathcal{C}} X_p\right] \leq \frac{n_P(\mathcal{C})}{a(i)} = \frac{n_P(\mathcal{C}) \cdot 2^i}{f} \ .$$

Due to Property (iv) in Lemma 5.1.5, for every cell $\mathcal{C} \in \mathcal{SP}$, we get

$$\mathbf{E}\left[\sum_{p \in \mathcal{C}} X_p\right] < 2^{d+1} \ .$$

Hence, we can group the cells from $\mathcal{SP}$ into sets $\mathcal{S}_1, \ldots, \mathcal{S}_\ell$ such that, for each set $\mathcal{S}_j$ with $1 \leq j < \ell$, we have

$$40 \leq \sum_{\mathcal{C} \in \mathcal{S}_j} \sum_{p \in \mathcal{C}} \mathbf{E}[X_p] < 40 + 2^{d+1} \tag{5.4}$$

and

$$\sum_{\mathcal{C}\in\mathcal{S}_\ell}\sum_{p\in\mathcal{C}}\mathbf{E}\left[X_p\right] < 40 + 2^{d+1} \tag{5.5}$$

for the set $\mathcal{S}_\ell$.

Next, we analyze the contribution of the sets $\mathcal{S}_1,\ldots,\mathcal{S}_\ell$ to the estimator $\mathrm{FL}(P,f)$. Due to Property (v) in Lemma 5.1.5, for any cell $\mathcal{C}\in\mathcal{SP}(i)$, we have either $a(i)\geq 1$ or $a(i)>0$ and $n_P(\mathcal{C})=0$. It follows from Equation (5.3) that

$$\mathbf{E}\left[\sum_{p\in\mathcal{C}}X_p\right] \geq \frac{n_P(\mathcal{C})}{a(i)} \ .$$

Due to Inequalities (5.4) and (5.5), the contribution of each $\mathcal{S}_j$, $1\leq j\leq\ell$, to the estimator $\mathrm{FL}(P,f)$ is

$$\begin{aligned}
\sum_{i=0}^{\lceil\log(\Delta)\rceil}\sum_{\mathcal{C}\in\mathcal{S}_j\cap\mathcal{SP}(i)} n_P(\mathcal{C})\cdot 2^i &\leq \sum_{i=0}^{\lceil\log(\Delta)\rceil}\sum_{\mathcal{C}\in\mathcal{S}_j\cap\mathcal{SP}(i)} 2^i\cdot a(i)\cdot\mathbf{E}\left[\sum_{p\in\mathcal{C}}X_p\right]\\
&= f\cdot\sum_{i=0}^{\lceil\log(\Delta)\rceil}\sum_{\mathcal{C}\in\mathcal{S}_j\cap\mathcal{SP}(i)}\mathbf{E}\left[\sum_{p\in\mathcal{C}}X_p\right]\\
&< f\cdot(40+2^{d+1})\\
&\in \mathcal{O}(f) \ .
\end{aligned}$$

Hence, we have $\mathrm{FL}(P,f)\in\mathcal{O}(f\ell)$. This means, the assertion of the lemma follows if the number of marked cells in $\mathcal{SP}$ is $\Omega(\ell)$. We consider the cases $\ell\leq 2$ and $\ell>2$.

First, we consider the case that $\ell>2$. We define the random variable

$$Y_j := \sum_{\mathcal{C}\in\mathcal{S}_j}\sum_{p\in\mathcal{C}}X_p \ .$$

By a Chernoff bound, we obtain

$$\mathbf{Pr}\left[Y_j\leq\left(1-\frac{1}{2}\right)\cdot\mathbf{E}\left[Y_j\right]\right]\leq\exp\left(-\frac{\mathbf{E}\left[Y_j\right]}{8}\right) \ .$$

This implies that $\mathbf{Pr}\left[Y_j\leq 20\right]\leq 1/e^5$ for $1\leq j<\ell$. Hence, with probability at least $1-1/e^5$, at least one of the cells in $\mathcal{S}_j$ is marked. For $1\leq j<\ell$, let Z_j denote the indicator random variable for the event that no cell in $\mathcal{S}_j$ is marked. The expected value of Z_j is at most $1/e^5$. By Markov's inequality, we get

$$\mathbf{Pr}\left[\sum_{j=1}^{\ell-1}Z_j\geq 32\cdot\mathbf{E}\left[\sum_{j=1}^{\ell-1}Z_j\right]\right]\leq\frac{1}{32} \ .$$

Thus, we have

$$\sum_{j=1}^{\ell-1}Z_j < 32\cdot\mathbf{E}\left[\sum_{j=1}^{\ell-1}Z_j\right] = 32\cdot\frac{\ell-1}{e^5}\leq\frac{\ell}{3}$$

with probability at least 31/32. It follows that, with probability at least 31/32, the number of marked cells in $\mathcal{SP}$ is at least $\ell - 1 - \ell/3 \in \Omega(\ell)$. Thus, we have $\mathrm{FL}_{\mathrm{rand}}(P, f) \in \Omega(f\ell)$, so $\mathrm{FL}_{\mathrm{rand}}(P, f) \in \Omega(\mathrm{FL}(P, f))$.

In case that $\ell \leq 2$, we have $\mathrm{FL}(P, f) \in \mathcal{O}(f)$. Furthermore, we can assume that P contains at least $32 \cdot \lceil f/\Delta \rceil$ points, otherwise we have one of the special cases considered in Section 5.1.1. It follows that the expected number of marked points in the cell $\mathcal{C}$ in grid $\mathcal{G}(\lceil \log(\Delta) \rceil)$ is at least

$$\begin{aligned} \mathbf{Pr}\left[\sum_{p \in \mathcal{C}} X_p\right] &\geq \min\left\{\frac{1}{a(\lceil \log(\Delta) \rceil)}, 1\right\} \cdot 32 \cdot \left\lceil \frac{f}{\Delta} \right\rceil \\ &= \min\left\{\frac{2^{\lceil \log(\Delta) \rceil}}{f}, 1\right\} \cdot 32 \cdot \left\lceil \frac{f}{\Delta} \right\rceil \\ &\geq 32 \end{aligned}$$

since $\lceil f/\Delta \rceil \geq 1$. By a Chernoff bound, we get

$$\mathbf{Pr}\left[\sum_{p \in \mathcal{C}} X_p \leq \left(1 - \frac{1}{2}\right) \cdot \mathbf{E}\left[\sum_{p \in \mathcal{C}} X_p\right]\right] \leq \exp\left(-\frac{\mathbf{E}\left[\sum_{p \in \mathcal{C}} X_p\right]}{8}\right) \leq \exp\left(-\frac{32}{8}\right) \leq \frac{1}{32} \ .$$

Thus, with probability at least 31/32, the cell $\mathcal{C}$ in $\mathcal{G}(\lceil \log(\Delta) \rceil)$ is marked. Due to our construction, we have $\mathcal{F} \geq 1$, so $\mathrm{FL}_{\mathrm{rand}}(P, f) \in \Omega(f)$. Hence, we get $\mathrm{FL}_{\mathrm{rand}}(P, f) \in \Omega(\mathrm{FL}(P, f))$, which completes the proof. □

To prove the upper bound, we first observe that every cell $\mathcal{C}$ is either contained in $\mathcal{SP}$ or it can be partitioned into cells from $\mathcal{SP}$ ($\mathcal{C}$ lies above $\mathcal{SP}$) or it is a subcell of a cell in $\mathcal{SP}$ ($\mathcal{C}$ lies below $\mathcal{SP}$). We will first show that the overall expected number of sample points from cells that lie below $\mathcal{SP}$ or that do not lie 'far above' $\mathcal{SP}$ is $\mathcal{O}(\mathrm{FL}(P, f)/f)$. Hence, the overall cost caused by these cells is $\mathcal{O}(\mathrm{FL}(P, f))$. Then, we prove that the expected contribution of cells 'far above' $\mathcal{SP}$ is also $\mathcal{O}(\mathrm{FL}(P, f))$. The latter fact follows because every such cell $\mathcal{C}$ in grid $\mathcal{G}(i)$ has a (smaller) active cell from $\mathcal{SP}$ within distance 2^{i-1}. These active cells are typically marked, with the result that the expected contribution of $\mathcal{C}$ is small.

Definition 5.2.3 (Height of a Cell)**.** *We say that a cell $\mathcal{C}$ in grid $\mathcal{G}(i)$ has* height *k if the smallest cell in $\mathcal{SP}$ that is contained in $\mathcal{C}$ has side length 2^{i-k}. If no cell in $\mathcal{SP}$ is contained in $\mathcal{C}$, then we define its height to be $-\infty$.*

Lemma 5.2.4. $\mathrm{FL}_{\mathrm{rand}}(P, f) \in \mathcal{O}(\mathrm{FL}(P, f))$ *with probability at least* 15/16*.*

Proof. Let $i \in [\lceil \log(\Delta) \rceil + 1]$ be any level, and let X_p denote the indicator random variable for the event $h_i(p) = 1$. Furthermore, for a cell $\mathcal{C}$ in grid $\mathcal{G}(i)$, let

$$X_{\mathcal{C}} := \sum_{p \in P \cap \mathcal{C}} X_p$$

denote the random variable for the number of sample points in cell $\mathcal{C}$. With this definition, it follows that, for every cell $\mathcal{C}$ in grid $\mathcal{G}(i)$, we have

$$\mathbf{E}[X_{\mathcal{C}}] = n_P(\mathcal{C}) \cdot \min\left\{\frac{1}{a(i)}, 1\right\} \quad .$$

By the definition of $a(i)$, we get

$$\mathbf{E}[X_{\mathcal{C}}] \leq \frac{n_P(\mathcal{C})}{a(i)} = \frac{n_P(\mathcal{C}) \cdot 2^i}{f} \quad .$$

For any $k \in \mathbb{N}_0$ and any level $i \in \{k, \ldots, \lceil \log(\Delta) \rceil\}$, let us now consider an arbitrary cell $\mathcal{C}$ in grid $\mathcal{G}(i)$ with height k. The cell $\mathcal{C}$ can be partitioned into cells $\mathcal{C}_1, \ldots, \mathcal{C}_\ell$ from $\mathcal{SP}$ that differ in their side lengths by a factor of at most 2^k. Since $\alpha(i) \leq 2^k \cdot \alpha(i-k)$, we have

$$\mathbf{E}[X_{\mathcal{C}}] \leq 2^k \cdot \mathbf{E}\left[\sum_{j=1}^{\ell} X_{\mathcal{C}_j}\right] \quad .$$

Observe that cells from the same grid cannot overlap and two cells from different grids only overlap if the smaller cell is completely contained in the bigger cell. Thus, due to the definition of height, the set of cells of height k do not overlap. Due to linearity of expectation, it follows that

$$\begin{aligned}
\mathbf{E}\left[\sum_{\text{cells } \mathcal{C} \text{ of height } k \text{ with } k \in \mathbb{N}_0} X_{\mathcal{C}}\right] &\leq 2^k \cdot \mathbf{E}\left[\sum_{i=0}^{\lceil \log(\Delta) \rceil} \sum_{\mathcal{C} \in \mathcal{SP}(i)} X_{\mathcal{C}}\right] \\
&\leq 2^k \cdot \sum_{i=0}^{\lceil \log(\Delta) \rceil} \sum_{\mathcal{C} \in \mathcal{SP}(i)} \frac{n_P(\mathcal{C}) \cdot 2^i}{f} \\
&\leq 2^k \cdot \frac{\mathrm{FL}(P, f)}{f} \quad .
\end{aligned}$$

Hence, for $k^* := \lceil \log(10\sqrt{d}) \rceil$, the expected number of sample points in cells with a non-negative height of at most k^* is less than $10\sqrt{d} \cdot \mathrm{FL}(P, f)/f$. Next, we consider the expected number of sample points in cells with a negative height. For any level $i \in [\lceil \log(\Delta) \rceil + 1]$, let $\mathcal{C}' \in \mathcal{SP}(i)$ be any cell with height 0. Then, the expected number of sample points in all the cells that are below $\mathcal{SP}$ and that are contained in $\mathcal{C}'$ is

$$\begin{aligned}
\mathbf{E}\left[\sum_{j=0}^{i-1} \sum_{\mathcal{C} \in \mathcal{G}(j): \mathcal{C} \subset \mathcal{C}'} X_{\mathcal{C}}\right] &\leq \sum_{j=0}^{i-1} \sum_{\mathcal{C} \in \mathcal{G}(j): \mathcal{C} \subset \mathcal{C}'} \frac{n_P(\mathcal{C}) \cdot 2^j}{f} \\
&= \sum_{j=0}^{i-1} \frac{n_P(\mathcal{C}') \cdot 2^{i-1-j}}{f} \\
&\leq \mathbf{E}[X_{\mathcal{C}'}] \quad .
\end{aligned}$$

Summing up over all cells in $\mathcal{SP}$, we obtain that the expected number of sample points in cells below $\mathcal{SP}$ is at most

$$\begin{aligned}\mathbf{E}\left[\sum_{\text{cells } \mathcal{C} \text{ of height } -\infty} X_{\mathcal{C}}\right] &\leq \sum_{i=0}^{\lceil \log(\Delta) \rceil} \sum_{\mathcal{C} \in \mathcal{SP}(i)} \mathbf{E}\left[X_{\mathcal{C}}\right] \\ &\leq \sum_{i=0}^{\lceil \log(\Delta) \rceil} \sum_{\mathcal{C} \in \mathcal{SP}(i)} \frac{n_P(\mathcal{C}) \cdot 2^i}{f} \\ &= \frac{\mathrm{FL}(P,f)}{f} \ .\end{aligned}$$

Thus, the expected number of sample points in cells with height at most k^* is less than $11\sqrt{d} \cdot \mathrm{FL}(P,f)/f$. By Markov's inequality, we obtain

$$\mathbf{Pr}\left[\sum_{\text{cells } \mathcal{C} \text{ of height at most } k^*} X_{\mathcal{C}} \geq 32 \cdot \mathbf{E}\left[\sum_{\text{cells } \mathcal{C} \text{ of height at most } k^*} X_{\mathcal{C}}\right]\right] \leq \frac{1}{32} \ .$$

Hence, with probability at least 31/32, the opening cost for facilities in cells with height at most k^* is less than $f \cdot 352\sqrt{d} \cdot \mathrm{FL}(P,f)/f \in \mathcal{O}(\mathrm{FL}(P,f))$.

Now, for any level $i \in \{k^*+1, \ldots, \lceil \log(\Delta) \rceil\}$, let us consider an arbitrary cell $\mathcal{C}$ in grid $\mathcal{G}(i)$ with height bigger than k^*. By the definition of height and the value of k^*, $\mathcal{C}$ contains a subcell from $\mathcal{SP}$ with side length less than $2^{i-k^*} \leq 2^i/(10\sqrt{d})$. Due to Property (*iii*) in Lemma 5.1.5, we know that, for any level $j \in [\lceil \log(\Delta) \rceil + 1]$, every cell in $\mathcal{SP}(j)$ has an active cell with side length at most 2^j in $\mathcal{SP}$ within a distance of at most $5\sqrt{d} \cdot 2^j$. We conclude that there is an active cell with side length less than $2^i/(10\sqrt{d})$ in $\mathcal{SP}$ within a distance of less than 2^{i-1} from $\mathcal{C}$. Now, observe that every parent cell of an active cell is active and contains the cell. Hence, there is a cell in grid $\mathcal{G}(i-1)$ within a distance of less than 2^{i-1} from $\mathcal{C}$ that is active. To simplify further descriptions, we will rephrase this as follows. Let the *level-j-neighborhood* of $\mathcal{C}$ be the set of all the cells in $\mathcal{G}(j)$ that share some part of their boundary or interior with $\mathcal{C}$. Then, every cell in grid $\mathcal{G}(i)$ with height at least k^* is active and contains an active cell in $\mathcal{SP}$ or has a cell in its level-$(i-1)$-neighborhood that is active and contains an active cell in $\mathcal{SP}$.

Now, we proceed as follows. For each active cell $\mathcal{A}_i$ in $\mathcal{SP}(i)$, we consider the cell itself and all cells that contain it. For each such cell $\mathcal{A}_j$ in grid $\mathcal{G}(j)$, $j \in \{i, \ldots, \lceil \log(\Delta) \rceil\}$, we assume that all the 2^d cells in the level-$(j+1)$-neighborhood of $\mathcal{A}_j$ contain an open facility if and only if $\mathcal{A}_j$ is not marked. Thus, the expected contribution of $\mathcal{A}_i$ and all the cells which belong to a level-j-neighborhood of $\mathcal{A}_i$ with $j \in \{i, \ldots, \lceil \log(\Delta) \rceil\}$ is at most

$$f + 2^d \cdot f \cdot \sum_{j=i}^{\lceil \log(\Delta) \rceil - 1} \mathbf{Pr}\left[\mathcal{A}_j \text{ is not marked}\right] \ .$$

Each point in $\mathcal{A}_j$ is not sampled with probability at most $1 - \min\{1/a(j), 1\}$. Hence, if $a(j) \leq 1$, we have $\mathbf{Pr}\left[\mathcal{A}_j \text{ is not marked}\right] = 0$. Otherwise, since $n_P(\mathcal{A}_j) \geq n_P(\mathcal{A}_i) \geq a(i)$,

we obtain

$$\begin{aligned}\mathbf{Pr}\left[\mathcal{A}_j \text{ is not marked}\right] &\leq \left(1-\frac{1}{a(j)}\right)^{n_P(\mathcal{A}_i)} \\ &\leq \exp\left(-\frac{n_P(\mathcal{A}_i)}{a(j)}\right) \\ &\leq \exp\left(-\frac{2^{j-i}\cdot n_P(\mathcal{A}_i)}{a(i)}\right) \\ &\leq \exp\left(-(j-i)\right) \ ,\end{aligned}$$

where the second inequality is due to a bound on Euler's number (see Inequality (B.2)). It follows that

$$\sum_{j=i}^{\lceil\log(\Delta)\rceil-1} \mathbf{Pr}\left[\mathcal{A}_j \text{ is not marked}\right] \leq \sum_{j=i}^{\lceil\log(\Delta)\rceil-1} e^{-(j-i)} \leq \sum_{j=0}^{\infty} e^{-j} = \frac{e}{e-1} \ .$$

Thus, the expected contribution of $\mathcal{A}_i$ and all the cells which belong to a level-j-neighborhood of $\mathcal{A}_i$ with $j \in \{i, \ldots, \lceil\log(\Delta)\rceil\}$ is at most $\mathcal{O}(f)$. Observe that each cell with height greater than k^* is a cell in such a neighborhood of an active cell since it contains itself an active cell from $\mathcal{SP}$ or one of its neighbors contains an active cell from $\mathcal{SP}$. It follows that the expected contribution from cells with height greater than k^* is $\mathcal{O}(f)$ times the number of active cells in $\mathcal{SP}$. Since a cell in any level $i \in [\lceil\log(\Delta)\rceil + 1]$ must contain at least $a(i)$ points to be active, the number of active cells in $\mathcal{SP}$ is

$$\begin{aligned}\sum_{i=0}^{\lceil\log(\Delta)\rceil} \sum_{\mathcal{C}\in\mathcal{SP}(i)} \min\left\{\left\lfloor\frac{n_P(\mathcal{C})}{a(i)}\right\rfloor, 1\right\} &\leq \sum_{i=0}^{\lceil\log(\Delta)\rceil} \sum_{\mathcal{C}\in\mathcal{SP}(i)} \frac{n_P(\mathcal{C})}{a(i)} \\ &= \frac{1}{f}\cdot \sum_{i=0}^{\lceil\log(\Delta)\rceil} \sum_{\mathcal{C}\in\mathcal{SP}(i)} n_P(\mathcal{C})\cdot 2^i \\ &\leq \frac{\mathrm{FL}(P,f)}{f} \ .\end{aligned}$$

Thus, the expected opening cost for facilities in cells with height greater than k^* is $\mathcal{O}(\mathrm{FL}(P,f))$. By Markov's inequality, the opening cost for facilities in cells with height greater than k^* is less than 32 times its expected value, which is also $\mathcal{O}(\mathrm{FL}(P,f))$, with probability at least 31/32. Together with the first part of the proof, we obtain that the total opening cost for facilities is $\mathcal{O}(\mathrm{FL}(P,f))$ with probability at least 15/16. □

5.3 Streaming Algorithm

In this section, we describe how our randomized algorithm can be transferred to the dynamic geometric data stream model. For each level $i \in [\lceil\log(\Delta)\rceil + 1]$, let $\mathcal{M}(i)$ be the

subset of marked cells in $\mathcal{G}(i)$, and let $\mathcal{U}(i)$ be the subset of cells in $\mathcal{G}(i)$ that have a cell contained in the set $\bigcup_{j=0}^{i-1} \mathcal{M}(j)$ within a distance of less than 2^{i-1}. Thus, we have $\mathrm{FL}_{\mathrm{rand}}(P, f) = f \cdot \sum_{i=0}^{\lceil \log(\Delta) \rceil} |\mathcal{M}(i) \backslash \mathcal{U}(i)|$. Recall that, in the streaming context, P refers to the current point set, i.e., the set of points obtained after having applied an input sequence of insertions and deletions.

The difficulty is to maintain, for each level $i \in [\lceil \log(\Delta) \rceil + 1]$, a good estimator for the value $|\mathcal{M}(i) \backslash \mathcal{U}(i)|$ in the streaming model. We use a similar technique as described in [64] to solve this problem. In particular, we use two data structures that both maintain the number of distinct elements in a stream, under insertions and deletions. The first data structure called $\mathrm{DE}_1(i)$ is supposed to maintain a good estimator for the value $|\mathcal{M}(i) \cup \mathcal{U}(i)|$. The second data structure called $\mathrm{DE}_2(i)$ is supposed to maintain a good estimator for the value $|\mathcal{U}(i)|$. We can show that the difference of these two estimators is a good estimator for the desired value $|\mathcal{M}(i) \backslash \mathcal{U}(i)|$. Next, we explain this method in more detail.

For any point $p \in \{1, \ldots, \Delta\}^d$ and any level $i \in [\lceil \log(\Delta) \rceil + 1]$, the cell in level i that contains the point p is denoted by $\mathcal{C}_p(i)$, and the set of neighbor cells of $\mathcal{C}_p(i)$ in level i is denoted by $\Gamma(\mathcal{C}_p(i))$. Furthermore, let $h_i : \{1, \ldots, \Delta\}^d \to \{0, 1\}$ be the random function introduced in Section 5.2.1. Then, our implementation of an insert operation of p is as follows (see also Algorithm 5.3.2). If $h_k(p) = 0$ for each index $k \in [\lceil \log(\Delta) \rceil + 1]$, then p is not a sample point and we do nothing with p. Otherwise, let $k \in [\lceil \log(\Delta) \rceil + 1]$ be the smallest index with $h_k(p) = 1$. Thus, p is sampled in level k, so $\mathcal{C}_p(k)$ is a marked cell. We insert $\mathcal{C}_p(k)$ in $\mathrm{DE}_1(k)$ and, for each $j \in \{k+1, k+2, \ldots, \lceil \log(\Delta) \rceil\}$, we insert all cells in $\mathcal{G}(j)$ such that $\mathcal{C}_p(k)$ is within a distance of less than 2^{j-1} in both $\mathrm{DE}_1(j)$ and $\mathrm{DE}_2(j)$. A deletion of p is implemented analogously (see Algorithm 5.3.3). After having processed the whole input stream according to this, we compute an estimator for the facility location cost. For each level $i \in [\lceil \log(\Delta) \rceil + 1]$, let $|\mathrm{DE}_1(i)|$ and $|\mathrm{DE}_2(i)|$ be the number of distinct elements in $\mathrm{DE}_1(i)$ and $\mathrm{DE}_2(i)$, respectively. Then, our estimator for the optimal facility location cost in the dynamic geometric data stream model is

$$\mathrm{FL}_{\mathrm{stream}}(P, f) := f \cdot \sum_{i=0}^{\lceil \log(\Delta) \rceil} |\mathrm{DE}_1(i)| - |\mathrm{DE}_2(i)| \ . \tag{5.6}$$

A description of our algorithm in pseudocode is given by Algorithms 5.3.1, 5.3.2, and 5.3.3.

5.3.1 Analysis of the Estimator

In this section, we show that our streaming algorithm outputs a constant-factor approximation of the optimal facility location cost, has polylogarithmic update time, and uses polylogarithmic memory space. For that purpose, we analyze the quality and complexity of the distinct elements data structures and the random sampling technique.

Distinct Elements Data Structures

It follows from the analysis in Sections 5.1 and 5.2 that $\mathrm{FL}_{\mathrm{rand}}(P, f) \in \Omega(\mathrm{FacLoc}^*(P, f))$ and $\mathrm{FL}_{\mathrm{rand}}(P, f) \in \mathcal{O}(\mathrm{FacLoc}^*(P, f))$ is true with probability at least $7/8$. Thus, with prob-

Algorithm 5.3.1 FacLocCost(f, Δ)

```
1: for i ← 0 to ⌈log(Δ)⌉ do
2:    create random function h_i(·)
3:    initialize empty data structures DE_1(i) and DE_2(i)
4: for each pair (p, operation) in the stream do
5:    if operation = insert then
6:       Insert(p)
7:    if operation = delete then
8:       Delete(p)
9: z ← 0
10: for i ← 0 to ⌈log(Δ)⌉ do
11:    z ← z + |DE_1(i)| − |DE_2(i)|
12: return f · z
```

Algorithm 5.3.2 Insert(p)

```
1: sampled ← false
2: i ← 0
3: while sampled = false and i ≤ ⌈log(Δ)⌉ do
4:    if h_i(p) = 1 then
5:       sampled ← true
6:       insert C_p(i) in DE_1(i)
7:       for j ← i + 1 to ⌈log(Δ)⌉ do
8:          insert all cells from G(j) that contain a cell from Γ(C_p(j − 1)) in DE_1(j) and
            DE_2(j)
9:    i ← i + 1
```

ability at least 7/8, we have that $1/c \cdot \text{FacLoc}^*(P, f) \leq \text{FL}_{\text{rand}}(P, f) \leq c \cdot \text{FacLoc}^*(P, f)$ for an appropriately chosen constant c. Next, we analyze how much the estimator $\text{FL}_{\text{stream}}(P, f)$ might differ from $\text{FL}_{\text{rand}}(P, f)$. For this purpose, let us assume that we have one fixed random function $h_i : \{1, \ldots, \Delta\}^d \to \{0, 1\}$ for each level $i \in [\lceil \log(\Delta) \rceil + 1]$ that is used by both our randomized algorithm and our streaming algorithm. We will show that the difference between $\text{FL}_{\text{stream}}(P, f)$ and $\text{FL}_{\text{rand}}(P, f)$, which is caused by using DE data structures to maintain the number of distinct elements in a data stream under insertions and deletions, can be upper bounded by $1/(2c) \cdot \text{FacLoc}^*(P, f)$ with probability greater than 7/8. Then, we have $1/(2c) \cdot \text{FacLoc}^*(P, f) \leq \text{FL}_{\text{stream}}(P, f) \leq (2c^2 + 1)/(2c) \cdot \text{FacLoc}^*(P, f)$ with high constant probability, which implies $\text{FL}_{\text{stream}}(P, f) \in \Theta(\text{FacLoc}^*(P, f))$ with high constant probability.

We use the technique proposed by Kane et al. [72] to realize the DE data structures. Then, due to Corollary 5.1.2, for a precision parameter ε and an error probability parameter δ, which we will both specify later, each of our DE data structures computes a $(1 \pm \varepsilon)$-approximation of the number of distinct elements in a data stream under insertions and deletions with probability at least $1 - \delta$.

Algorithm 5.3.3 DELETE(p)

```
1: sampled ← false
2: i ← 0
3: while sampled = false and i ≤ ⌈log(Δ)⌉ do
4:    if h_i(p) = 1 then
5:       sampled ← true
6:       delete C_p(i) from DE_1(i)
7:       for j ← i + 1 to ⌈log(Δ)⌉ do
8:          delete all cells from 𝒢(j) that contain a cell from Γ(C_p(j − 1)) from DE_1(j) and
            DE_2(j)
9:    i ← i + 1
```

We will show that the difference between $\mathrm{FL}_{\mathrm{stream}}(P,f)$ and $\mathrm{FL}_{\mathrm{rand}}(P,f)$ depends on the value $\sum_{i=0}^{\lceil\log(\Delta)\rceil} f\cdot|\mathcal{M}(i)|$. For that reason, we next give an appropriate upper bound of $\sum_{i=0}^{\lceil\log(\Delta)\rceil} f\cdot|\mathcal{M}(i)|$.

Lemma 5.3.1. *If* $\mathrm{FL}_{\mathrm{rand}}(P,f)\leq c\cdot\mathrm{FacLoc}^*(P,f)$*, then we have*

$$\sum_{i=0}^{\lceil\log(\Delta)\rceil} f\cdot|\mathcal{M}(i)|\leq c\cdot 2^d\cdot(\log(\Delta)+2)\cdot\mathrm{FacLoc}^*(P,f)\ .$$

Proof. We open in each marked cell in $\mathcal{G}(0)$ one facility. Thus, we have

$$f\cdot|\mathcal{M}(0)|=f\cdot|\mathcal{F}(0)|\leq\mathrm{FL}_{\mathrm{rand}}(P,f)\leq c\cdot\mathrm{FacLoc}^*(P,f)\ .$$

For any level $i\in[\lceil\log(\Delta)\rceil+1]$, we open a facility in each cell in $\mathcal{M}(i)$ such that no subcell of this cell is marked and no smaller cell within a distance of less than 2^{i-1} is marked. Let us consider any cell $\mathcal{C}$ in level $i-1$ that is a marked cell or contains a marked cell. Let $\mathcal{C}'$ be this marked cell. There are 2^d cells in $\mathcal{G}(i)$ such that $\mathcal{C}$ is within a distance of less than 2^{i-1} from these cells, namely the set of cells in the level-i-neighborhood of $\mathcal{C}$ (see Figure 5.4). Recall that the level-i-neighborhood of $\mathcal{C}$ is the set of all the cells in level i that share some part of their boundary or interior with $\mathcal{C}$. It follows that $\mathcal{C}'$ prevents at most 2^d cells in $\mathcal{M}(i)$ from opening a facility. Now, either $\mathcal{C}'$ contains an open facility or there exists a smaller marked cell $\mathcal{C}''$ that prevents $\mathcal{C}'$ from opening a facility. Since $\mathcal{C}''$ has to be within distance of less than 2^{i-2} from $\mathcal{C}'\subseteq\mathcal{C}$, it is located in the level-$(i-2)$-neighborhood of $\mathcal{C}$. We can recursively apply this argument until we have found the first marked cell that is not prevented by any smaller marked cell from opening a facility. Note that this happens in level 0 at the latest. Since $\sum_{j=0}^{i-2}2^j\leq 2^{i-1}$, these marked cells are all located in the level-$(i-1)$-neighborhood of $\mathcal{C}$ and, thus, also in the level-i-neighborhood of $\mathcal{C}$. Hence, for a fraction of at least $1/2^d$ cells in $\mathcal{M}(i)$, there exists at least one cell in $\bigcup_{j=0}^{i}\mathcal{F}(j)$. Thus, we have

$$f\cdot|\mathcal{M}(i)|\leq f\cdot 2^d\cdot\sum_{j=0}^{i}|\mathcal{F}(j)|\leq 2^d\cdot\mathrm{FL}_{\mathrm{rand}}(P,f)\leq 2^d\cdot c\cdot\mathrm{FacLoc}^*(P,f)\ .$$

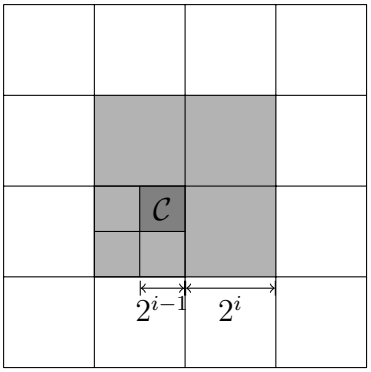


Figure 5.4: Illustration of the area of influence of a marked cell. The cell $\mathcal{C} \in \mathcal{G}(i-1)$ is a marked cell or contains a marked cell. The area of influence in grid $\mathcal{G}(i)$, i.e., the subset of cells in $\mathcal{G}(i)$ that are within distance of less than 2^{i-1} from $\mathcal{C}$, is colored in light gray. The white cells in grid $\mathcal{G}(i)$ are outside of the influence of the marked cell in $\mathcal{C}$.

Now, the lemma follows from the fact that there are at most $\log(\Delta) + 2$ levels.

□

Finally, we can upper bound the difference between $\mathrm{FL}_{\mathrm{stream}}(P, f)$ and $\mathrm{FL}_{\mathrm{rand}}(P, f)$.

Lemma 5.3.2. *If* $\mathrm{FL}_{\mathrm{rand}}(P, f) \leq c \cdot \mathrm{FacLoc}^*(P, f)$ *and if we run each* DE *data structure to maintain a* $(1 \pm \varepsilon)$*-approximation of the number of distinct elements in data streams under insertions and deletions with a precision parameter* ε*,* $0 < \varepsilon \leq 1/(8c^2 \cdot 2^{2d} \cdot (\log(\Delta) + 2)^2)$*, and an error probability* δ*,* $0 < \delta < 1/(16(\log(\Delta) + 2))$*, then*

$$|\,\mathrm{FL}_{\mathrm{stream}}(P, f) - \mathrm{FL}_{\mathrm{rand}}(P, f)| < \frac{1}{2c} \cdot \mathrm{FacLoc}^*(P, f)$$

with probability greater than $7/8$*.*

Proof. Since we use two DE data structures per level and there are at most $(\log(\Delta) + 2)$ levels, we use at most $2(\log(\Delta) + 2)$ DE data structures in total. By the union bound, the probability that at least one of these DE data structures fails is less than $1/8$. Hence, the probability that each DE data structure maintains a $(1 \pm \varepsilon)$-approximation is greater than $7/8$.

Since we run each DE data structure with a precision parameter ε, we can upper bound the difference between $\mathrm{FL}_{\mathrm{stream}}(P, f)$ and $\mathrm{FL}_{\mathrm{rand}}(P, f)$ by

$$|\,\mathrm{FL}_{\mathrm{stream}}(P, f) - \mathrm{FL}_{\mathrm{rand}}(P, f)| \leq \varepsilon \cdot f \cdot \sum_{i=0}^{\lceil \log(\Delta) \rceil} |\mathcal{M}(i)| + 2 \cdot |\mathcal{U}(i)| \ .$$

Due to Lemma 5.3.1 and $\varepsilon \leq 1/(8c^2 \cdot 2^{2d} \cdot (\log(\Delta)+2)^2) < 1/(4c^2 \cdot 2^d \cdot (\log(\Delta)+2))$, we have

$$\begin{aligned} \varepsilon \cdot f \cdot \sum_{i=0}^{\lceil \log(\Delta) \rceil} |\mathcal{M}(i)| &\leq \varepsilon \cdot c \cdot 2^d \cdot (\log(\Delta)+2) \cdot \mathrm{FacLoc}^*(P,f) \\ &< \frac{1}{4c} \cdot \mathrm{FacLoc}^*(P,f) \ . \end{aligned}$$

Next, we upper bound the value $\varepsilon \cdot f \cdot \sum_{i=0}^{\lceil \log(\Delta) \rceil} |\mathcal{U}(i)|$. Observe that the set $\mathcal{U}(0)$ is empty. Furthermore, for any cell $\mathcal{C} \in \bigcup_{j=0}^{i-1} \mathcal{M}(j)$, there are at most 2^d cells in $\mathcal{G}(i)$ that are within a distance of less than 2^{i-1} from $\mathcal{C}$. Thus, there exists at least one cell in $\bigcup_{j=0}^{i-1} \mathcal{M}(j)$ for a fraction of at least $1/2^d$ cells in $\mathcal{U}(i)$. Hence, we have

$$|\mathcal{U}(i)| \leq 2^d \cdot \sum_{j=0}^{i-1} |\mathcal{M}(j)| \leq 2^d \cdot \sum_{j=0}^{\lceil \log(\Delta) \rceil} |\mathcal{M}(j)| \ .$$

Summation over all levels results in

$$\sum_{i=0}^{\lceil \log(\Delta) \rceil} |\mathcal{U}(i)| \leq 2^d \cdot (\log(\Delta)+2) \cdot \sum_{i=0}^{\lceil \log(\Delta) \rceil} |\mathcal{M}(i)| \ .$$

Due to Lemma 5.3.1 and $\varepsilon \leq 1/(8c^2 \cdot 2^{2d} \cdot (\log(\Delta)+2)^2)$, we get

$$\begin{aligned} \varepsilon \cdot f \cdot \sum_{i=0}^{\lceil \log(\Delta) \rceil} 2 \cdot |\mathcal{U}(i)| &\leq \varepsilon \cdot f \cdot 2 \cdot 2^d \cdot (\log(\Delta)+2) \cdot \sum_{i=0}^{\lceil \log(\Delta) \rceil} |\mathcal{M}(i)| \\ &\leq \varepsilon \cdot f \cdot 2c \cdot 2^{2d} \cdot (\log(\Delta)+2)^2 \cdot \mathrm{FacLoc}^*(P,f) \\ &\leq \frac{1}{4c} \cdot \mathrm{FacLoc}^*(P,f) \ . \end{aligned}$$

Thus, we obtain

$$|\,\mathrm{FL}_{\mathrm{stream}}(P,f) - \mathrm{FL}_{\mathrm{rand}}(P,f)| \leq \varepsilon \cdot f \cdot \sum_{i=0}^{\lceil \log(\Delta) \rceil} |\mathcal{M}(i)| + |\mathcal{U}(i)| < \frac{1}{2c} \cdot \mathrm{FacLoc}^*(P,f)$$

with probability greater than 7/8. □

We summarize our results achieved so far in the following lemma. Note that Lemma 5.3.3 considers only the space requirement and update time of the DE data structures. The space and time needed to do the random sampling will be analyzed later.

Lemma 5.3.3. *If we run each* DE *data structure to maintain a* $(1\pm\varepsilon)$*-approximation of the number of distinct elements in data streams under insertions and deletions with a precision parameter* $\varepsilon := 1/(8c^2 \cdot 2^{2d} \cdot (\log(\Delta)+2)^2)$ *and an error probability* $\delta := 1/(17(\log(\Delta)+2))$*, then* $\mathrm{FL}_{\mathrm{stream}}(P,f) \in \Theta(\mathrm{FacLoc}^*(P,f))$ *with probability greater than* 3/4*. The* DE *data structures require* $\mathcal{O}(\log^6(\Delta) \cdot (\log(\log(\Delta)))^2)$ *bits of space and* $\mathcal{O}(\log(\Delta) \cdot \log(\log(\Delta)))$ *update time.*

Proof. Due to Lemmas 5.1.6, 5.1.7, 5.2.2, and 5.2.4, we have

$$\frac{1}{c} \cdot \mathrm{FacLoc}^*(P, f) \leq \mathrm{FL}_{\mathrm{rand}}(P, f) \leq c \cdot \mathrm{FacLoc}^*(P, f)$$

for an appropriately chosen constant $c \geq 1$ with probability at least 7/8. If this is the case and each DE data structure is run with a precision parameter $\varepsilon = 1/(8c^2 \cdot 2^{2d} \cdot (\log(\Delta)+2)^2)$ and an error probability $\delta = 1/(17(\log(\Delta)+2))$, then it follows from Lemma 5.3.2 that the difference between $\mathrm{FL}_{\mathrm{stream}}(P, f)$ and $\mathrm{FL}_{\mathrm{rand}}(P, f)$ is at most $1/(2c) \cdot \mathrm{FacLoc}^*(P, f)$ with probability greater than 7/8. Thus, we obtain

$$\frac{1}{2c} \cdot \mathrm{FacLoc}^*(P, f) \leq \mathrm{FL}_{\mathrm{stream}}(P, f) \leq \frac{2c^2+1}{2c} \cdot \mathrm{FacLoc}^*(P, f)$$

with probability greater than 3/4. This proves that $\mathrm{FL}_{\mathrm{stream}}(P, f) \in \Theta(\mathrm{FacLoc}^*(P, f))$ with probability greater than 3/4.

Due to Corollary 5.1.2 and for our values of ε and δ, each DE data structure has a space requirement of

$$\begin{aligned} & \mathcal{O}(1/\varepsilon^2 \cdot \log(N) \cdot (\log(1/\varepsilon) + \log(\log(M))) \cdot \log(1/\delta)) \\ = \; & \mathcal{O}(\log(\Delta)^4 \cdot \log(N) \cdot (\log(\log(\Delta)) + \log(\log(M))) \cdot \log(\log(\Delta))) \end{aligned}$$

bits, where N is the size of the domain of the elements and M is the multiplicity of single elements in the DE data structure. Since the grid of any level contains at most Δ^d cells and each DE data structure contains only cells from one level, we have $N = \Delta^d$. Furthermore, due to our implementation of Insert(p) and Delete(p) for any point $p \in \{1, \ldots, \Delta\}^d$, the multiplicity of a cell in any DE data structure is at most $M = \Delta^d$. Hence, each DE data structure needs $\mathcal{O}(\log^5(\Delta) \cdot (\log(\log(\Delta)))^2)$ bits of space. Since we use $\mathcal{O}(\log(\Delta))$ DE data structures, the total space requirement for the DE data structures is upper bounded by $\mathcal{O}(\log^6(\Delta) \cdot (\log(\log(\Delta)))^2)$ bits.

While running Insert(p) for any point $p \in \{1, \ldots, \Delta\}^d$, we insert at most a constant number of cells in each DE data structure. Thus, due to Corollary 5.1.2 and for our value of δ, the time to process an Insert operation is $\mathcal{O}(\log(\log(\Delta)))$ for each DE data structure. Analogously, we can upper bound the time to process a Delete operation. It follows that the total update time for the $\mathcal{O}(\log(\Delta))$ DE data structures is $\mathcal{O}(\log(\Delta) \cdot \log(\log(\Delta)))$, which completes the proof of the lemma. □

Random Sampling

For each level $i \in [\lceil \log(\Delta) \rceil + 1]$, we use the random function $h_i : \{1, \ldots, \Delta\}^d \to \{0, 1\}$ described in Section 5.2.1 to realize the random sampling of points. To overcome the assumption of full randomness needed for the creation of these $h_i(\cdot)$, we use a pseudo-random generator of Nisan [95]. This approach was first proposed in [62].

First, we briefly summarize some facts of pseudo-random generators for space-bounded computation proposed by Nisan [95]. Then, we show how to utilize these facts for the

creation of the random functions. Let ALG be any algorithm that uses at most $\mathcal{O}(k)$ bits of memory and, thus, has at most $2^{\mathcal{O}(k)}$ distinct states. Furthermore, we assume that ALG uses at most g chunks of random bits, where each chunk is of length $\ell \in \mathcal{O}(k)$. Let $\mathrm{ALG}(x)$ be the state of ALG after having used the random bit sequence $x \in \{0,1\}^{g\ell}$. Then, there is a pseudo-random generator for ALG with the following properties:

Lemma 5.3.4 ([95])**.** *Let* ALG *be an algorithm that uses $\mathcal{O}(k)$ bits of memory and g chunks of random bits, where each chunk is of length $\ell \in \mathcal{O}(k)$. Then, there exists a pseudo-random generator $R : \{0,1\}^s \to (\{0,1\}^\ell)^g$ for* ALG *which expands s random bits into $t := g \cdot \ell$ bits such that $s \in \mathcal{O}(k \log(t))$ and*

$$\sum_{\text{states } z \text{ of ALG}} |\mathbf{Pr}\left[\mathrm{ALG}(x) = z\right] - \mathbf{Pr}\left[\mathrm{ALG}(R(y)) = z\right]| \leq 2^{-k}$$

where x is chosen uniformly at random from $\{0,1\}^t$ and y is chosen uniformly at random from $\{0,1\}^s$. For any $y \in \{0,1\}^s$, any length-ℓ chunk of $R(y)$ can be computed using $\mathcal{O}(\log(t))$ arithmetic operations on $\mathcal{O}(\ell)$-bit words.

In the proof of the following lemma, we show how to apply a pseudo-random generator to reduce the randomness needed for the creation of the random functions. To do so, we proceed in a similar way as Indyk [62, 65].

Lemma 5.3.5. *There is an implementation of algorithm* FacLocCost *that outputs a constant-factor approximation of the optimal facility location cost for the current point set with probability greater than $2/3$. The implementation requires $\mathcal{O}(\log^7(\Delta) \cdot (\log(\log(\Delta)))^2)$ bits of space and $\mathcal{O}(\log^7(\Delta) \cdot (\log(\log(\Delta)))^2)$ random bits. An insertion or deletion of a point requires $\mathcal{O}(\log^2(\Delta))$ arithmetic operations on $\mathcal{O}(\log(\Delta))$-bit words.*

Proof. Let FLCFullyRandom be the implementation of algorithm FacLocCost that uses the type of DE data structures given in Corollary 5.1.2 and the kind of random functions proposed in Lemma 5.2.1. To prove the lemma, we adopt the argumentation given by Indyk in [62, 65]. Due to Lemma 5.3.3, the total space requirement for the DE data structures is upper bounded by $\mathcal{O}(\log^6(\Delta) \cdot (\log(\log(\Delta)))^2)$ bits. Furthermore, FLCFullyRandom requires $\mathcal{O}(\log^2(\Delta))$ random bits per point in $\{1, \ldots, \Delta\}^d$ in total for the creation of the $\mathcal{O}(\log(\Delta))$ random functions. Since we might access a specific point several times and the output of each random function for this point should be always the same, we have to store the random bits of all Δ^d points. Obviously, an algorithm working in the dynamic geometric data stream model cannot use $\Omega(\Delta)$ bits of space. This problem is avoidable by allowing a negligible probability of error in the computation of the number of open facilities. For the moment, let us assume the stream is sorted, which means that the insertions and deletions of a specific point occur subsequently in the stream. Then, it is sufficient to compute the output of each random function only once per point. Thus, in case of a sorted stream, algorithm FLCFullyRandom uses $\mathcal{O}(\log^6(\Delta) \cdot (\log(\log(\Delta)))^2)$ bits and $\mathcal{O}(\Delta^d \cdot \log(\Delta))$ chunks each consisting of $\mathcal{O}(\log(\Delta))$ random bits. Note that there are $\mathcal{O}(\log(\Delta))$ chunks per point in $\{1, \ldots, \Delta\}^d$, one for each

level. Due to Lemma 5.3.4, there exists a pseudo-random generator R which given a random seed of size $\mathcal{O}(\log^6(\Delta)\cdot(\log(\log(\Delta)))^2\cdot\log(\Delta))$ expands it to $\Delta^d\cdot\log(\Delta)$ chunks of $\mathcal{O}(\log(\Delta))$ random bits such that each chunk can be computed using $\mathcal{O}(\log(\Delta))$ arithmetic operations on $\mathcal{O}(\log(\Delta))$-bit words and using these chunks results in negligible probability of error in the computation of the number of open facilities. Let us denote the implementation of algorithm FacLocCost which uses a pseudo-random generator R for the creation of the random functions by FLCPseudoRandom. Then, according to Lemma 5.3.4 and since we can assume that $\Delta \geq 4$, the probability that the implementation FLCFullyRandom differs in its computations from the ones of the implementation FLCPseudoRandom is at most

$$\begin{aligned}\mathbf{Pr}\left[\text{FLCFullyRandom} \neq \text{FLCPseudoRandom}\right] &\leq 2^{-\log^6(\Delta)\cdot(\log(\log(\Delta)))^2} \\ &\leq 2^{-6\log(\Delta)} \\ &= \Delta^{-6}\ .\end{aligned}$$

Since, for a fixed random seed, algorithm FLCPseudoRandom does not depend on the order in which the insertions and deletions of points appear in the stream, we also get $\mathbf{Pr}\left[\text{FLCFullyRandom} \neq \text{FLCPseudoRandom}\right] \leq 1/\Delta^6$ for the unsorted stream.

Due to Lemma 5.3.3, we obtain that the implementation FLCFullyRandom of algorithm FacLocCost has an error probability of less than $1/4$. Since we assume that $\Delta \geq 4$ and $\mathbf{Pr}\left[\text{FLCFullyRandom} \neq \text{FLCPseudoRandom}\right] \leq 1/\Delta^6$, the implementation FLCPseudoRandom works with error probability less than $1/3$.

Due to Lemma 5.3.3, the space requirement of the DE data structures is upper bounded by $\mathcal{O}(\log^6(\Delta)\cdot(\log(\log(\Delta)))^2)$ bits and their update time is $\mathcal{O}(\log(\Delta)\cdot\log(\log(\Delta)))$. For the random functions, the pseudo-random generator of Nisan [95] needs a random seed of size $\mathcal{O}(\log^7(\Delta)\cdot(\log(\log(\Delta)))^2)$. Furthermore, for any level $i \in [\lceil\log(\Delta)\rceil + 1]$ and any point $p \in \{1,\ldots,\Delta\}^d$, the value $h_i(p)$ can be computed using $\mathcal{O}(\log(\Delta))$ arithmetic operations on $\mathcal{O}(\log(\Delta))$-bit words. Thus, we need $\mathcal{O}(\log^7(\Delta)\cdot(\log(\log(\Delta)))^2)$ random bits in total and, for any point, the output values of all random functions can be computed using $\mathcal{O}(\log^2(\Delta))$ arithmetic operations on $\mathcal{O}(\log(\Delta))$-bit words. □

As explained in the proof of Corollary 5.1.2, we can use a standard amplification technique to obtain the following main result:

Theorem 5. *There is a randomized streaming algorithm that computes with probability $1-\delta$ a constant-factor approximation of the facility location cost for a stream of points with uniform opening costs and demands in the discrete Euclidean space $\{1,\ldots,\Delta\}^d$ under insertions and deletions, where d is a constant. The algorithm has a space requirement of $\mathcal{O}(\log^7(\Delta)\cdot(\log(\log(\Delta)))^2\cdot\log(1/\delta))$ bits and uses $\mathcal{O}(\log^7(\Delta)\cdot(\log(\log(\Delta)))^2\cdot\log(1/\delta))$ random bits. An insertion or deletion of a point requires $\mathcal{O}(\log^2(\Delta)\cdot\log(1/\delta))$ arithmetic operations on $\mathcal{O}(\log(\Delta))$-bit words.*

6 A k-Means Implementation for Data Streams

In this chapter, we develop an efficient algorithm for the k-means clustering problem in the insertion-only data stream model. We call our algorithm StreamKM++. The k-means clustering problem is closely related to the facility location problem. Given a set of points and a natural number k, the goal of the k-means clustering problem is to place k facilities, the so-called cluster centers, such that the sum of the squared distances of the points to their nearest cluster center is minimized. To approach this problem, our streaming algorithm maintains a small summary of the input points using the merge-and-reduce technique [16, 58], i.e., the data is organized in a small number of samples, each representing $2^i m$ input points (for some $i \in \mathbb{N}_0$ and a fixed $m \in \mathbb{N}$). Every time when two samples representing the same number of input points exist, we take the union (merge) and create a new sample (reduce). After having processed the whole input stream in this way, we apply the k-Means++ algorithm [9] on the sample to obtain a k-means clustering.

For the reduce step, we develop a new coreset construction. A coreset is a small weighted point set that approximates the original input point set with respect to a given optimization problem, which is in our case the k-means clustering problem. Our focus is to propose a coreset construction that is suitable for high-dimensional data. Existing constructions based on grid-computations [44, 58] yield coresets of a size that is exponential in the dimension. Since the k-Means++ seeding works well for high-dimensional data, a coreset construction based on this approach seems to be more promising. We give a theoretical analysis of this approach in Section 6.2.

In order to implement this coreset construction efficiently, we propose a new data structure, which we call *coreset tree*. This is a tree-like data structure that stores points in such a way that we can perform a fast adaptive sampling which is very similar to the k-Means++ seeding. According to our experiments, the seed computed on the coreset tree has essentially the same properties as the original k-Means++ seed. The advantage of the coreset tree approach is that the running time is significantly shorter than the running time of the original k-Means++ seeding.

It should be noted that the k-Means++ seeding has also been theoretically investigated in [3] and [4]. Aggarwal et al. [3] used the k-Means++ seeding to obtain a small weighted point set such that an optimal k-means clustering of the original point set can be approximated well by clustering the small weighted set. Ailon et al. [4] used the k-Means++ seeding to obtain a streaming algorithm for the k-means clustering problem that guarantees an approximation factor of $\mathcal{O}(c^\alpha \log(k))$, where c is some constant, $\alpha \approx \log(n)/\log(M)$, n is the number of input points in the stream, and M is the amount of work memory available to the algorithm. However, our result differs from the results given in [3] and [4] and was obtained independently.

In Section 6.5, we compare algorithm StreamKM++ with algorithms BIRCH [111] and StreamLS [96, 52] as well as with the non-streaming version of algorithm k-Means++. It turns out that our algorithm is slower than BIRCH, but it computes significantly better solutions (in terms of the sum of squared errors). In addition, to obtain the desired number of clusters, our algorithm does not require the trial-and-error adjustment of parameters as BIRCH does. The quality of the clustering of algorithm StreamLS is comparable to that of our algorithm, but the running time of StreamKM++ scales much better with the number of cluster centers. For example, on the dataset *Tower*, our algorithm computes a clustering with $k = 100$ centers in about 3% of the running time of StreamLS. In comparison with the standard implementation of k-Means++, our algorithm runs much faster on larger datasets and computes solutions that are on a par with k-Means++. For example, on the dataset *Covertype*, our algorithm computes a clustering with $k = 50$ centers of essentially the same quality as k-Means++ does, but it needs only about 3% of the running time of k-Means++.

Next, we introduce some notation and give a brief overview of the considered competitors of StreamKM++.

6.1 Preliminaries

6.1.1 Definition of Euclidean k-Means Clusterings

Recall from Section 2.1 that, for any two points $p, q \in \mathbb{R}^d$ and any set of points $C \subset \mathbb{R}^d$, we denote the Euclidean distance between p and q by $\mathrm{D}(p, q) := \|p - q\|$, and we define

$$\mathrm{D}(p, C) := \min_{c \in C} \mathrm{D}(p, c) \ .$$

Similarly, for squared Euclidean distances, we define

$$\mathrm{D}^2(p, q) := \|p - q\|^2 \quad \text{and} \quad \mathrm{D}^2(p, C) := \min_{c \in C} \mathrm{D}^2(p, c) \ .$$

Let $P \subset \mathbb{R}^d$ be a set of points with size $|P| =: n$. The *Euclidean k-means clustering problem* for P is given as follows.

Definition 6.1.1 (Euclidean k-Means Clustering Problem)**.** *For a set $P \subset \mathbb{R}^d$ and $k \in \mathbb{N}$, the* Euclidean k-means clustering problem *is to find a set $C := \{c_1, \ldots, c_k\}$ of k cluster centers in $\mathbb{R}^d$ and a partition of the set P into k clusters $C_1, \ldots, C_k$ such that the k-means clustering cost*

$$\mathrm{Means}(P, C, C_1, \ldots, C_k) := \sum_{i=1}^{k} \sum_{p \in C_i} \mathrm{D}^2(p, c_i)$$

is minimized.

Analogously, for a weighted set $S \subset \mathbb{R}^d$ with weight function $w : S \to \mathbb{R}_{\geq 0}$ and $k \in \mathbb{N}$, the weighted Euclidean k-means clustering problem *is to find a set $C := \{c_1, \ldots, c_k\}$ of k*

cluster centers in $\mathbb{R}^d$ *and a partition of the set* S *into* k *clusters* $C_1, \ldots, C_k$ *such that the* k*-means clustering cost*

$$\text{Means}(S, C, C_1, \ldots, C_k) := \sum_{i=1}^{k} \sum_{q \in C_i} w(q) \cdot \text{D}^2(q, c_i)$$

is minimized.

If a partition $C_1, \ldots, C_k$ of P relates each point to its nearest cluster center, i.e., if, for each $p \in P$ and each $i \in \{1, \ldots, k\}$, we have

$$p \in C_i \Rightarrow \text{D}(p, c_i) = \min_{j \in \{1,\ldots,k\}} \text{D}(p, c_j) \ ,$$

then we shortly write

$$\text{Means}(P, C) := \text{Means}(P, C, C_1, \ldots, C_k) \ .$$

Furthermore, we denote the cost of an optimal Euclidean k-means clustering of P by

$$\text{Means}_k^*(P) := \min_{C' \subset \mathbb{R}^d : |C'| = k} \text{Means}(P, C') \ .$$

6.1.2 Definition of Coresets

An important concept that we use is the notion of coresets. Generally speaking, a coreset for a set P is a small (weighted) set such that, for any set of k cluster centers, the (weighted) clustering cost for the coreset is an approximation of the clustering cost for the original set P with small relative error. The advantage of such a coreset is that we can apply any fast approximation algorithm (for the weighted problem) on the usually much smaller coreset to compute an approximate solution for the original set P more efficiently. We use the following formal definition:

Definition 6.1.2 (Coreset for k-Means Clustering Problem)**.** *Let* $P \subset \mathbb{R}^d$ *be a set of points, let* $k \in \mathbb{N}$*, and let* ε*,* $0 < \varepsilon \leq 1$*, be a precision parameter. A weighted multiset* $S \subset \mathbb{R}^d$ *with positive weight function* $w : S \to \mathbb{R}_{\geq 0}$ *is called* (k, ε)*-coreset of* P *for the* k*-means clustering problem if, for each* $C \subset \mathbb{R}^d$ *of size* $|C| = k$*, we have*

$$(1 - \varepsilon) \cdot \text{Means}(P, C) \ \leq \ \text{Means}(S, C) \ \leq \ (1 + \varepsilon) \cdot \text{Means}(P, C) \ .$$

Our clustering algorithm maintains a small coreset in the insertion-only data stream model (see Section 2.4.4 for a definition of this data stream model).

6.1.3 k-Means Clustering Algorithms

In the experiments, we compare StreamKM++ with two frequently used clustering algorithms for processing data streams, namely with algorithm BIRCH of Zhang et al. [111] and with a streaming variant of the local search algorithm given by O'Callaghan et al. [96] and Guha et al. [52], which we refer to as algorithm StreamLS. On smaller datasets, we also compare our algorithm with a classical implementation of Lloyd's k-means algorithm [80], using initial seeds either uniformly at random (algorithm k-Means) or according to the adaptive, non-uniform seeding from Arthur and Vassilvitskii [9] (algorithm k-Means++). In the following, we will give a brief overview of these k-means clustering algorithms.

Algorithm k-Means

One of the most widely used clustering algorithms is Lloyd's algorithm. This algorithm is sometimes also called *the k-means algorithm* [80, 39, 82]. Lloyd's algorithm is based on two observations:

1. Given a fixed set of centers, we obtain the best clustering by assigning each point to the nearest center.

2. Given a cluster, the best center of the cluster is the center of gravity (i.e., the mean) of its points.

Lloyd's algorithm applies these two local optimizations steps repeatedly to the current solution, until no more improvement is possible. See Algorithm 6.1.1 for a description in pseudocode.

Algorithm 6.1.1 k-Means(P, k)

1: choose k initial centers $c_1, \ldots, c_k$ uniformly at random from P
2: **repeat**
3: partition P into k subsets $P_1, \ldots, P_k$ such that P_i, $1 \leq i \leq k$, contains all points whose nearest center is c_i
4: replace the current set of centers by a new set of centers $c_1, \ldots, c_k$ such that center c_i, $1 \leq i \leq k$, is the center of gravity of P_i
5: **until** the set of centers has not changed

It is known that the algorithm converges to a local optimum [100], and the quality of the computed solution is sensitive to the choice of the starting centers. Kanungo et al. [73] give a simple example where, for a fixed set of starting centers, Lloyd's algorithm converges to a local minimum that is arbitrarily bad compared to the optimal solution. This example can be extended to the case where the starting centers are chosen by uniform seeding as given in Algorithm 6.1.1.

Algorithm k-Means++

Recently, Arthur and Vassilvitskii developed the k-Means++ algorithm [9], which is a seeding procedure for Lloyd's k-means algorithm. This seeding procedure considers the fact that the quality of the solution of Lloyd's k-means algorithm depends strongly on the initial set of centers. In order to achieve a better arrangement, it chooses the initial set of centers adaptively and non-uniformly at random by choosing each point as the next center with probability proportional to its squared distance from the nearest center already chosen. Note that, for a given set of centers, the squared distance of a point from its nearest center corresponds to the current contribution of this point to the total k-means clustering cost. The k-Means++ seeding procedure is given by Algorithm 6.1.2. For simplicity of description, we say that Algorithm 6.1.2 chooses the set C *at random according to* D^2.

Algorithm 6.1.2 AdaptiveSeeding(P, k)

1: choose an initial center c_1 uniformly at random from P
2: $C \leftarrow \{c_1\}$
3: **for** $i \leftarrow 2$ **to** k **do**
4: choose the next center c_i at random from P, where the probability of each $p \in P$ is given by $\mathrm{D}^2(p, C)/\operatorname{Means}(P, C)$
5: $C \leftarrow C \cup \{c_i\}$

By replacing line 1 of Algorithm 6.1.1 with Algorithm 6.1.2, Arthur and Vassilvitskii developed a k-means clustering algorithm, known as k-Means++ algorithm, which gives good experimental results and guarantees a solution with certain quality. More precisely, they showed the following:

Lemma 6.1.3 ([9])**.** *Let $C \subseteq P$ be a set of k points chosen at random according to D^2. Then, we have*

$$\mathbf{E}\left[\operatorname{Means}(P, C)\right] \leq 8\left(2 + \ln(k)\right) \operatorname{Means}_k^*(P) \ .$$

Algorithm BIRCH

One of the earliest and best known practical clustering algorithms for data streams is BIRCH (which is an acronym for 'Balanced Iterative Reducing and Clustering using Hierarchies') [111]. BIRCH is a heuristic which exploits the observation that the point space is usually not uniformly occupied. It scans the given set of input points once and computes a pre-clustering by summarizing dense regions of points by their so-called clustering features. Such a clustering feature consists of the number of points in the region, the center of gravity, and the sum of squared distances to the origin. Thereby, the problem of clustering the original input point set is reduced to the problem of clustering the set of summaries, which is much smaller than the original point set. The pre-clustering is then clustered by using an agglomerative (bottom-up) clustering algorithm. In this process, the algorithm uses the clustering features to calculate the intra-cluster distances. BIRCH successively merges the closest pair of clusters until the desired number of clusters is obtained.

To a certain extent, BIRCH uses a kind of coreset construction. However, there is no theoretical analyses of this method known. For more details about BIRCH, the reader is referred to [111].

Algorithm StreamLS

Another well-known clustering algorithm for data streams is the streaming implementation of algorithm LSearch from O'Callaghan et al. [96] and Guha et al. [52], which we refer to as StreamLS. This algorithm partitions the input stream into chunks and computes for each chunk a k-means clustering solution using a local search algorithm from Guha et al. [53]. Finally, the local search algorithm is applied once more on the union of the solutions for the chunks to obtain a k-means clustering for the whole input stream.

The local search algorithm of Guha et al. [53] takes advantage of the relationship between the k-means clustering problem and the uniform facility location problem (see Section 2.2 for a definition of the uniform facility location problem). More precisely, it is based on the observation that if the opening cost of a facility increases, then the number of facilities (or cluster centers) of an optimal solution tends to decrease. Hence, to solve the k-means problem, the algorithm of Guha et al. [53] performs a binary search on the opening cost of a facility to find a cost that gives the desired number of cluster centers. During the binary search, each facility location problem is solved by starting with an initial solution that is obtained by a simple non-uniform sampling approach and then refining this solution by making local improvements. More details can be found in [96, 53, 52].

6.2 Coreset Construction

Our k-means clustering algorithm uses a coreset construction based on the k-Means++ seeding procedure from Arthur and Vassilviskii [9]. One reason for this design decision was that the k-Means++ seeding works well for high-dimensional datasets, which is often required in practice. This nice property does not apply to many other clustering methods, like the grid-based methods from Har-Peled and Mazumdar [58] and Frahling and Sohler [44], for instance.

Let $P \subset \mathbb{R}^d$ be a set of points with size $|P| =: n$. For an arbitrary fixed parameter $m \in \mathbb{N}$, our coreset construction is as follows (see also Algorithm 6.2.1). First, we choose a set $S := \{q_1, q_2, \ldots, q_m\}$ of size m at random according to D^2. Let Q_i denote the set of points from P that are closest to q_i (breaking ties arbitrarily). By using weight function $w : S \to \mathbb{R}_{\geq 0}$ with $w(q_i) = |Q_i|$, we obtain the weighted set S as our coreset.

Note that our coreset construction is rather easy to implement and its running time has a merely linear dependency on the dimension d. Furthermore, empirical evaluation (as given in Section 6.5) suggests that our construction leads to good coresets even for relatively small choices of m (i.e., say $m = 200k$). Unfortunately, we do not have a formal proof supporting this observation. However, we are able to do a first step by proving that, at least in low-dimensional spaces, our construction indeed leads to small coresets.

Algorithm 6.2.1 AdaptiveCoreset(P, m)

```
choose an initial coreset point q_1 uniformly at random from P
w(q_1) ← 0
S ← {q_1}
for i ← 2 to m do
    choose q_i at random according to D^2 from P
    w(q_i) ← 0
    S ← S ∪ {q_i}
for each p ∈ P do
    let q_i ∈ S, 1 ≤ i ≤ m, be the nearest coreset point to p
    w(q_i) ← w(q_i) + 1
```

Our proof is based on Lemma 6.2.1. Intuitively, this lemma states that if we consider an optimal m-clustering of P, with m large enough, then the optimal m-clustering cost is merely a tiny fraction of the optimal k-clustering cost of P. Lemma 6.2.1 is a consequence of the fact that there exist (k, γ)-coresets of size $m \in (d/\gamma)^{\mathcal{O}(d)} k \log(n)$, which has already been proven by Har-Peled and Mazumdar [58].

Lemma 6.2.1. *Let γ, $0 < \gamma \leq 1$, and let $m \in \mathbb{N}$. If*

$$m \geq \left(\frac{16d}{\gamma}\right)^{d/2} \cdot k \cdot \lceil \log(n) + 3 \rceil \ ,$$

then we get

$$\text{Means}_m^*(P) \leq \gamma \cdot \text{Means}_k^*(P) \ .$$

Proof. Let $C := \{c_1, \ldots, c_k\}$ be an optimal solution to the Euclidean k-means clustering problem for P with $|P| =: n$, i.e., $\text{Means}(P, C) = \text{Means}_k^*(P)$. We consider an exponential grid around each center in C. The construction of this grid is essentially the same as the one from Har-Peled and Mazumdar [58]. In detail, the construction is defined as follows.

Let the average cost per point of an optimal k-clustering solution for P be denoted by

$$R := \frac{\text{Means}_k^*(P)}{n} \ .$$

Furthermore, for each $j \in \{0, 1, \ldots, \lceil \log(n) + 2 \rceil\}$ and each center $c_i \in C$, let $\mathcal{V}_{ij}$ be the axis-parallel square centered at c_i with side length

$$r_j := \sqrt{2^j R} \ .$$

Then, we recursively define $\mathcal{W}_{i0} := \mathcal{V}_{i0}$ and $\mathcal{W}_{ij} := \mathcal{V}_{ij} \backslash \mathcal{V}_{i,j-1}$ for $j \in \{1, 2, \ldots, \lceil \log(n) + 2 \rceil\}$. Obviously, each point in P is contained within a $\mathcal{W}_{ij}$ since otherwise there would be a point $p \in P$ with

$$\text{D}^2(p, C) \ > \ \left(\frac{r_{\lceil \log(n)+2 \rceil}}{2}\right)^2 \ = \ \frac{2^{\lceil \log(n)+2 \rceil} R}{4} \ \geq \ nR \ \geq \ \text{Means}_k^*(P) \ ,$$

which is a contradiction.

For each i, j individually, we partition $\mathcal{W}_{ij}$ into small grid cells with side length

$$r'_j := \sqrt{\frac{\gamma}{9d}} \cdot r_j = \sqrt{\frac{\gamma}{9d} \cdot 2^j R} \ .$$

We remark that the small grid cells do not have to fit properly in $\mathcal{W}_{ij}$. In fact, we impose a grid with side length r'_j on $\mathcal{W}_{ij}$ such that $\mathcal{W}_{ij}$ is completely covered. Then, the partition of $\mathcal{W}_{ij}$ consists of all the small cells that completely cover $\mathcal{W}_{ij}$ as well as all parts of the small cells that partly cover $\mathcal{W}_{ij}$. This partition is illustrated by Figure 6.1.

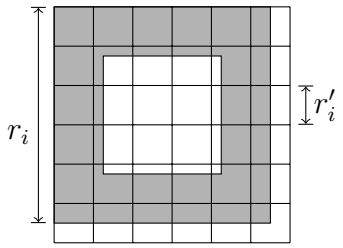


Figure 6.1: Illustration of the partition of $\mathcal{W}_{ij}$ into small grid cells. The area $\mathcal{W}_{ij}$ is colored in gray. The white parts of the small cells do not belong to the partition of $\mathcal{W}_{ij}$.

For each grid cell $\mathcal{C}$ such that $\mathcal{C} \cap \mathcal{W}_{ij}$ contains points from P, we select a single point from $P \cap \mathcal{C} \cap \mathcal{W}_{ij}$ as the representative of all the points in $P \cap \mathcal{C} \cap \mathcal{W}_{ij}$. Let G be the set of all these representatives. Since we have

$$\frac{r_i}{r'_i} = \sqrt{\frac{9d}{\gamma}} \geq 3 \ ,$$

there are at most

$$\begin{aligned} \sum_{c_i \in C} \sum_{j=0}^{\lceil \log(n)+2 \rceil} \left(\left\lceil \frac{r_j}{r'_j} \right\rceil \right)^d &\leq \sum_{c_i \in C} \sum_{j=0}^{\lceil \log(n)+2 \rceil} \left(\frac{4}{3} \cdot \frac{r_j}{r'_j} \right)^d \\ &= k \cdot \lceil \log(n) + 3 \rceil \cdot \left(\frac{16d}{\gamma} \right)^{d/2} \end{aligned}$$

grid cells. Since this number is smaller or equal to m, we obtain $|G| \leq m$.

Let g_p denote the representative of $p \in P$ in G. Then, we have

$$\text{Means}^*_m(P) \ \leq \ \text{Means}^*_{|G|}(P) \ \leq \ \text{Means}(P, G) \ \leq \ \sum_{p \in P} \text{D}^2\left(p, g_p\right) \ . \tag{6.1}$$

For each point $p \in P$, the distance from its representative g_p is upper bounded by the diagonal of the grid cell that contains p. Thus, for each $p \in \mathcal{W}_{i0}$, we have

$$\text{D}^2\left(p, g_p\right) \ \leq \ \left(\sqrt{d} \cdot r'_0\right)^2 \ \leq \ \frac{\gamma R}{9} \ . \tag{6.2}$$

Furthermore, for each $p \in \mathcal{W}_{ij}$ with $j \geq 1$, we know that c_i is the center of $\mathcal{V}_{i,j-1}$ and p is not contained in $\mathcal{V}_{i,j-1}$. It follows that

$$\mathrm{D}^2(p, C) \;\geq\; \left(\frac{r_{j-1}}{2}\right)^2 \;\geq\; 2^{j-3} R \;\; .$$

Hence, in this case, we get

$$\mathrm{D}^2\left(p, g_p\right) \;\leq\; \left(\sqrt{d} \cdot r'_j\right)^2 \;=\; \frac{\gamma}{9} \cdot 2^j \, R \;\leq\; \frac{8\gamma}{9} \cdot \mathrm{D}^2(p, C) \;\; . \tag{6.3}$$

Due to Inequalities (6.1)–(6.3) and the definition of R, we obtain

$$\begin{aligned}
\mathrm{Means}^*_m(P) \;&\leq\; \sum_{p \in P} \mathrm{D}^2\left(p, g_p\right) \\
&\leq\; \sum_{p \in P} \left(\frac{\gamma R}{9} + \frac{8\gamma}{9} \cdot \mathrm{D}^2(p, C)\right) \\
&=\; n \cdot \frac{\gamma}{9} R + \frac{8\gamma}{9} \sum_{p \in P} \mathrm{D}^2(p, C) \\
&=\; \frac{\gamma}{9} \cdot \mathrm{Means}^*_k(P) + \frac{8\gamma}{9} \cdot \mathrm{Means}^*_k(P) \\
&=\; \gamma \cdot \mathrm{Means}^*_k(P) \;\; .
\end{aligned}$$

□

Now, we go back to our coreset construction. Given the point set P and a parameter $m \in \mathbb{N}$, let S be our weighted coreset chosen at random according to D^2 from P by Algorithm 6.2.1. Furthermore, let C be an arbitrary set of k centers. For each point $p \in P$, we denote the point from S whose weight has been increased by 1 due to p in line 9 of Algorithm 6.2.1 by q_p, i.e., q_p is a point from S closest to p. Then, the difference between the cost of clustering P and the cost of clustering S is at most

$$\begin{aligned}
\left|\mathrm{Means}(P, C) - \mathrm{Means}(S, C)\right| \;&=\; \left|\sum_{p \in P} \mathrm{D}^2(p, C) - \sum_{p \in P} \mathrm{D}^2(q_p, C)\right| \\
&\leq\; \sum_{p \in P} \left|\mathrm{D}^2(p, C) - \mathrm{D}^2(q_p, C)\right| \;\; .
\end{aligned}$$

We partition P into two subsets P_{near} and P_{dist}. Roughly speaking, the set P_{near} contains each point $p \in P$ whose distance from its coreset point q_p is small compared to the distance from its nearest center in C. More precisely, for any constant ε with $0 < \varepsilon \leq 1$, we define

$$P_{\mathrm{near}} := \left\{p \in P \mid \mathrm{D}(p, q_p) \leq \varepsilon \, \mathrm{D}(p, C)\right\} \;\; .$$

The set P_{dist} contains all the other points from P, i.e.,

$$P_{\mathrm{dist}} := \left\{p \in P \mid \mathrm{D}(p, q_p) > \varepsilon \, \mathrm{D}(p, C)\right\} \;\; .$$

First, in Claim 6.2.2, we estimate the error of the clustering cost that occurs for any point in P_{near}. Then, in Claim 6.2.3, we give an estimation of the error for any point in P_{dist}.

Claim 6.2.2. *If $p \in P_{\text{near}}$, then*

$$\left|\mathrm{D}^2(p,C) - \mathrm{D}^2(q_p,C)\right| \leq 3\varepsilon\,\mathrm{D}^2(p,C) \ .$$

Proof. For the moment, let us assume that $\mathrm{D}(p,C) \leq \mathrm{D}(q_p,C)$. Let c_p denote the element from C closest to p. By triangle inequality and the definition of P_{near}, we have

$$\begin{aligned} \mathrm{D}(q_p,C) &\leq \mathrm{D}(q_p,c_p) \\ &\leq \mathrm{D}(p,c_p) + \mathrm{D}(p,q_p) \\ &\leq (1+\varepsilon)\cdot\mathrm{D}(p,C) \ . \end{aligned}$$

Hence, for the squared distances, we obtain

$$\begin{aligned} \mathrm{D}^2(q_p,C) &\leq (1+\varepsilon)^2\cdot\mathrm{D}^2(p,C) \\ &\leq (1+3\varepsilon)\cdot\mathrm{D}^2(p,C) \ . \end{aligned}$$

Thus, we get $\mathrm{D}^2(q_p,C) - \mathrm{D}^2(p,C) \leq 3\varepsilon\,\mathrm{D}^2(p,C)$, which proves the claim in the case $\mathrm{D}(p,C) \leq \mathrm{D}(q_p,C)$.

Now, assume that $\mathrm{D}(p,C) > \mathrm{D}(q_p,C)$. Let c_s denote the element from C closest to q_p. Again, by triangle inequality and the definition of P_{near}, we have

$$\begin{aligned} \mathrm{D}(p,C) &\leq \mathrm{D}(p,c_s) \\ &\leq \mathrm{D}(q_p,c_s) + \mathrm{D}(p,q_p) \\ &\leq \mathrm{D}(q_p,C) + \varepsilon\,\mathrm{D}(p,C) \ . \end{aligned}$$

It follows that $(1-\varepsilon)\cdot\mathrm{D}(p,C) \leq \mathrm{D}(q_p,C)$. For the squared distances, we obtain

$$\begin{aligned} \mathrm{D}^2(q_p,C) &\geq (1-\varepsilon)^2\cdot\mathrm{D}^2(p,C) \\ &> (1-2\varepsilon)\cdot\mathrm{D}^2(p,C) \ . \end{aligned}$$

Hence, we get

$$\begin{aligned} \mathrm{D}^2(p,C) - \mathrm{D}^2(q_p,C) &< 2\varepsilon\,\mathrm{D}^2(p,C) \\ &< 3\varepsilon\,\mathrm{D}^2(p,C) \ , \end{aligned}$$

which proves the claim in the case $\mathrm{D}(p,C) > \mathrm{D}(q_p,C)$. □

Claim 6.2.3. *If $p \in P_{\text{dist}}$, then*

$$\left|\mathrm{D}^2(p,C) - \mathrm{D}^2(q_p,C)\right| \leq \frac{3}{\varepsilon}\,\mathrm{D}^2(p,q_p) \ .$$

Proof. Let c_p denote the element from C closest to p, and let c_s denote the element from C closest to q_p. By triangle inequality, we have

$$\begin{aligned} \mathrm{D}(p,C) &\leq \mathrm{D}(p,c_s) \\ &\leq \mathrm{D}(p,q_p)+\mathrm{D}(q_p,c_s) \\ &= \mathrm{D}(p,q_p)+\mathrm{D}(q_p,C) \ . \end{aligned}$$

Similarly, we get

$$\begin{aligned} \mathrm{D}(q_p,C) &\leq \mathrm{D}(q_p,c_p) \\ &\leq \mathrm{D}(p,q_p)+\mathrm{D}(p,c_p) \\ &= \mathrm{D}(p,q_p)+\mathrm{D}(p,C) \ . \end{aligned}$$

It follows that $|\mathrm{D}(p,C)-\mathrm{D}(q_p,C)| \leq \mathrm{D}(p,q_p)$ and $\mathrm{D}(p,C)+\mathrm{D}(q_p,C) \leq 2\,\mathrm{D}(p,C)+\mathrm{D}(p,q_p)$. Since $\mathrm{D}(p,q_p) > \varepsilon\,\mathrm{D}(p,C)$ and $\varepsilon \leq 1$, we get

$$\begin{aligned} \left|\mathrm{D}^2(p,C)-\mathrm{D}^2(q_p,C)\right| &= |\mathrm{D}(p,C)-\mathrm{D}(q_p,C)| \cdot (\mathrm{D}(p,C)+\mathrm{D}(q_p,C)) \\ &\leq \mathrm{D}(p,q_p) \cdot (2\,\mathrm{D}(p,C)+\mathrm{D}(p,q_p)) \\ &\leq \left(\frac{2}{\varepsilon}+1\right)\mathrm{D}^2(p,q_p) \\ &\leq \frac{3}{\varepsilon}\,\mathrm{D}^2(p,q_p) \ . \end{aligned}$$

□

Now, we can show our main result.

Theorem 6. *Let $k \in \mathbb{N}$, let ε, $0 < \varepsilon \leq 1$, be a precision parameter, and let δ, $0 < \delta < 1$, be an error probability. Given a point set $P \subset \mathbb{R}^d$ of size $|P| =: n$ and a size parameter*

$$m \in \left(\frac{d}{\delta\varepsilon}\right)^{\mathcal{O}(d)} \cdot k \cdot \log(n) \cdot \log^{d/2}\left(\frac{k\log(n)}{\delta\varepsilon}\right) \ ,$$

algorithm AdaptiveCoreset *computes a weighted multiset S with size m that is a $(k, 6\varepsilon)$-coreset of P with probability at least $1-\delta$.*

Proof. Due to Claims 6.2.2 and 6.2.3, we have

$$\begin{aligned} & |\mathrm{Means}(P,C)-\mathrm{Means}(S,C)| \\ \leq\ & \sum_{p\in P}\left|\mathrm{D}^2(p,C)-\mathrm{D}^2(q_p,C)\right| \\ \leq\ & \sum_{p\in P_{\mathrm{near}}}\left|\mathrm{D}^2(p,C)-\mathrm{D}^2(q_p,C)\right| + \sum_{p\in P_{\mathrm{dist}}}\left|\mathrm{D}^2(p,C)-\mathrm{D}^2(q_p,C)\right| \\ \leq\ & 3\varepsilon \sum_{p\in P_{\mathrm{near}}}\mathrm{D}^2(p,C) + \frac{3}{\varepsilon}\sum_{p\in P_{\mathrm{dist}}}\mathrm{D}^2(p,q_p) \\ \leq\ & 3\varepsilon \cdot \mathrm{Means}(P,C) + \frac{3}{\varepsilon}\cdot\mathrm{Means}(P,S) \ . \end{aligned}$$

Due to Lemma 6.1.3 and Markov's inequality, we obtain

$$\text{Means}(P,S) \leq \frac{8}{\delta}\left(2+\ln(m)\right)\cdot \text{Means}_m^*(P)$$

with probability at least $1-\delta$. Hence, by using Lemma 6.2.1 with

$$\gamma := \frac{\varepsilon^2\delta}{8(2+\ln m)} \ ,$$

we have

$$\begin{aligned}
\text{Means}(P,S) &\leq \frac{8}{\delta}\left(2+\ln(m)\right)\cdot \text{Means}_m^*(P)\\
&\leq \frac{8}{\delta}\left(2+\ln(m)\right)\cdot \gamma \cdot \text{Means}_k^*(P)\\
&\leq \varepsilon^2\,\text{Means}_k^*(P)\\
&\leq \varepsilon^2\ \text{Means}(P,C)
\end{aligned}$$

and, thus, $|\text{Means}(P,C)-\text{Means}(S,C)| \leq 6\varepsilon\cdot\text{Means}(P,C)$ with probability $1-\delta$, provided that the coreset size m satisfies the condition

$$m \geq \left(\frac{16d}{\gamma}\right)^{d/2}\cdot k\cdot\lceil\log(n)+3\rceil \ . \tag{6.4}$$

Hence, the only thing left to do is to prove that there exists a coreset size

$$m \in \left(\frac{d}{\delta\varepsilon}\right)^{\mathcal{O}(d)}\cdot k\cdot\log(n)\cdot\log^{d/2}\left(\frac{k\log(n)}{\delta\varepsilon}\right) \tag{6.5}$$

that satisfies Inequality (6.4). Since we can assume that $n\geq 16$ and $m\geq 8$, Inequality (6.4) is satisfied if we have

$$\frac{m}{\log^{d/2}(m)} \geq \frac{2\cdot 16^d\, d^{d/2}\, k\log(n)}{\delta^{d/2}\varepsilon^d} \ . \tag{6.6}$$

We conclude that Condition (6.5) and Inequality (6.6) are both satisfied for a choice of

$$m = (2d)^{d/2}\cdot\frac{2\cdot 16^d\, d^{d/2}\, k\log(n)}{\delta^{d/2}\varepsilon^d}\cdot\log^{d/2}\left(\frac{2\cdot 16^d\, d^{d/2}\, k\log(n)}{\delta^{d/2}\varepsilon^d}\right)$$

since we have

$$\begin{aligned}
\log^{d/2}(m) &= \log^{d/2}\left((2d)^{d/2}\cdot\frac{2\cdot 16^d\, d^{d/2}\, k\log(n)}{\delta^{d/2}\varepsilon^d}\cdot\log^{d/2}\left(\frac{2\cdot 16^d\, d^{d/2}\, k\log(n)}{\delta^{d/2}\varepsilon^d}\right)\right)\\
&\leq \left(\frac{d}{2}\right)^{d/2}\cdot\log^{d/2}\left(2d\cdot\frac{2\cdot 16^d\, d^{d/2}\, k\log(n)}{\delta^{d/2}\varepsilon^d}\cdot\log\left(\frac{2\cdot 16^d\, d^{d/2}\, k\log(n)}{\delta^{d/2}\varepsilon^d}\right)\right)\\
&\leq \left(\frac{d}{2}\right)^{d/2}\cdot\log^{d/2}\left(\left(\frac{2\cdot 16^d\, d^{d/2}\, k\log(n)}{\delta^{d/2}\varepsilon^d}\right)^4\right)\\
&= (2d)^{d/2}\cdot\log^{d/2}\left(\frac{2\cdot 16^d\, d^{d/2}\, k\log(n)}{\delta^{d/2}\varepsilon^d}\right) \ .
\end{aligned}$$

□

Please note that the size bound on the number of coreset points m from Theorem 6 is merely a sufficient condition, and that, to the best of our knowledge, there is no reason to assume that this size bound is tight. Hence, in compliance with our experiments, the actual dependency of m on the dimension d may as well be better than is suggested by the theorem.

6.3 The Coreset Tree

Unfortunately, there is one practical problem concerning the k-MEANS++ seeding procedure. Assume that we have chosen a sample set $S = \{q_1, q_2, \ldots, q_i\}$ from the input set $P \subseteq \mathbb{R}^d$ so far, where $i < m$ and $|P| =: n$. In order to compute the probabilities to choose the next sample point q_{i+1}, we need to determine the squared distance from each point in P to its nearest neighbor in S. Hence, using a standard implementation of such a computation, we require $\Theta(dnm)$ time to obtain all m coreset points, which is too slow for larger values of m. For this reason, we propose a new data structure called *coreset tree*, which speeds up this computation. Roughly speaking, a coreset tree is a hierarchical decomposition of P where each leaf represents a set of this decomposition. The advantage of the coreset tree is that it allows us to compute subsequent sample points by taking only points from a subset of P into account whose size is significantly smaller than n. We obtain that if the constructed coreset tree is balanced (i.e., the tree is of depth $\Theta(\log(m))$ and each leaf represents roughly the same number of points), we merely need $\Theta(dn \log(m))$ time to compute all m coreset points. This intuition is supported by our empirical evaluation on real-world datasets, where we find that the process of sampling according to D^2 is significantly sped up while the resulting sample set S has essentially the same properties as the original k-MEANS++ seed.

In the following, we will explain the construction of the coreset tree in more detail. A description in pseudocode is given by Algorithm 6.3.1.

6.3.1 Definition of the Coreset Tree

A coreset tree T for a point set P is a binary tree that is associated with a hierarchical divisive clustering for P: One starts with a single cluster that contains the whole point set P and successively partitions existing clusters into two subclusters such that the points in one subcluster are far from the points in the other subcluster. The division step is repeated until the number of clusters corresponds to the desired number of clusters. Associated with this procedure, the coreset tree T has to satisfies the following properties:

(i) Each node of T is associated with a cluster in the hierarchical divisive clustering.

(ii) The root of T is associated with the single cluster that contains the whole point set P.

(iii) The nodes associated with the two subclusters of a cluster C are the child nodes of the node associated with C.

With each node v of T, we store the following attributes: A point set P_v, a representative point q_v from P_v, and a value weight(v). Here, point set P_v is the cluster associated with node v. Note that this attribute has only to be stored explicitly in the leaf nodes of T, while, for an internal node v, the set P_v is implicitly defined by the union of the point sets of its children. The representative point q_v of a node v is obtained by using the technique of non-uniform sampling according to D^2. At any time, the set of all the points q_ℓ stored at a leaf node ℓ are the points that have been chosen so far to be points of the eventual coreset. Furthermore, for a leaf node ℓ, the attribute weight(ℓ) equals Means(P_ℓ, q_ℓ), which is the sum of squared distances over all points in P_ℓ to q_ℓ. The value weight(v) of an internal node v is defined as the sum of the weights of its children.

6.3.2 Construction of the Coreset Tree

For sake of simplicity, at any time, we number the leaf nodes of the current coreset tree consecutively starting with 1. At the beginning, T consists of one node, the root, which is given the number 1 and is associated with the whole point set P. The attribute q_1 of the root is our first point in S and is obtained by sampling one point uniformly at random from P. Now, let us assume that our current tree has i leaf nodes $1, 2, \ldots, i$, the corresponding sample points are $q_1, q_2, \ldots, q_i$, and $P_1, P_2, \ldots, P_i$ are the associated clusters. We obtain the next sample point q_{i+1}, new clusters in our hierarchical divisive clustering, and, thus, new nodes in T by performing the following three steps:

1. Choose a leaf node ℓ at random, where the probability of each leaf node ℓ' is proportional to cost$(P_{\ell'}, q_{\ell'})$.

2. Choose a new sample point, denoted by q_{i+1}, from the subset P_ℓ at random according to D^2.

3. Based on q_ℓ and q_{i+1}, split P_ℓ into two subclusters and create two child nodes of ℓ in T.

The first step is implemented as follows: Starting at the root of T, let u be the current inner node. Then, we select randomly a child node of u, where the probability distribution for the child nodes of u is given by their associated weights. More precisely, each child node v of the current node u is chosen with probability weight(v)/ weight(u). We continue this selection process until we reach a leaf node. Let ℓ be the selected leaf node, let q_ℓ be the sample point stored at ℓ, and let P_ℓ be the subset of P associated with ℓ. It is easy to check that, in doing so, we have chosen ℓ among the leaf nodes with probability cost$(P_\ell, q_\ell)/\sum_{j=1}^{i}$ cost(P_j, q_j).

In the second step, we choose the new sample point q_{i+1} from P_ℓ at random according to D^2, i.e., each $p \in P_\ell$ is chosen with probability $D^2(p, q_\ell)/$ cost(P_ℓ, q_ℓ). In doing so, each point in P is sampled with a probability that is proportional to its squared distance to its center in the clustering induced by the partition of the leaf nodes (giving the clusters) and their sample points (being the centers). That is, we use the same distribution as the

Algorithm 6.3.1 TreeCoreset(P, m)

1: choose q_1 uniformly at random from P
2: $root \leftarrow$ node with $q_{root} \leftarrow q_1$ and weight$(root) \leftarrow$ Means(P, q_1)
3: $S \leftarrow \{q_1\}$
4: **for** $i \leftarrow 2$ **to** m **do**
5: start at $root$, iteratively select one of the two child nodes at random according to their weights until a leaf ℓ is chosen
6: choose q_i according to D^2 from P_ℓ
7: $S \leftarrow S \cup \{q_i\}$
8: create two child nodes ℓ_1, ℓ_2 of ℓ and update weight(ℓ)
9: propagate update of weight attribute upwards up to node $root$

k-Means++ seeding does with the exception that the probability of choosing a point $p \in P_j$ is proportional to $\mathrm{D}^2(p, q_j)$ rather than proportional to $\mathrm{D}^2(p, \{q_1, \ldots, q_i\})$.

In the third step, we create two child nodes ℓ_1 and ℓ_2 of ℓ and compute the associated partition of P_ℓ as well as the corresponding attributes. We store at node ℓ_1 the point q_ℓ and at node ℓ_2 our new sample point q_{i+1}. Based on these two representative points, we partition P_ℓ into two subsets P_{ℓ_1} and P_{ℓ_2}. The set P_{ℓ_1} contains all the points from P_ℓ which are closer to q_ℓ than to q_{i+1}, i.e.,

$$P_{\ell_1} = \{p \in P_\ell \mid \mathrm{D}(p, q_\ell) < \mathrm{D}(p, q_{i+1})\} \ .$$

The set P_{ℓ_2} contains all the remaining points from P_ℓ, i.e.,

$$P_{\ell_2} = \{p \in P_\ell \mid \mathrm{D}(p, q_{i+1}) \leq \mathrm{D}(p, q_\ell)\} \ .$$

The node ℓ_1 is associated with the set P_{ℓ_1}, and ℓ_2 is associated with the set P_{ℓ_2}. We determine the weight attribute for the nodes ℓ_1 and ℓ_2 as described above. Recall here that the weight attribute of an inner node of T is defined as the sum of the weights of its child nodes. Consequently, we update the weight of the parent node ℓ of ℓ_1 and ℓ_2 according to this. Afterwards, this update is propagated upwards, until we reach the root of the tree.

6.3.3 Extraction of the Coreset

As soon as the coreset tree T has m leaf nodes, we can construct our coreset. Let $q_1, q_2, \ldots, q_m$ be the representative points stored at the leaf nodes of T. Furthermore, let Q_i denote the set of points from P which are closest to q_i (breaking ties arbitrarily). Then, we obtain the coreset $S = \{q_1, q_2, \ldots, q_m\}$ where the weight of q_i is given by the number of points in Q_i.

6.4 Streaming Algorithm

In this section, we describe our clustering algorithm for data streams. To this end, let m be a fixed size parameter. First, we extract a small coreset of size m from the data stream by

Algorithm 6.3.2 InsertPoint(p)

```
1: put p into B_0
2: if B_0 is full then
3:    create empty bucket S
4:    move points from B_0 to S
5:    empty B_0
6:    i ← 1
7:    while B_i is not empty do
8:       create coreset from the union of B_i and S
9:       store coreset in S
10:      empty B_i
11:      i ← i + 1
12:   move points from S to B_i
```

using the merge-and-reduce technique from Har-Peled and Mazumdar [58], which is based on the theory of decomposable search problems of Bentley and Saxe [16]. This streaming method is described in detail in the section below. For the reduce step, we employ our new coreset construction, using the coreset trees as given in Section 6.3.

Afterwards, a k-clustering can be obtained at any time by running any k-means algorithm on the coreset. Note that since the size of the coreset is much smaller than the size of the data stream, it is no longer inefficient to use algorithms that require random access on their input data. In our implementation, we run the k-Means++ algorithm from Arthur and Vassilvitskii [9] on our coreset five times independently and choose the best clustering result obtained this way. We call the resulting algorithm StreamKM++.

6.4.1 The Merge-and-Reduce Technique

In order to maintain a small coreset for all points in the data stream, we use the merge-and-reduce method [16, 58]. For a data stream containing n points, the algorithm maintains $L := \lceil \log(n/m) + 2 \rceil$ buckets $B_0, B_1, \ldots, B_{L-1}$. Bucket B_0 can store any number between 0 and m points. For $i \geq 1$, bucket B_i is either empty or contains exactly m points. The idea of this approach is that, at any time, if bucket B_i is full, it contains a coreset of size m representing $2^{i-1}m$ points from the data stream.

New points from the data stream are always inserted into the first bucket B_0. If bucket B_0 is full (i.e., contains m points), all points from B_0 need to be moved to bucket B_1. If bucket B_1 is empty, we are done. However, if bucket B_1 already contains m points, we compute a new coreset S of size m from the union of the $2m$ points stored in B_0 and B_1 by using the coreset construction described above. Now, both buckets B_0 and B_1 are emptied and the m points from coreset S need to be moved into bucket B_2. If bucket B_2 is full, we repeat the process with S and B_2. Overall, the process is repeated iteratively until we find the first empty bucket in which we can move the coreset S. A description in pseudocode for inserting a point from the data stream into the buckets is given by Algorithm 6.3.2.

At any time, it is possible to compute a coreset of size m for all the points in the data stream that we have seen so far. For this purpose, we compute a coreset from the union of the at most $m\lceil\log(n/m)+2\rceil$ weighted coreset points stored in all the buckets $B_0, B_1, \ldots, B_{L-1}$ by using the coreset tree construction. In this way, we obtain the desired coreset of size m.

Note that the coreset tree construction can be easily generalized to input points with integer weights. Therefore, each time when we choose a new coreset point, we compute the probabilities of the points according to D^2, as described before, and then multiply each probability with the weight of the appropriate point. We also incorporate the point weights when we compute the weight attribute of a new leaf node. These two adaptations can be thought of as replacing each weighted point by multiple copies of the same point each having weight 1.

6.4.2 Complexity

Using our implementation, a single merge-and-reduce step is guaranteed to be executed in time $\mathcal{O}(dm^2)$ (or even in time $\Theta(dm\log(m))$ if we assume the used coreset tree to be balanced). For a stream of n points, $\lceil n/m\rceil$ such steps are needed. The amortized running time of all merge-and-reduce steps is at most $\mathcal{O}(dnm)$. The final merge-and-reduce step, to obtain a coreset of size m for the union of all buckets, can be done in time $\mathcal{O}(dm^2\log(n/m))$. Finally, algorithm k-MEANS++ is executed five times on an input set of size m, requiring time $\Theta(dkm)$ per iteration. Summing up, the total running time of algorithm STREAMKM++ is $\mathcal{O}(dnm)$, and the amortized processing time per data item is $\mathcal{O}(dm)$. Obviously, algorithm STREAMKM++ needs at most $\Theta(dm\log(n/m))$ memory units. Hence, both the processing time and the space requirement have a low dependency on the dimension d. As a result, our approach is suitable for high-dimensional data.

Of course, careful consideration has to be given to the choice of the coreset size parameter m. Our experiments show that a choice of $m = 200k$ is sufficient for a good clustering quality without sacrificing too much running time.

6.5 Empirical Evaluation

We conducted several experiments on different datasets to evaluate the quality of algorithm STREAMKM++.[1] A description of the datasets can be found in the next section. The computation on the biggest dataset, which is denoted by *BigCross*, was performed on a DELL Optiplex 620 machine with 3 GHz Pentium D CPU and 2 GB main memory, using Linux 2.6.9 kernel. For all remaining datasets, the computation was performed on a DELL Optiplex 620 machine with 3 GHz Pentium D CPU and 4 GB main memory, using Linux 2.6.18 kernel.

[1] The source code, the documentation, and the datasets of our experiments can be found at `http://www.cs.upb.de/en/fachgebiete/ag-bloemer/research/clustering/streamkmpp/`

We compared algorithm StreamKM++ with two frequently used clustering algorithms for processing data streams, namely with algorithm BIRCH [111] and with algorithm StreamLS [96, 52]. On the smaller datasets, we also compared our algorithm with a vanilla implementation of Lloyd's algorithm [80], using initial seeds either uniformly at random (algorithm k-Means) or according to the non-uniform seeding from Arthur and Vassilvitskii [9] (algorithm k-Means++). All algorithms were compiled using `g++` from the GNU Compiler Collection on optimization level 2. The quality measure for all experiments was the sum of squared distances, to be referred as the cost of the clustering.

6.5.1 Datasets

Since synthetic datasets are typically easy to cluster, we focused our experiments on real-world datasets to obtain practically relevant results. Our main source was the UCI Machine Learning Repository [13]. In the following, we will give a brief description of all the datasets used in our empirical evaluation.

Spambase[2] is a dataset that contains data about spam e-mails and non-spam e-mails, including work and personal e-mails. Each data entry is a vector consisting of frequencies of certain words or characters occurring in the message and a class attribute that denotes whether the corresponding e-mail was considered as spam or not. After removing the classification attribute, 4 601 data points in 57 dimensions remained.

Intrusion[2,3] comprises data about TCP transmissions in a simulated environment. This simulation included different types of network attacks and intrusion attempts as well as normal network traffic. We used a 10% subset of the whole unlabeled dataset[4] and excluded all symbolic features. Eventually, 311 078 data points in 34 dimensions remained.

Covertype[2,5] contains cartographic data about some wilderness areas inside the Roosevelt National Forest of northern Colorado. The leading thought of analyzing this dataset is to be able to predict the forest cover type of specific regions from cartographic variables, which is a classification task. After removing the classification attribute, 581 012 data points in 54 dimensions remained.

The *Tower*[6] dataset consists of the RGB values of a 2 560 by 1 920 pixel image file. All 4 915 200 pixels are mapped into a 3-dimensional space of integer values between 0 and 255, representing the colors used in the image. Note that clustering techniques are frequently used for lossy image compression: Individual colors can be substituted with their corresponding cluster center.

The *Census 1990*[2] dataset consists of a one percent sample of the Public Use Microdata Samples (PUMS) person records, sampled from the full 1990 census set contributed by

[2]The dataset was contributed by the UCI Machine Learning Repository [13].
[3]The *Intrusion* dataset is part of the kddcup99 dataset.
[4]Available for free download at `http://kdd.ics.uci.edu/databases/kddcup99/kddcup.newtestdata_10_percent_unlabeled.gz`
[5]Copyright by Jock A. Blackard, Colorado State University
[6]The *Tower* dataset was contributed by Gereon Frahling and is available for free download at: `http://homepages.uni-paderborn.de/frahling/coremeans.html`

	data points	dimension	type
Spambase	4 601	57	float
Intrusion	311 078	34	int, float
Covertype	581 012	54	int
Tower	4 915 200	3	int
Census 1990	2 458 285	68	int
BigCross	11 620 300	57	int
Normdata	100 000	15	float

Table 6.1: Overview of the datasets

the U.S. Department of Commerce Census Bureau. Most of the data is citizen-related information, like personal income or age, for instance. The dataset has 2 458 285 data points in 68 dimensions. To our knowledge, it is one of the largest naturally structured and free accessible datasets available.

To run our algorithm on really huge datasets, we created the Cartesian product of the *Tower* and *Covertype* dataset. In this way, we got a naturally structured dataset that is large enough to test our algorithm's ability of handling huge amounts of data. We used a 1.5 GB sized subset of the Cartesian product consisting of 11 620 300 data points with 57 attributes, which we refer to as the *BigCross* dataset.

To evaluate the impact of the number of well separated clusters of a dataset, we also considered a number of synthetic datasets, to which we collectively refer as the *Normdata* datasets. To generate these datasets, we used essentially the same construction that has already been used in [9] to evaluate the k-MEANS++ algorithm. More precisely, for different values of k, we chose k 'true' centers uniformly at random from a 15-dimensional hypercube of side length 100. We then chose randomly points from a uniform mixture of 15-dimensional normal distributions of variance 1 around these center points. In this way, we obtained k well separated clusters. Each *Normdata* dataset consists of 100 000 points.

The size and the dimensionality of the datasets are summarized in Table 6.1.

6.5.2 Parameters of the Algorithms

For algorithm BIRCH, we set all parameters of the experimental environment, except for the memory settings, as recommended by the authors of BIRCH. Like Guha et al. [52], we observed that the CF-Tree had less leaves than it was allowed to use. The CF-Tree is the data structure that is used to compute the pre-clustering into the so-called clustering features (see also Section 6.1). The more leaves it has, the finer is the pre-clustering. Therefore, from time to time, BIRCH did not produce the correct number of centers, especially when the number of clusters k was high. For this reason, the memory settings had to be manually adjusted for each individual dataset. The complete list of parameters is given in Tables A.1 and A.2 in Appendix A.1.

For algorithm StreamKM++, we experimentally determined an appropriate coreset size m as a function of k. For obvious reasons, we need to choose $m \geq k$. To estimate an m that is sufficient to obtain good approximation results, we ran several experiments for different values of k and m on the datasets *Covertype* and *Tower*. Due to the randomized[7] nature of StreamKM++, we conducted ten runs for each combination of k and m. Figure 6.2 shows the average running times and cost of the clusterings. Concerning the cost, we observed that, for coreset sizes that are only marginally larger than k, the quality of a clustering can be improved considerably by increasing the coreset size. In contrast to that, for coreset sizes of, say, $m = 100k$ or more, the quality improves only slightly with increasing coreset size. For instance, the cost of a 50-clustering of either dataset computed on 20 000 coreset points is only marginally smaller than the cost of a 50-clustering computed on 10 000 coreset points. However, with respect to the running time, we observed that the growth of the running time depends roughly linear on the coreset size. Overall, we conclude the following. On the one hand, m should be chosen not too small (e.g., a very small multiple of k) because, for these values of m, the quality of a clustering can be easily improved, without sacrificing too much running time. On the other hand, m should not be chosen too large (e.g., a large multiple of k) because the increase in quality is only very small compared to clusterings for smaller coresets, but the running time is significantly higher. Therefore, we assume that our choice of $m = 200k$ provides a good trade-off for arbitrary datasets. However, smaller sizes such as $m = 20k$ or $m = 50k$ might still be sufficient to obtain very good clustering results on datasets with k well separated clusters.

For algorithm StreamLS the size of the data chunks is set equal to the coreset size m of algorithm StreamKM++. This is done to enable a fair comparison of both algorithms by allowing the same memory usage. We have to point out that, due to its nature, algorithm StreamLS does not always compute the prespecified number of cluster centers. In such a case, the difference varies from dataset to dataset and usually lies within a 20% margin from the prespecified number.

6.5.3 Comparison of the Algorithms

Comparison with BIRCH and StreamLS

To compare StreamKM++ with BIRCH and StreamLS, we conducted several experiments for different values of k on the four larger real-world datasets, i.e., the datasets *Covertype*, *Tower*, *Census 1990*, and *BigCross*. In each of these experiments, we set $m = 200k$. For the randomized algorithms StreamKM++ and StreamLS, ten experiments were conducted for each fixed k. For BIRCH, a single run was used since it is a deterministic algorithm. The average running times and cost of the clusterings are summarized in Figures 6.3 and 6.4. The interested reader can find the concrete values of all experiments in Appendices A.2 and A.3.

In our experiments, algorithm BIRCH had the best running time of all algorithms. However, this comes at the expense of a high k-means clustering cost. In terms of the sum

[7] We used the Mersenne Twister PRNG [85].

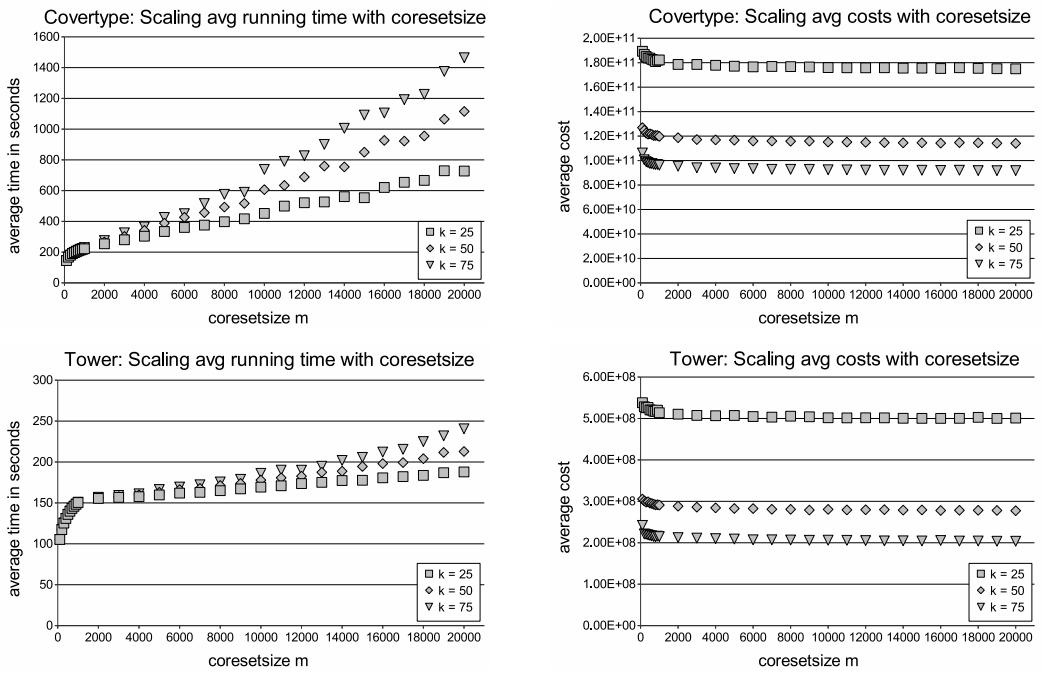


Figure 6.2: Experimental results for different coreset sizes

of squared distances, algorithms StreamKM++ and StreamLS outperform BIRCH by a factor of up to 2. Furthermore, as already mentioned, one drawback of algorithm BIRCH is the need of adjusting parameters manually to obtain a clustering with the desired number of centers.

By comparing StreamKM++ and StreamLS, we observed that the quality of the clusterings were on a par. More precisely, the absolute value of the cost of both algorithms lies within a $\pm 5\%$ margin from each other. In contrast to algorithm StreamLS, the number of centers computed by our algorithm always equals its prespecified value. Hence, the cost of clusterings computed by algorithm StreamKM++ tends to be more stable than the costs computed by algorithm StreamLS. The standard deviations of the running times and clustering cost for $k = 20$ are given in Tables 6.2 and 6.3. A complete overview for all experiments can be found in Appendix A.4.

In terms of running time, it turns out that our algorithm scales much better with increasing number of centers than algorithm StreamLS does. While for about $k \leq 10$ centers StreamLS is sometimes faster than our algorithm, for a larger number of centers, our algorithm easily outperforms StreamLS. For instance, on the dataset *Tower*, StreamKM++ computes a clustering with $k = 100$ centers in about 3% of the running time of StreamLS.

To investigate the impact of the number of clusters on the running time further, we conducted experiments on the synthetic datasets *Normdata* for different values of k and m. As described before, for both StreamKM++ and StreamLS, ten experiments were conducted for each combination of k and m. The average running times of the clusterings

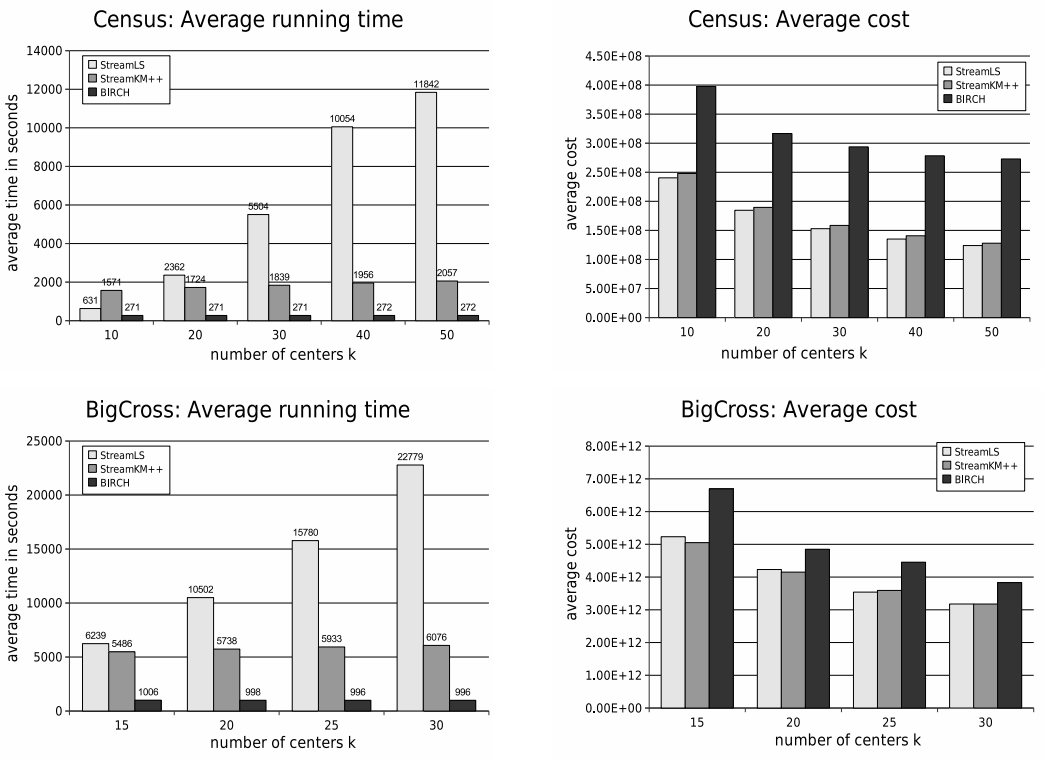


Figure 6.3: Experimental results for the datasets *Census 1990* and *BigCross*

$k = 20$	running time (in sec)		
	StreamKM++	StreamLS	k-Means++
Spambase	1.09	-	3.88
Intrusion	3.22	-	98.11
Covertype	6.93	18.18	1249.18
Tower	0.58	14.11	1594.76
Census 1990	5.16	54.30	-
BigCross	11.49	162.44	-

Table 6.2: Standard deviation of the running time for $k = 20$

are shown in Figure 6.5. Note that we omitted a figure presenting the average cost of the clusterings because both StreamKM++ and StreamLS always found an optimal or near optimal clustering. The interested reader can find the average values as well as the standard deviations for both running times and cost of the clusterings in the appendix. Figure 6.5 reveals the difference between the running times of StreamKM++ and StreamLS. The ratio between the running time needed by StreamKM++ and the running time needed by StreamLS is decreasing with increasing number of clusters. For $m = 500$, StreamKM++ computed the clusterings for $k = 100$ in about 8% of the running time of StreamLS and, for $k = 200$, it finished the clustering in about 2% of the running time of StreamLS.

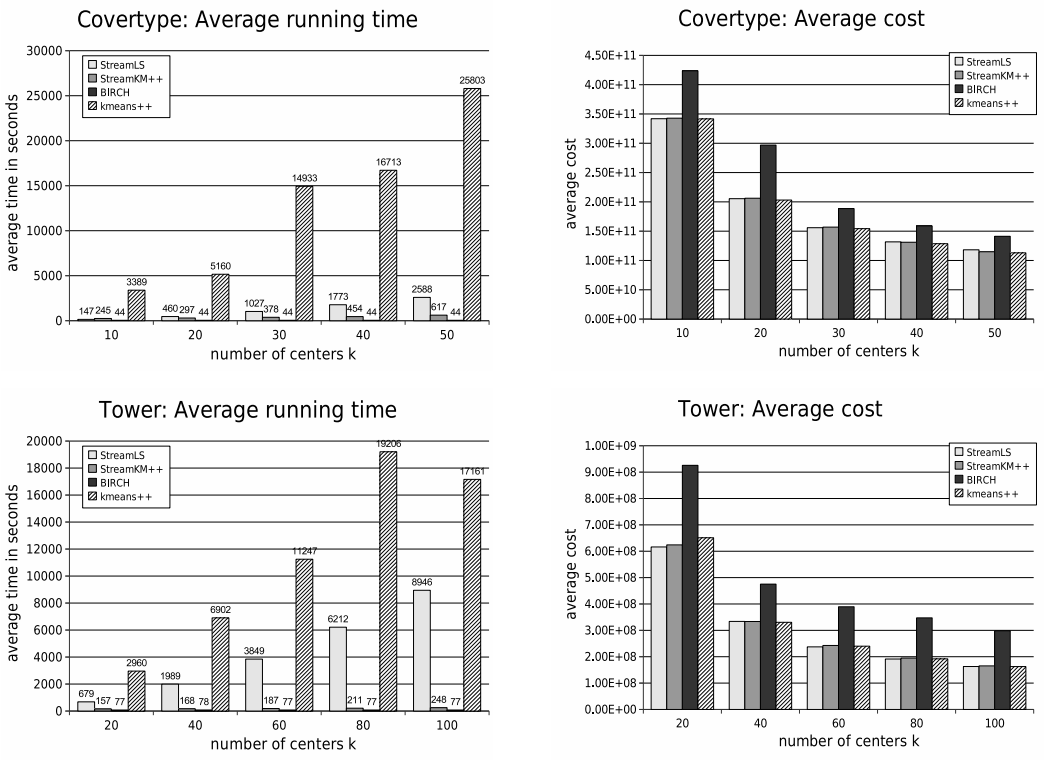


Figure 6.4: Experimental results for the datasets *Covertype* and *Tower*

$k = 20$	cost		
	StreamKM++	StreamLS	k-Means++
Spambase	$6.49 \cdot 10^5$	-	$1.73 \cdot 10^6$
Intrusion	$8.54 \cdot 10^{10}$	-	$3.70 \cdot 10^{11}$
Covertype	$1.08 \cdot 10^9$	$1.03 \cdot 10^{10}$	$9.17 \cdot 10^8$
Tower	$7.31 \cdot 10^6$	$2.71 \cdot 10^7$	$4.39 \cdot 10^7$
Census 1990	$3.66 \cdot 10^6$	$3.14 \cdot 10^6$	-
BigCross	$2.46 \cdot 10^{10}$	$3.36 \cdot 10^{11}$	-

Table 6.3: Standard deviation of the cost for $k = 20$

For $m = 1\,000$, StreamKM++ computed the clusterings for $k = 100$ in about 38% of the running time of StreamLS, whereas, for $k = 200$, it needed about 3% of the running time of StreamLS.

Overall, we conclude that, if the first priority is the quality of the clustering, then our algorithm provides a good alternative to BIRCH and StreamLS. This applies particularly if the number of cluster centers is large.

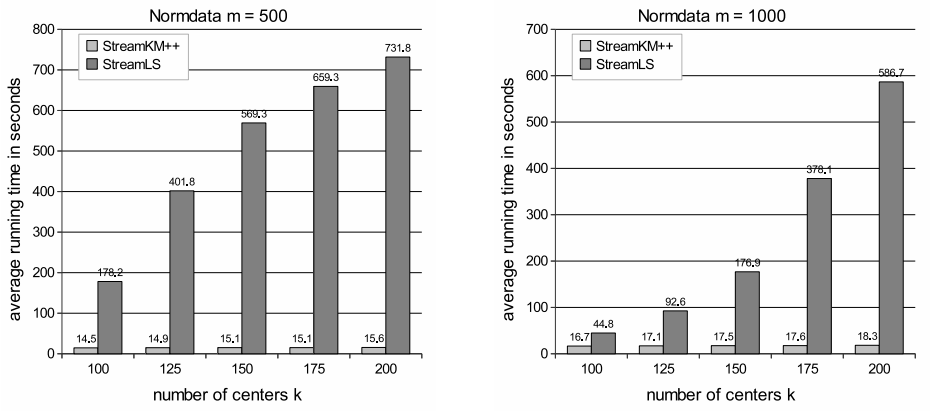


Figure 6.5: Experimental results for the *Normdata* datasets

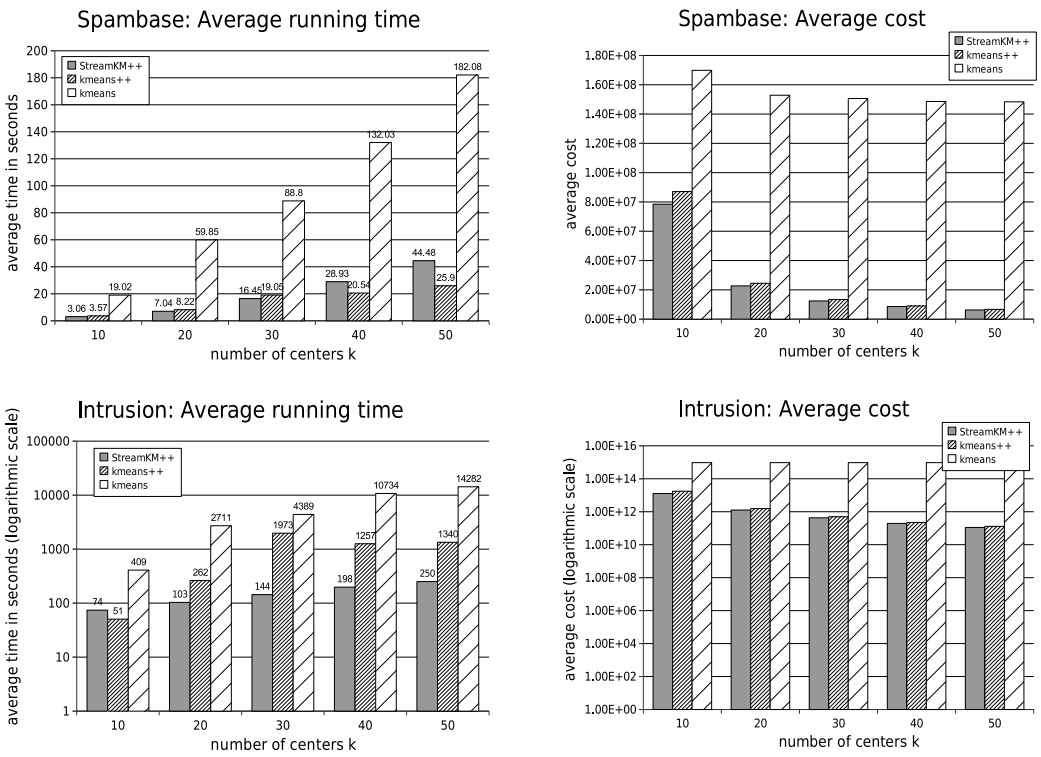


Figure 6.6: Experimental results for the datasets *Spambase* and *Intrusion*

Comparison with k-Means and k-Means++

We also compared the quality of StreamKM++ with the quality of classical non-streaming k-means algorithms. Because of their popularity, we have chosen k-Means and k-Means++ as competitors. These algorithms are designed to work in a classical non-streaming setting and, due to their need for random access on the data, are not suited for larger datasets. For this reason, we have run k-Means only on the two smallest datasets *Spambase* and

Intrusion, while k-MEANS++ has been evaluated only on the four smaller datasets *Covertype*, *Tower*, *Spambase*, and *Intrusion*. For each fixed k, we conducted ten experiments. The results of these experiments are summarized in Figures 6.6 and 6.4. Please note that the results for the dataset *Intrusion* are on a logarithmic scale. The concrete values of all experiments can be found in Appendices A.2 and A.3. The standard deviations of the running times and clustering cost are given in Appendix A.4.

As expected, k-MEANS++ is clearly superior to the classical algorithm k-MEANS both in terms of quality and running time. Comparing k-MEANS++ with our streaming algorithm, we find that on all datasets the quality of the clusterings computed by algorithm STREAMKM++ is on a par with or even better than the clusterings obtained by algorithm k-MEANS++. We conjecture that this is due to the fact that, in the last step of our algorithm, we run the k-MEANS++ algorithm five times on the coreset and choose the best clustering result obtained this way. On the other hand, for the experiments with the k-MEANS++ algorithm, we run the k-MEANS++ algorithm only once in each repetition of the experiment. However, the running time of k-MEANS++ is only comparable with algorithm STREAMKM++ for the smallest dataset *Spambase*. Even for moderately large datasets, like the *Covertype* dataset, we obtain that algorithm STREAMKM++ is orders of magnitude faster than k-MEANS++. We conclude that algorithm k-MEANS++ should only be used if the size of the dataset is not too large. For larger datasets, algorithm STREAMKM++ computes comparable clusterings in a significantly improved running time.

7 Well-Separated Pair Decomposition with Slack

In this chapter, we study the construction of well-separated pair decompositions (WSPDs) for point sets. Intuitively, two point sets are called a well-separated pair if the shortest distance from any point in one set to any point in the other set is large compared to the diameter of both sets. A well-separated pair decomposition of a point set consists of a collection of well-separated pairs that covers all the pairs of distinct points, i.e., any two distinct points belong to the different subsets of some pair. In this way, a WSPD of size t allows all pairwise distances to be compactly summarized by t distances.

Now, let us assume that we are given a huge point set P. Due to the size of P, it could be useful to have a compact representation that fairly captures the pairwise distances of P and uses space sublinear in $|P|$. In case that the structure of P is very simple (e.g., P is a multiset with many duplicates), it might be possible to construct a WSPD whose representation has sublinear size. However, in case that the structure of P is more complex, one cannot find a sublinear space representation of a WSPD such that all pairwise distances of P are well preserved. To be able to deal with this problem, we introduce the notion of a WSPD with slack. A WSPD with slack σ for P guarantees that at least a $(1-\sigma)$-fraction of all the pairwise distances of P are well preserved.

After giving a formal definition of a WSPD with slack, we present an efficient construction of a WSPD with low slack for low-dimensional Euclidean point sets in Section 7.2. Our construction is similar to the one we used in Chapter 5. We build a quadtree partition for the input points in which we recursively split every cell that contains more than a certain threshold of points. Based on this partition, we obtain a representation whose space requirement is polylogarithmic in both the size and the spread of the point set. In Section 7.3, we show how to transfer our construction for low-dimensional Euclidean point sets to point sets with bounded doubling dimension. Based on the techniques developed in this chapter, we will design streaming algorithms to compute low-distortion embeddings with low slack in Chapter 8.

7.1 Preliminaries

At first, we briefly recapitulate the classic notion of a well-separated pair decomposition (WSPD). A more detailed description can be found in [22, 103]. Afterwards, we relax the classic notion and give a formal definition of a WSPD with slack.

Let $M = (X, \mathrm{D})$ be any metric space, where X is a set of n points and D is a distance function defined on X (see Section 2.1 for a definition of metric spaces). Throughout this

chapter, we assume that the minimum pairwise distance between two points in X is at least 1, and the maximum pairwise distance is at most Δ. For any constant parameter ε with $0 < \varepsilon < 1$, two non-empty subsets $X_1, X_2 \subseteq X$ are called *ε-well-separated* if we have $\max\{\mathrm{diam}(X_1), \mathrm{diam}(X_2)\} \leq \varepsilon \cdot \mathrm{D}(X_1, X_2)$, where $\mathrm{diam}(X_1)$ and $\mathrm{diam}(X_2)$ are the diameters of X_1 and X_2, respectively. The value ε is often called *separation parameter*. Based on this, an ε-WSPD for M is defined as follows.

Definition 7.1.1 (WSPD)**.** *Let ε, $0 < \varepsilon < 1$, be a separation parameter. Let $M = (X, \mathrm{D})$ be any n-point metric space, and let $\mathcal{P}$ be a collection of ε-well-separated pairs of subsets $\{(A_0, B_0), \ldots, (A_{t-1}, B_{t-1})\}$, where $A_i, B_i \subseteq X$ for $i \in [t]$. $\mathcal{P}$ is called an* ε-WSPD *for M if every pair of points $(a, b) \in X \times X$, $a \neq b$, lies in $A_i \times B_i$ or $B_i \times A_i$ for exactly one index $i \in [t]$.*

The usefulness of an ε-WSPD for M is that, for any $i \in [t]$, the distances between pairs of points from $A_i \times B_i$ are all identical to within a factor of $1+2\varepsilon$. Thus, if we store instead of each pair (A_i, B_i) a pair of representative points $(R(A_i), R(B_i))$ with $R(A_i) \in A_i$ and $R(B_i) \in B_i$, then $\mathrm{D}(R(A_i), R(B_i))$ is a $(1 \pm 2\varepsilon)$-approximation of all the distances between pairs of points from $A_i \times B_i$. We also say that the distances between pairs of points from $A_i \times B_i$ are $(1 \pm 2\varepsilon)$-*preserved*. Hence, an ε-WSPD for M has the property that all pairwise distances of M are $(1 \pm 2\varepsilon)$-preserved.

Typically, one assumes that the size of an ε-WSPD for M is linear in n. Since we restrict the space requirement of the representation of M to be sublinear in n and there does not exist an ε-WSPD for any separation parameter ε and for any metric M (e.g., the uniform n-point metric) that has sublinear size, we introduce the notion of a WSPD with slack.

Definition 7.1.2 (WSPD with Slack)**.** *Let ε, $0 < \varepsilon < 1$, be a separation parameter and σ, $0 < \sigma < 1$, be a slack parameter. Let $M = (X, \mathrm{D})$ be any n-point metric space, and let $\mathcal{P}$ be a collection of pairs of subsets $\{(A_0, B_0), \ldots, (A_{t-1}, B_{t-1})\}$, where $A_i, B_i \subseteq X$ for $i \in [t]$. Let I_ε be the subset of indices such that, for all $j \in I_\varepsilon$, (A_j, B_j) is ε-well-separated. $\mathcal{P}$ is called an* ε-WSPD with slack σ *for M if every pair of points $(a, b) \in X \times X$, $a \neq b$, lies in $A_i \times B_i$ or $B_i \times A_i$ for at most one index $i \in [t]$ and*

$$\sum_{j \in I_\varepsilon} |A_j| \cdot |B_j| \geq (1 - \sigma) \cdot n^2 \ .$$

Despite the fact that the distance function D is symmetric, the slack σ of a WSPD is measured by the quantity of the fraction of all *ordered* pairs $(a, b) \in X \times X$ that do not satisfy the condition given in Definition 7.1.1. The assumption of having n^2 (instead of $\binom{n}{2}$) pairwise distances simplifies descriptions and makes our proofs cleaner, without changing the results in any significant way.

7.2 Construction for Euclidean Metric Spaces

This section deals with the construction of a WSPD with slack for Euclidean metrics. Let $M = (P, \mathrm{D})$ be an n-point Euclidean space with constant dimension d, let ε, $0 < \varepsilon < 1$,

be a separation parameter, and let σ, $2^{2d}/n < \sigma < 1$, be a slack parameter. In order to construct an ε-WSPD with slack σ for M, we impose $\lceil \log(\Delta) \rceil + 1$ nested square grids over P denoted by $\mathcal{G}(0), \mathcal{G}(1), \ldots, \mathcal{G}(\lceil \log(\Delta) \rceil)$. The side length of each cell in grid $\mathcal{G}(i)$ is 2^i. We say that the grid cells in $\mathcal{G}(i)$ are in level i.

Our algorithm consists of three phases. In the first phase, we compute a partition of the space based on the heavy cells in the grids (see Definition 7.2.1). Then, it follows a refinement phase, where each cell of the space partition is further subdivided into smaller cells, which we call *cubelets*. In the last phase, we determine a so-called *representative* for each cubelet and compute an ε-WSPD with slack σ from the set of representatives.

Definition 7.2.1 (Heavy Cell)**.** *We call a grid cell* heavy *if it contains at least $h(\sigma) \cdot n$ points of P, where $h(\sigma) := \sigma/2^d$ is a function dependent on σ. A grid cell that is not heavy is called* light*.*

Now, we describe the three phases in detail (see Algorithm 7.2.1 for a description in pseudocode). In the first phase, we build a partition of the point space based on a quadtree. To recapitulate, a quadtree for a d-dimensional point set is a rooted tree in which every internal node has 2^d children. Furthermore, every node corresponds to a grid cell and, for any internal node v, the cells of its children form a partition of the cell corresponding to v. Thus, the cells of the leaf nodes form a partition of the cell of the root node. We call this partition a *quadtree partition*. Our quadtree partition for P is now constructed as follows. We start with the coarsest grid $\mathcal{G}(\lceil \log(\Delta) \rceil)$ and identify all heavy cells in this grid, i.e., cells containing at least $h(\sigma) \cdot n$ points. Then, we subdivide every heavy cell $\mathcal{C}$ into 2^d equal sized subcells. These subcells are contained in grid $\mathcal{G}(\lceil \log(\Delta) \rceil - 1)$. We call $\mathcal{C}$ the parent cell of these subcells. If none of the subcells is heavy, we stop our process. Otherwise, the algorithm recursively subdivides every heavy cell such that, at the end of the first phase, we have only light cells in our space partition. Note that all the cells in grid $\mathcal{G}(0)$ are light since such a cell can contain at most 2^d points and $\sigma > 2^{2d}/n$ implies $h(\sigma) \cdot n > 2^d$. Figure 7.1 illustrates the quadtree partitioning with the help of an example.

The refinement phase consists of three steps. The first refinement is that we build a so-called balanced or restricted quadtree partition of the quadtree partition obtained so far, i.e., the side length of each cell is allowed to differ from the side lengths of all neighboring cells by a factor of at most 2 [33, 107]. That means that we further subdivide every leaf cell $\mathcal{C}$ of the quadtree which has a neighboring cell whose side length is less than half of the side length of $\mathcal{C}$. We say that two cells are neighbors if they share some part of the boundary. In Figure 7.2, the first refinement step is illustrated by using the example from Figure 7.1. In the second refinement step, every leaf cell of the balanced quadtree is subdivided into ${\ell_1}^d$ equal sized cubes, where $\ell_1 := \left\lceil 6\sqrt{d} \right\rceil$. Finally, we subdivide every cube into ${\ell_2(\varepsilon)}^d$ equal sized cubelets, where $\ell_2(\varepsilon) := \left\lceil 2\sqrt{d}/\varepsilon \right\rceil$ is a function dependent on ε. Note that we could have merged the second and third refinement step into one step, but the definition of cubes makes the analysis easier.

It remains to determine the representatives. For each non-empty cubelet $\mathcal{C}$, we replace all the points inside of $\mathcal{C}$ by one representative. This representative is set to the location of

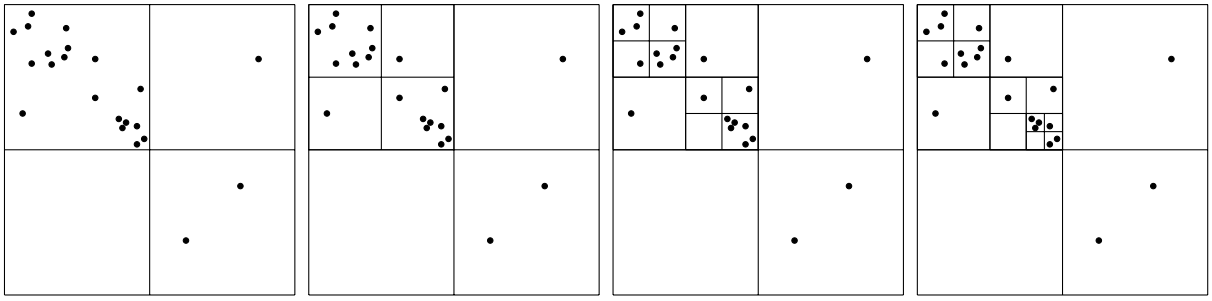


Figure 7.1: Example illustrating the quadtree partition for a point set in the plane. A cell is heavy if it contains at least 5 points. (a)-(d) The quadtree partition for subsequent depths of the recursion, i.e., after having subdivided each heavy cell in grid $\mathcal{G}(\lceil\log(\Delta)\rceil)$, $\mathcal{G}(\lceil\log(\Delta)\rceil - 1)$, $\mathcal{G}(\lceil\log(\Delta)\rceil - 2)$, and $\mathcal{G}(\lceil\log(\Delta)\rceil - 3)$, respectively. Since no cell in partition (d) is heavy, the recursion stops here.

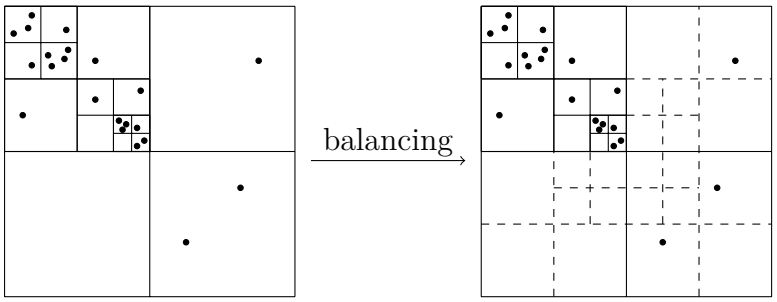


Figure 7.2: Example illustrating the refinement of a quadtree partition to get a balanced quadtree partition. Cells created during the balancing process are indicated by dashed lines.

an arbitrary point it represents (i.e., a point from $P \cap \mathcal{C}$) and weighted by the total number of replaced points. Finally, the collection of all representative pairs is our ε-WSPD with slack σ for M. Note that we implicitly store this information by storing the set of all representatives.

7.2.1 Analysis of the Construction

First, we prove that our construction yields an ε-WSPD with slack σ for M. Then, we analyze its complexity. For that purpose, for any level $i \in [\lceil\log(\Delta)\rceil + 1]$, let $\mathcal{L}(i)$ be the set of all the leaf cells of the unbalanced quadtree whose side length is 2^i, i.e., leaf cells in level i. Analogously, let $\mathcal{L}^+(i)$ be the set of all the leaf cells of the balanced quadtree whose side length is 2^i. We define $\mathcal{L}^*(i)$ to be the set of all the cubes contained in a cell in $\mathcal{L}^+(i)$. Furthermore, we denote by $\mathcal{H}(i)$ the set of heavy cells in level i that do not have a heavy subcell. Note that the parent cell of any cell in $\mathcal{L}(i)$ is in $\mathcal{H}(i+1)$.

Algorithm 7.2.1 ConstructWSPD(P, ε, σ)

initialize space partition with the cells in grid $\mathcal{G}(\lceil \log(\Delta) \rceil)$
initialize queue Q with the cells in grid $\mathcal{G}(\lceil \log(\Delta) \rceil)$
QuadtreePartition(Q)
$Q \leftarrow$ insert all cells of space partition
BalancedQuadtreePartition(Q)
for each cell $\mathcal{C}$ in space partition **do**
  split $\mathcal{C}$ into ${\ell_1}^d$ cubes
for each cube $\mathcal{C}$ in space partition **do**
  split $\mathcal{C}$ into $\ell_2(\varepsilon)^d$ cubelets
initialize empty set R of representatives
for each non-empty cubelet $\mathcal{C}$ in space partition **do**
  $q \leftarrow$ arbitrary point from $P \cap \mathcal{C}$ weighted by number of points in $\mathcal{C}$
  $R \leftarrow R \cup q$
return R

Algorithm 7.2.2 QuadtreePartition(P, σ, Q)

while Q is not empty **do**
  $\mathcal{C} \leftarrow$ remove first cell from Q
  if $\mathcal{C}$ is heavy **then**
    split $\mathcal{C}$ into 2^d subcells
    $Q \leftarrow$ insert all subcells of $\mathcal{C}$

Algorithm 7.2.3 BalancedQuadtreePartition(P, σ, Q)

while Q is not empty **do**
  $\mathcal{C} \leftarrow$ remove first cell from Q
  if $\mathcal{C}$ violates the balancing condition **then**
    split $\mathcal{C}$ into 2^d subcells
    $Q \leftarrow$ insert all subcells of $\mathcal{C}$
    $Q \leftarrow$ insert all neighbors of $\mathcal{C}$ that now violate the balancing condition

Separation and Slack

In the second phase of the algorithm, every cube in $\bigcup_{i=0}^{\lceil \log(\Delta) \rceil} \mathcal{L}^*(i)$ is divided into $\ell_2(\varepsilon)^d$ equal sized cubelets. The next lemma shows that the choice of the function $\ell_2(\varepsilon)$ guarantees that any two cubelets of different non-neighboring cubes are ε-well-separated.

Lemma 7.2.2. *Let $\ell_2(\varepsilon) := \left\lceil 2\sqrt{d}/\varepsilon \right\rceil$. If each cube in $\bigcup_{i=0}^{\lceil \log(\Delta) \rceil} \mathcal{L}^*(i)$ is divided into $\ell_2(\varepsilon)^d$ equal sized cubelets, then any two cubelets which are not contained in the same cube or in neighboring cubes are ε-well-separated.*

Proof. Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be any two cubelets which are not contained in the same cube or in neighboring cubes. Furthermore, let $\mathcal{C}_1$ be in any level $i \in [\lceil \log(\Delta) \rceil + 1]$ and $\mathcal{C}_2$ be in a

level j. Without loss of generality, we assume that $j \in \{i, \ldots, \lceil \log(\Delta) \rceil\}$. We consider the two cases $j \in \{i, i+1\}$ and $j \in \{i+2, \ldots, \lceil \log(\Delta) \rceil\}$.

We start with the case $j \in \{i, i+1\}$. The side length of a cube in level i is $2^i/\ell_1$, where $\ell_1 = \left\lceil 6\sqrt{d} \right\rceil$. Since the side lengths of neighboring cells in $\bigcup_{k=0}^{\lceil \log(\Delta) \rceil} \mathcal{L}^+(k)$ differ by a factor of at most 2 and each cell in $\bigcup_{k=0}^{\lceil \log(\Delta) \rceil} \mathcal{L}^+(k)$ is divided into equal sized cubes, the distance between the cube containing $\mathcal{C}_1$ and the cube containing $\mathcal{C}_2$ is at least $2^i/\ell_1$. Since the diagonal of the bigger cubelet $\mathcal{C}_2$ is

$$\operatorname{diag}(\mathcal{C}_2) \;=\; \frac{\sqrt{d} \cdot 2^j}{\ell_1 \cdot \ell_2(\varepsilon)} \;\leq\; \frac{\varepsilon \cdot 2^i}{\ell_1} \;,$$

we get that $\mathcal{C}_1$ and $\mathcal{C}_2$ are ε-well-separated (see Section 7.1 for a definition of an ε-well-separated pair).

Now, we consider the case $j \in \{i+2, \ldots, \lceil \log(\Delta) \rceil\}$. Due to the balanced quadtree partitioning, the side lengths of neighboring cells in $\bigcup_{k=0}^{\lceil \log(\Delta) \rceil} \mathcal{L}^+(k)$ differ by a factor of at most 2. Hence, the distance between any cell in $\mathcal{L}^+(i)$ and any cell in $\mathcal{L}^+(j)$ is at least $\sum_{k=i+1}^{j-1} 2^k = 2^j - 2^{i+1} \geq 2^{j-1}$. Since $\mathcal{C}_1$ is contained in a cell in $\mathcal{L}^+(i)$ and $\mathcal{C}_2$ is contained in a cell in $\mathcal{L}^+(j)$, the distance between $\mathcal{C}_1$ and $\mathcal{C}_2$ is at least 2^{j-1}. Since the diagonal of the bigger cubelet $\mathcal{C}_2$ is

$$\operatorname{diag}(\mathcal{C}_2) \;=\; \frac{\sqrt{d} \cdot 2^j}{\ell_1 \cdot \ell_2(\varepsilon)} \;<\; \varepsilon \cdot 2^{j-1} \;,$$

the two cubelets $\mathcal{C}_1$ and $\mathcal{C}_2$ are ε-well-separated. □

Due to Lemma 7.2.2, to bound the slack of the ε-WSPD for M, we have to bound the number of points in each cube in $\bigcup_{i=0}^{\lceil \log(\Delta) \rceil} \mathcal{L}^*(i)$. The following lemma shows that this is guaranteed by our choice of $h(\sigma)$.

Lemma 7.2.3. *Let $h(\sigma) := \sigma/2^d$, and let p_1 and p_2 be any two points in P. If the cubelet that contains p_1 and the cubelet that contains p_2 are not ε-well-separated, then p_2 belongs to the σn closest points of p_1.*

Proof. At first, we bound the maximum distance $\mathrm{D}(p_1, p_2)$ between p_1 and p_2. Then, we bound the total number of points whose distance from p_1 is at most $\mathrm{D}(p_1, p_2)$. If this number is at most σn, then the correctness of the lemma follows.

For any level $i \in [\lceil \log(\Delta) \rceil + 1]$, let $\mathcal{C}_1^* \in \mathcal{L}^*(i)$ be the cube that contains p_1, and let $\mathcal{C}_2^*$ be the cube that contains p_2. Due to Lemma 7.2.2, $\mathcal{C}_1^*$ and $\mathcal{C}_2^*$ must be neighbors. Since we use a balanced quadtree partitioning, the side lengths of $\mathcal{C}_1^*$ and $\mathcal{C}_2^*$ differ by a factor of at most 2. Since $\mathcal{C}_1^* \in \mathcal{L}^*(i)$, the side lengths of $\mathcal{C}_1^*$ is $2^i/\ell_1$ with $\ell_1 = \left\lceil 6\sqrt{d} \right\rceil$. We have to consider the cases (i) $\mathcal{C}_2^* \in \mathcal{L}^*(i)$, (ii) $\mathcal{C}_2^* \in \mathcal{L}^*(i+1)$, and (iii) $\mathcal{C}_2^* \in \mathcal{L}^*(i-1)$.

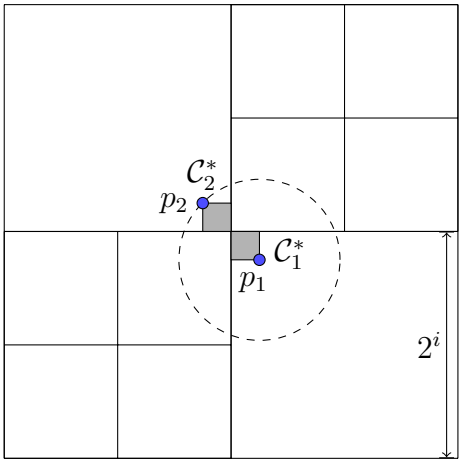


Figure 7.3: Sketch of Case (i) in the proof of Lemma 7.2.3.

Case (i) is illustrated in Figure 7.3. In this case, the maximum distance between p_1 and p_2 is at most

$$\begin{aligned} \mathrm{D}(p_1, p_2) &\leq \sqrt{d} \cdot \left(\frac{2^i}{\ell_1} + \frac{2^i}{\ell_1} \right) \\ &= \sqrt{d} \cdot \left(\frac{2^{i+1}}{\ell_1} + \frac{2^i}{\sqrt{d}\ell_1} \right) - \frac{2^i}{\ell_1} \\ &\leq 2^{i-1} - \frac{2^i}{\ell_1} \ . \end{aligned}$$

Since the cube $\mathcal{C}_1^*$ is contained in a cell in $\mathcal{L}^+(i)$ and we use a balanced quadtree partitioning, the side lengths of all neighboring cells of the cell containing $\mathcal{C}_1^*$ are at least 2^{i-1}. Now, since the side length of $\mathcal{C}_1^*$ is $2^i/\ell_1$, the ball with center p_1 and radius $\mathrm{D}(p_1, p_2) \leq 2^{i-1} - 2^i/\ell_1$ is covered by at most 2^d cells in $\bigcup_{k=0}^{\lceil \log(\Delta) \rceil} \mathcal{L}^+(k)$.

In Case (ii), the maximum distance between p_1 and p_2 is at most

$$\mathrm{D}(p_1, p_2) \leq \sqrt{d} \cdot \left(\frac{2^i}{\ell_1} + \frac{2^{i+1}}{\ell_1} \right) < 2^i - \frac{2^i}{\ell_1} \ .$$

Furthermore, since $\mathcal{C}_2^*$ is contained in a cell in $\mathcal{L}^+(i+1)$, the side lengths of all common neighbors of the cells containing $\mathcal{C}_1^*$ and $\mathcal{C}_2^*$ are at least 2^i. Now, since the side length of $\mathcal{C}_1^*$ is $2^i/\ell_1$, the ball with center p_1 and radius $\mathrm{D}(p_1, p_2) \leq 2^i - 2^i/\ell_1$ can be covered by at most 2^d cells in $\bigcup_{k=0}^{\lceil \log(\Delta) \rceil} \mathcal{L}^+(k)$.

Case (iii) is symmetric to Case (ii).

As a result, in all three cases, we have to count the number of points in 2^d cells in $\bigcup_{k=0}^{\lceil \log(\Delta) \rceil} \mathcal{L}^+(k)$. Since the cells in $\bigcup_{k=0}^{\lceil \log(\Delta) \rceil} \mathcal{L}^+(k)$ are light cells, each one of them contains at most $h(\sigma) \cdot |P|$ points. It follows that the number of points whose distance from p_1 is at most $\mathrm{D}(p_1, p_2)$ is at most $\sigma \cdot |P|$. □

Remark 7.2.4. *Lemma 7.2.3 does not only imply that the collection of all representative pairs is an ε-WSPD with slack σ for M. It also lets us know where the slack arises. More precisely, for each point in P, the distances to its σn closest neighbors in P can be arbitrarily distorted, but the distances to all other points in P are $(1 \pm 2\varepsilon)$-preserved.*

Complexity

In order to upper bound the number of representatives, we have to bound the number of cells in the balanced quadtree partition. This is done by first analyzing the dependency of this number on the number of cells in the unbalanced quadtree partition. The proof of the following lemma is basically given in [33]. Only a few adjustments to our scenario have been made. However, for sake of completeness, we include the full proof here.

Lemma 7.2.5 ([33])**.** *The number of cells in the balanced quadtree partition is*

$$\left| \bigcup_{i=0}^{\lceil \log(\Delta) \rceil} \mathcal{L}^+(i) \right| \in \mathcal{O}\left(6^d \cdot \left| \bigcup_{i=0}^{\lceil \log(\Delta) \rceil} \mathcal{L}(i) \right| \right) .$$

Proof. The proof follows the one in Chapter 14 from [33]. We call the cells in the unbalanced quadtree *old cells* and the cells that are in the balanced but not in the unbalanced quadtree *new cells*. We will show that, for each old cell, there are at most $3^d - 1$ cells in the same level that have to be split. Since each split operation creates 2^d new cells, the total number of new cells is at most 6^d times the total number of old cells. Since the total number of cells in the unbalanced quadtree is at most $2 \cdot |\bigcup_{i=0}^{\lceil \log(\Delta) \rceil} \mathcal{L}(i)|$, the total number of new cells is at most $2 \cdot 6^d \cdot |\bigcup_{i=0}^{\lceil \log(\Delta) \rceil} \mathcal{L}(i)|$. Thus, the number of cells in $\bigcup_{i=0}^{\lceil \log(\Delta) \rceil} \mathcal{L}^+(i)$ is obviously upper bounded by the number of leaf cells in the unbalanced quadtree plus the total number of new cells, which is at most $|\bigcup_{i=0}^{\lceil \log(\Delta) \rceil} \mathcal{L}(i)| + 2 \cdot 6^d \cdot |\bigcup_{i=0}^{\lceil \log(\Delta) \rceil} \mathcal{L}(i)|$.

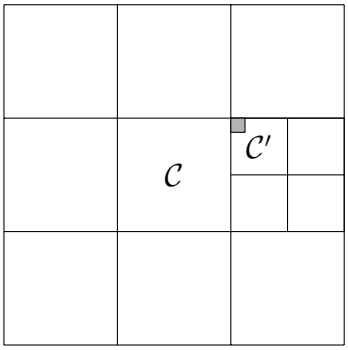


Figure 7.4: Illustration of the arrangement of cells in the proof of Lemma 7.2.5. The neighboring cell of $\mathcal{C}$ which causes the splitting of $\mathcal{C}$ is indicated in gray.

We use a charging argument to prove that, for each old cell, there are at most $3^d - 1$ cells in the same level that have to be split. Let us suppose that we have to split an (old or new) cell $\mathcal{C} \in \mathcal{G}(i)$ in any level $i \in [\lceil \log(\Delta) \rceil + 1]$ during the balancing process. Then,

we claim that at least one of the $3^d - 1$ neighboring cells of $\mathcal{C}$ in level i is old. We charge the splitting of $\mathcal{C}$ to this old cell.

Let us assume that our claim is wrong. Let $i \in [\lceil \log(\Delta) \rceil + 1]$ be the smallest level and $\mathcal{C} \in \mathcal{G}(i)$ be a cell such that $\mathcal{C}$ is split during the balancing process and has no neighboring cell in level i which is old (see Figure 7.4). Since $\mathcal{C}$ is split, there is a neighboring cell $\mathcal{C}''$ of $\mathcal{C}$ with side length at most 2^{i-2}. Let $\mathcal{C}' \in \mathcal{G}(i-1)$ be the cell that contains $\mathcal{C}''$. Due to the fact that $\mathcal{C}'$ is contained in a new cell, it is new itself. Also, all the neighboring cells of $\mathcal{C}'$ in level $i-1$ are new. This follows from the fact that all neighboring cells of $\mathcal{C}$ in level i are new and $\mathcal{C}$ is split during the balancing process. Furthermore, since $\mathcal{C}'$ contains the cell $\mathcal{C}''$, $\mathcal{C}'$ is split during the balancing process. Thus, $\mathcal{C}'$ is a cell that is split in the balancing process and there is no neighboring cell of $\mathcal{C}'$ in level $i-1$ which is old. This is a contradiction to the choice of $\mathcal{C}$. □

Lemma 7.2.6. *The number of cells in the balanced quadtree partition is* $\mathcal{O}(2^{\mathcal{O}(d)} \cdot \log(\Delta)/\sigma)$.

Proof. Each ancestor cell of a heavy cell is heavy, and heavy cells are split during the quadtree partitioning. Recall that, for any level $i \in [\lceil \log(\Delta) \rceil + 1]$, $\mathcal{H}(i)$ is the set of all heavy cells in level i that do not have a heavy subcell. Then, for each cell in $\mathcal{H}(i)$, there are 2^d cells in $\mathcal{G}(i-1)$ and at most $2^d \cdot (\lceil \log(\Delta) \rceil + 1)$ cells in $\bigcup_{j=i}^{\lceil \log(\Delta) \rceil} \mathcal{G}(j)$ that are cells in our quadtree. Since there are at most $1/h(\sigma)$ heavy cells in $\bigcup_{i=0}^{\lceil \log(\Delta) \rceil} \mathcal{H}(i)$, the total number of cells in the unbalanced quadtree is bounded by $\mathcal{O}(1/h(\sigma) \cdot 2^d \cdot \log(\Delta))$. Due to Lemma 7.2.5, the number of cells in the balanced quadtree partition is $\mathcal{O}(1/h(\sigma) \cdot 12^d \cdot \log(\Delta))$. Now, the lemma follows from $h(\sigma) = \sigma/2^d$. □

Lemma 7.2.7. *The space partition consists of* $\mathcal{O}(2^{\mathcal{O}(d)} \cdot d^d \cdot \log(\Delta)/(\varepsilon^d \sigma))$ *cubelets.*

Proof. Due to Lemma 7.2.6, there are $\mathcal{O}(2^{\mathcal{O}(d)} \cdot \log(\Delta)/\sigma)$ cells in the balanced quadtree partition. Now, the lemma follows from the fact that each cell in the balanced quadtree partition contains $\left\lceil 6\sqrt{d} \right\rceil^d$ cubes and each cube contains $\left\lceil 2\sqrt{d}/\varepsilon \right\rceil^d$ cubelets. □

Based on the results given above, we can now analyze the complexity of our algorithm.

Lemma 7.2.8. *The algorithm has a running time of*

$$\mathcal{O}\left(n \cdot \left(\frac{d^2}{\varepsilon} + d\log(\Delta)\right) + \frac{2^{\mathcal{O}(d)} \cdot d \cdot \log^2(\Delta)}{\sigma} + \frac{2^{\mathcal{O}(d)} \cdot d^d \cdot \log(\Delta)}{\varepsilon^d \sigma}\right)$$

and a space requirement of

$$\mathcal{O}\left(dn + \frac{2^{\mathcal{O}(d)} \cdot d^d \cdot \log(\Delta)}{\varepsilon^d \sigma}\right) .$$

Proof. Since each level of a quadtree forms a partition of P, the total number of points associated with cells at the same level of the quadtree is at most n. Thus, the arrangement of points in any level of the quadtree can be computed from the preceding level in $\mathcal{O}(dn)$

time. Since our quadtree has a height of at most $\lceil \log(\Delta) \rceil + 1$, the total running time to compute the arrangements of the points in all the levels of the quadtree is $\mathcal{O}(dn \log(\Delta))$.

During the balancing process, for each cell that has ever been split, we check if this cell has neighbors in the quadtree partition that violate the balancing condition. Let $\mathcal{C}$ be such a split cell in any level $i \in [\lceil \log(\Delta) \rceil + 1]$. Since we must only check neighbors of $\mathcal{C}$ whose side length is at least twice as big as the side length of $\mathcal{C}$, the level of the neighbors that we have to check is at least $i+1$ and the total number of these neighbors is at most $2^d - 1$ (see Figure 7.5). Let $\mathcal{C}'$ be any cell in $\mathcal{G}(i+1)$ that shares some part of its boundary with $\mathcal{C}$. If the subcells of $\mathcal{C}'$ are not cells of the current quadtree, then the leaf cell of the current quadtree that contains $\mathcal{C}'$ is a violating neighbor of $\mathcal{C}$. We can find this violating neighbor by searching for the leaf cell in the quadtree that contains the midpoint of $\mathcal{C}'$. Since the height of the quadtree is $\mathcal{O}(\log(\Delta))$, this cell can be found in $\mathcal{O}(d \log(\Delta))$ time. Furthermore, due to Lemma 7.2.6, the number of cells in the balanced quadtree is $\mathcal{O}(2^{\mathcal{O}(d)} \cdot \log(\Delta)/\sigma)$. Thus, the time for checking neighborhoods during the balancing process is $\mathcal{O}(2^{\mathcal{O}(d)} \cdot d \cdot \log^2(\Delta)/\sigma)$.

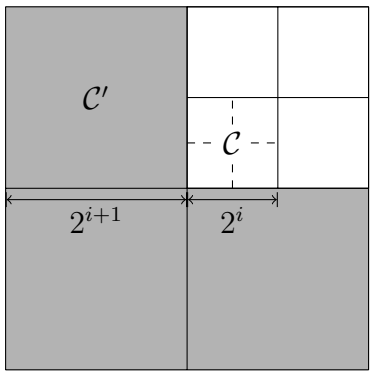


Figure 7.5: The cell $\mathcal{C}$ is split during the balancing operation. A neighbor of $\mathcal{C}$ in the current quadtree partition that violates the balancing condition has to contain at least one of the cells colored in gray.

Finally, the arrangement of the points in the cubelets can be computed from the balanced quadtree partition in $n \cdot d \cdot (\ell_1 \cdot \ell_2(\varepsilon)) \in \mathcal{O}(d^2 n/\varepsilon)$ time.

Obviously, each node of the final partition tree has to be created once. Since it follows from Lemma 7.2.7 that this tree consists of $\mathcal{O}(2^{\mathcal{O}(d)} \cdot d^d \cdot \log(\Delta)/(\varepsilon^d \sigma))$ nodes, this can be done in $\mathcal{O}(2^{\mathcal{O}(d)} \cdot d^d \cdot \log(\Delta)/(\varepsilon^d \sigma))$ time. Thus, the total running time is as claimed in the lemma. Furthermore, the space requirement for the d-dimensional points in P and the partition tree is upper bounded by $\mathcal{O}(dn + 2^{\mathcal{O}(d)} \cdot d^d \cdot \log(\Delta)/(\varepsilon^d \sigma))$. □

We summarize our results in the following theorem:

Theorem 7 (WSPD with Slack for Euclidean Spaces)**.** *Let P be a set of n points with spread Δ from a low-dimensional Euclidean space $\mathbb{R}^d$, let ε, $0 < \varepsilon < 1$, be a separation parameter, and let σ, $2^{2d}/n < \sigma < 1$, be a slack parameter. Then, there exists an algorithm*

that computes a weighted point set $P' \subset P$ with cardinality $\mathcal{O}(2^{\mathcal{O}(d)} \cdot d^d \cdot \log(\Delta)/(\varepsilon^d \sigma))$ which is an implicit representation of an ε-WSPD with slack σ for P. The algorithm has a running time of

$$\mathcal{O}\left(n \cdot \left(\frac{d^2}{\varepsilon} + d\log(\Delta)\right) + \frac{2^{\mathcal{O}(d)} \cdot d \cdot \log^2(\Delta)}{\sigma} + \frac{2^{\mathcal{O}(d)} \cdot d^d \cdot \log(\Delta)}{\varepsilon^d \sigma}\right)$$

and a space requirement of

$$\mathcal{O}\left(dn + \frac{2^{\mathcal{O}(d)} \cdot d^d \cdot \log(\Delta)}{\varepsilon^d \sigma}\right) \ .$$

Proof. We have $P' \subset P$ since the location of each representative is a point from P. It follows from Lemmas 7.2.2 and 7.2.3 that P' is an implicit representation of an ε-WSPD with slack σ for P. The cardinality of P' is implied by Lemma 7.2.7, and the running time and space requirement to compute P' is due to Lemma 7.2.8. In our construction, we made the assumption that $\sigma > 2^{2d}/n$ to ensure that all the cells in grid $\mathcal{G}(0)$ are light. □

7.3 Construction for Doubling Metric Spaces

We transfer our approach to construct a WSPD with slack for low-dimensional Euclidean point sets to point sets with bounded doubling dimension. The input of our algorithm is an n-point doubling metric space $M = (X, \mathrm{D})$ with bounded dimension λ, a separation parameter ε, $0 < \varepsilon < 1$, and a slack parameter σ, $(\lceil \log(\Delta) \rceil + 1) \cdot 2^{6\lambda+3}/n < \sigma < 1$. Recall from the definition of doubling metric spaces that each ball in M with any radius r centered at any point in X can be covered by 2^λ balls each of radius $r/2$ and centered at a point in X. We assume that the minimum pairwise distance between two points in X is at least 1, and the maximum pairwise distance is at most Δ. Furthermore, we assume access to a distance oracle that, given any two points from X, can compute in constant time the distance between these two points.

Our idea is to replace the square grids from Section 7.2 by *uniform cut decompositions*, which are defined as follows:

Definition 7.3.1 (Uniform Cut Decomposition, [35])**.** *Let X be a non-empty point set with a distance function defined on it. An* r-cut decomposition *of X is a partition of X into balls such that the following conditions are satisfied:*

i) Each ball is centered at a point in X.

ii) Each ball has radius r.

iii) Each point in X is covered by a ball.

Now, we explain our construction in detail (see Algorithm 7.3.1 for a description in pseudocode). For each $i \in [\lceil \log(\Delta) \rceil + 1]$, we compute a 2^i-cut decomposition of the point

set X. We say that the balls with radius 2^i are in level i. We denote the set of balls in level i by $\mathcal{G}(i)$. In case that a point is covered by more than one ball, we assign it to any one of them. We point out that the arrangement of balls in the uniform cut decomposition of any level does not depend on the arrangement of balls in the uniform cut decomposition of any other level. According to this and in contrast to our approach for low-dimensional Euclidean point sets, our algorithm for doubling metric spaces is not recursive.

To compute the WSPD for X, we identify in each level the *heavy balls*.

Definition 7.3.2 (Heavy Ball)**.** *We call a ball* heavy *if it contains at least $h(\sigma) \cdot n$ points of X, where $h(\sigma) := \sigma/((\lceil \log(\Delta) \rceil + 1) \cdot 2^{5\lambda+3})$ is a function dependent on σ. A ball that is not heavy is called* light*.*

Algorithm 7.3.1 ConstructWSPD(X, ε, σ)

```
1: initialize empty set R of representatives
2: for i ← 0 to ⌈log(Δ)⌉ do
3:    G(i) ← set of balls in 2^i-cut decomposition of X
4:    for each heavy ball B in G(i) do
5:       decompose B into mini balls with radius 2^(i−ℓ(ε))
6:       for each mini ball B' in B do
7:          y ← point located at center of B' and with weight w(y) ← 0
8:          for each point x ∈ X ∩ B' do
9:             if x is not marked then
10:               mark x
11:               w(y) ← w(y) + 1
12:         R ← R ∪ y
13: return R
```

In each level $i \in [\lceil \log(\Delta) \rceil + 1]$, we identify the heavy balls, i.e., balls containing at least $h(\sigma) \cdot n$ points from X. Each of these heavy balls is decomposed into mini balls of radius $2^{i-\ell(\varepsilon)}$, where $\ell(\varepsilon) := \lceil \log(1/\varepsilon) \rceil$. Note that all the balls in $\mathcal{G}(0)$ are light since such a ball can contain at most 2^λ points and $\sigma > (\lceil \log(\Delta) \rceil + 1) \cdot 2^{6\lambda+3}/n$ implies $h(\sigma) \cdot n > 2^\lambda$.

Next, we compute a representative for each point in X. Let $x \in X$ be any point. Then, the representative of x is the center of the smallest mini ball that contains x. Note that each mini ball belongs to a uniform cut decomposition of a heavy ball and each point in X is contained in at least one heavy ball since the ball in level $\lceil \log(\Delta) \rceil$ is heavy and covers all points from X. The set of representatives for X is the weighted set of center points of the mini balls where each such point is weighted by the number of points it represents. The collection of all representative pairs is our compact representation for M. As mentioned in Section 7.2, we implicitly store this information by storing the set of all representatives.

Note that, for any level $i \in [\lceil \log(\Delta) \rceil + 1]$, a 2^i-cut decomposition of X can be computed by applying the well-known 2-approximation algorithm for vertex cover. Let $G = (V, E)$ be any simple graph. Then, the vertex-cover algorithm chooses repeatedly any edge

$\{x, y\} \in E$, inserts x and y into the currently found vertex cover and removes all edges incident to x or y from E. This is done until the edge set E is empty. Since, in this way, we implicitly find a non-extendable matching of G which is always a vertex cover for G and an optimal cover contains at least one endpoint of each edge in this matching, the algorithm outputs a 2-approximation for vertex cover. Now, let G be the graph that has a vertex for each point in X and an edge $\{x, y\}$ for each unordered pair of points $x, y \in X$ with $\mathrm{D}(x, y) \leq 2^i$. Then, by applying the 2-approximation algorithm for vertex cover on G, we compute a 2^i-cut decomposition of X whose size is at most twice as big as the size of an optimal 2^i-cut decomposition of X.

7.3.1 Analysis of the Construction

First, we show that our construction computes an ε-WSPD with slack σ for M. Then, we analyze its complexity.

For any level $i \in [\lceil \log(\Delta) \rceil + 1]$, let $\mathcal{H}(i)$ be the set of heavy balls in level i. We call any two balls $\mathcal{B}(x_1, r_1)$ and $\mathcal{B}(x_2, r_2)$ *neighboring balls* if $\mathcal{B}(x_1, 3r_1)$ contains at least one point of $\mathcal{B}(x_2, r_2)$ or $\mathcal{B}(x_2, 3r_2)$ contains at least one point of $\mathcal{B}(x_1, r_1)$.

Separation and Slack

The following lemma proves that the choice of the function $\ell(\varepsilon)$ guarantees that any two mini balls contained in different non-neighboring heavy balls are ε-well-separated.

Lemma 7.3.3. *Let $\ell(\varepsilon) := \lceil \log(1/\varepsilon) \rceil$. If, for each level $i \in [\lceil \log(\Delta) \rceil + 1]$, each heavy ball in $\mathcal{H}(i)$ is decomposed into mini balls with radius $2^{i-\ell(\varepsilon)}$, then any two mini balls contained in different non-neighboring heavy balls are ε-well-separated.*

Proof. Let $\mathcal{B}_1$ be any heavy ball in $\mathcal{H}(i)$ in any level $i \in [\lceil \log(\Delta) \rceil + 1]$, and let $\mathcal{B}_2$ be any heavy ball in $\mathcal{H}(j)$ such that $\mathcal{B}_1$ and $\mathcal{B}_2$ are non-neighboring balls (see Figure 7.6 for an illustration). Without loss of generality, we assume that $j \in \{i, \ldots, \lceil \log(\Delta) \rceil\}$. Since the radius of $\mathcal{B}_2$ is 2^j and $\mathcal{B}_1$ is not a neighboring ball of $\mathcal{B}_2$, the distance between the center of $\mathcal{B}_2$ and any point in $\mathcal{B}_1$ is larger than $3 \cdot 2^j$. Hence, the distance between any point in $\mathcal{B}_1$ to any point in $\mathcal{B}_2$ is larger than 2^{j+1}. Thus, the distance between any point in any mini ball $\mathcal{B}_1^*$ in $\mathcal{B}_1$ to any point in any mini ball $\mathcal{B}_2^*$ in $\mathcal{B}_2$ is larger than 2^{j+1}. Now, the assertion of the lemma follows from the fact that the diameters of $\mathcal{B}_1^*$ and $\mathcal{B}_2^*$ are at most

$$\mathrm{diam}(\mathcal{B}_2^*) \;=\; 2 \cdot 2^{j-\ell(\varepsilon)} \;\leq\; \varepsilon \cdot 2^{j+1} \;\; .$$

□

Due to Lemma 7.3.3, to bound the slack of the ε-WSPD for M, we have to bound the number of points in each heavy ball. The following lemma shows that this is guaranteed by our choice of $h(\sigma)$.

Lemma 7.3.4. *Let $h(\sigma) := \sigma/((\lceil \log(\Delta) \rceil + 1) \cdot 2^{5\lambda+3})$, then the collection of all representative pairs is an ε-WSPD with slack σ for M.*

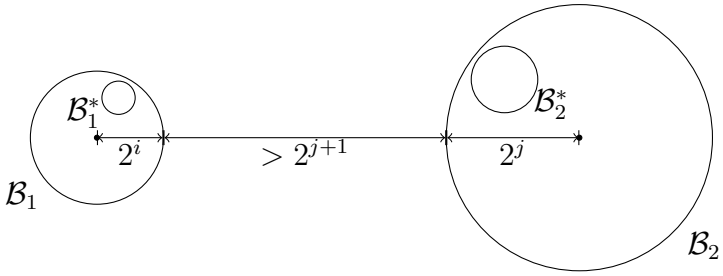


Figure 7.6: Illustration of the two non-neighboring heavy balls $\mathcal{B}_1$ and $\mathcal{B}_2$ in the proof of Lemma 7.3.3.

Proof. For any point $x_1 \in X$, we compute an upper bound on the number of points $x_2 \in X$ such that the radius of the smallest mini ball containing x_2 is at least as big as the radius of the smallest mini ball containing x_1 and the two mini balls are not ε-well-separated. By multiplying this number by $2n$, we get an upper bound on the slack of our ε-WSPD for all ordered pairs $(x_1, x_2) \in X \times X$.

Let x_1 and x_2 be any points in X that satisfy the condition above. Let $i \in [\lceil \log(\Delta) \rceil + 1]$ be the level such that $\mathcal{B}_1 \in \mathcal{H}(i)$ is the smallest heavy ball that contains x_1. Let x be the center of $\mathcal{B}_1$. Furthermore, let $\mathcal{B}_2 \in \mathcal{H}(j)$, $j \in \{i, \ldots, \lceil \log(\Delta) \rceil\}$, be the smallest heavy ball that contains x_2 (see Figure 7.7 for an illustration). First, we show that $\mathcal{B}_1$ and $\mathcal{B}_2$ must be neighboring balls. Then, we prove an upper bound of $2^{3\lambda+1}$ on the number of heavy balls in level j that are neighboring balls of $\mathcal{B}_1$. This allows us to derive an upper bound of $(\lceil \log(\Delta) \rceil + 1) \cdot 2^{3\lambda+1}$ on the total number of heavy balls in any level $j \in \{i, \ldots, \lceil \log(\Delta) \rceil\}$ that are neighboring balls of $\mathcal{B}_1$. Finally, we show that the total weight of all the representative points located in mini balls forming a cut decomposition of such a heavy ball is at most $\sigma n / ((\lceil \log(\Delta) \rceil + 1) \cdot 2^{3\lambda+2})$. Then, the number of neighboring heavy balls of $\mathcal{B}_1$ times the representative weight associated with a heavy ball times $2n$ is at most

$$(\lceil \log(\Delta) \rceil + 1) \cdot 2^{3\lambda+1} \cdot \frac{\sigma n}{(\lceil \log(\Delta) \rceil + 1) \cdot 2^{3\lambda+2}} \cdot 2n \leq \sigma \cdot n^2 \ ,$$

which proves that the slack is at most σ.

Since the smallest mini ball that contains x_1 and the smallest mini ball that contains x_2 are not ε-well-separated, it follows from Lemma 7.3.3 that $\mathcal{B}_1$ and $\mathcal{B}_2$ are neighboring balls. Since $\mathcal{B}_2$ is at least as big as $\mathcal{B}_1$, the distance from the center of $\mathcal{B}_2$ to the closest point in $\mathcal{B}_1$ is at most $3 \cdot 2^j$. It follows that the distance from the center of $\mathcal{B}_1$ to any point in $\mathcal{B}_2$ is at most $2^i + 3 \cdot 2^j + 2^j < 2^{j+3}$. Hence, $\mathcal{B}_2$ is completely contained in the ball $\mathcal{B}(x, 2^{j+3})$. Since M is a doubling metric space, $\mathcal{B}(x, 2^{j+3})$ can be covered by at most $2^{3\lambda}$ balls of radius 2^j. Since we use a 2-approximation algorithm to compute cut decompositions, the number of balls in level j that are completely contained in $\mathcal{B}(x, 2^{j+3})$ is at most $2^{3\lambda+1}$. Hence, the number of balls in level j that are neighboring balls of $\mathcal{B}_1$ is at most $2^{3\lambda+1}$. Summing up over all levels $j \in \{i, \ldots, \lceil \log(\Delta) \rceil\}$, the total number of balls that are neighboring balls of $\mathcal{B}_1$ is at most $(\lceil \log(\Delta) \rceil + 1) \cdot 2^{3\lambda+1}$.

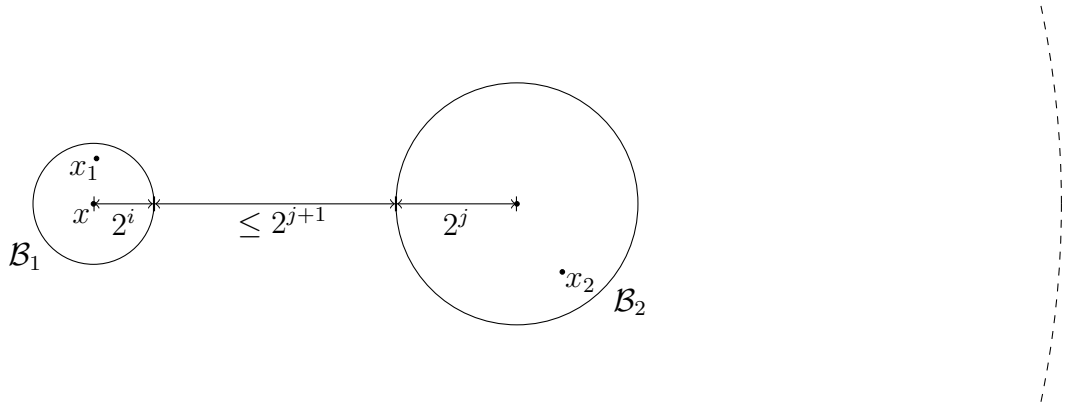


Figure 7.7: Illustration of the two neighboring heavy balls $\mathcal{B}_1$ and $\mathcal{B}_2$ in the proof of Lemma 7.3.4. The ball $\mathcal{B}(x, 2^{j+3})$, which completely contains $\mathcal{B}_2$, is indicated by the dashed arc.

Let $\mathcal{B}(x', 2^j) \in \mathcal{H}(j)$, $j \in \{i, \ldots, \lceil \log(\Delta) \rceil\}$, be any neighboring ball of $\mathcal{B}_1$. Observe that the representative point of any point $x'' \in \mathcal{B}(x', 2^j)$ is not from level j if x'' is covered by a heavy ball with radius smaller than 2^j. Obviously, it follows that the representative point of any point $x'' \in \mathcal{B}(x', 2^j)$ is not from level j if x'' is covered by a heavy ball in $\mathcal{H}(j-1)$. Thus, we compute an upper bound on the number of points in all the light balls in $\mathcal{G}(j-1)$ that are at least partly covered by $\mathcal{B}(x', 2^j)$. Observe that these balls are completely contained in the ball $\mathcal{B}(x', 2^{j+1})$. Due to our construction, the number of balls in $\mathcal{G}(j-1)$ that are completely contained in the ball $\mathcal{B}(x', 2^{j+1})$ is at most $2^{2\lambda+1}$. Since any light ball contains less than $h(\sigma) \cdot n$ points and $h(\sigma) = \sigma/((\lceil \log(\Delta) \rceil + 1) \cdot 2^{5\lambda+3})$, the total number of points in all the light balls in $\mathcal{G}(j-1)$ that are completely contained in the ball $\mathcal{B}(x', 2^{j+1})$ is less than $h(\sigma) \cdot n \cdot 2^{2\lambda+1} \leq \sigma n/((\lceil \log(\Delta) \rceil + 1) \cdot 2^{3\lambda+2})$. Thus, the total weight of representative points in $\mathcal{B}(x', 2^j)$ is at most $\sigma n/((\lceil \log(\Delta) \rceil + 1) \cdot 2^{3\lambda+2})$, which was the only thing left to prove the assertion of the lemma. □

Unfortunately, in contrast to our WSPD construction for low-dimensional Euclidean spaces, the property that, for each point, the distances to its σn closest neighbors can be arbitrarily distorted, but the distances to all the other points are $(1 \pm 2\varepsilon)$-preserved (see Remark 7.2.4), does not hold for our WSPD construction in doubling metric spaces. The reason is that neighboring balls can differ in their radii by more than a constant factor. Due to this fact, there might exist a ball with many neighboring balls of smaller radius, so the number of the distances from a single point to other points in X that are not $(1 \pm 2\varepsilon)$-preserved might be bigger than σn.

Complexity

In order to upper bound the number of representatives, we have to bound the total number of mini balls.

Lemma 7.3.5. *The number of mini balls is* $\mathcal{O}(2^{\mathcal{O}(\lambda)} \cdot \log^2(\Delta)/(\varepsilon^\lambda \sigma))$.

Proof. Let $i \in [\lceil \log(\Delta) \rceil + 1]$ be any level. Since any heavy ball contains at least $h(\sigma) \cdot n$ points from X and $h(\sigma) = \sigma/((\lceil \log(\Delta) \rceil + 1) \cdot 2^{5\lambda+3})$, the total number of heavy balls in level i is at most $1/h(\sigma) = ((\lceil \log(\Delta) \rceil + 1) \cdot 2^{5\lambda+3})/\sigma$. Since M is a doubling metric space with dimension λ, any ball in level i can be decomposed into at most $(2^\lambda)^{\ell(\varepsilon)}$ balls with radius $2^{i-\ell(\varepsilon)}$. Thus, by applying the 2-approximation algorithm for vertex cover, we decompose each heavy ball into at most

$$2 \cdot (2^\lambda)^{\ell(\varepsilon)} = 2 \cdot (2^\lambda)^{\lceil \log(1/\varepsilon) \rceil} \leq \frac{2^{\lambda+1}}{\varepsilon^\lambda}$$

mini balls. It follows that the total number of mini balls in level i is at most

$$\frac{(\lceil \log(\Delta) \rceil + 1) \cdot 2^{5\lambda+3}}{\sigma} \cdot \frac{2^{\lambda+1}}{\varepsilon^\lambda} = \frac{(\lceil \log(\Delta) \rceil + 1) \cdot 2^{6\lambda+4}}{\varepsilon^\lambda \sigma} \quad .$$

Summing up over all levels, we obtain that the total number of mini balls is upper bounded by $(\lceil \log(\Delta) \rceil + 1)^2 \cdot 2^{6\lambda+4}/(\varepsilon^\lambda \sigma)$. □

Lemma 7.3.6. *The algorithm has a running time of* $\mathcal{O}(n^2 \cdot \log(\Delta))$ *and a space requirement of* $\mathcal{O}(n \cdot \log(\Delta))$.

Proof. By applying a standard implementation of the described 2-approximation algorithm for vertex cover, we can decompose the set X into balls with any specified radius in $\mathcal{O}(n^2)$ time. Recall that we assume access to a distance oracle that, given any two points from X, computes in constant time the distance between these two points. Since we have $\lceil \log(\Delta) \rceil + 1$ levels, the total running time to compute all uniform cut decompositions is $\mathcal{O}(n^2 \cdot \log(\Delta))$. Since each uniform cut decomposition is a partition of X, we can find the heavy balls of any level in $\mathcal{O}(n)$ time. Setting the representative points for any level can also be done in $\mathcal{O}(n)$ time. Since there are $\lceil \log(\Delta) \rceil + 1$ levels, the total running time for finding heavy balls and setting representatives is $\mathcal{O}(n \cdot \log(\Delta))$. Thus, the total running time is $\mathcal{O}(n^2 \cdot \log(\Delta))$.

For each level, we store a uniform cut decomposition of X, and, for a subset of X, we store a decomposition into mini balls. Since the number of balls and mini balls per level is less than n, the space requirement of our algorithm is $\mathcal{O}(n \cdot \log(\Delta))$. □

We summarize our results in the following theorem:

Theorem 8 (WSPD with Slack for Doubling Metric Spaces)**.** *Let* $M = (X, \mathrm{D})$ *be an* n*-point metric space with bounded doubling dimension* λ *and spread* Δ*, let* ε*,* $0 < \varepsilon < 1$*, be a separation parameter, and let* σ*,* $(\lceil \log(\Delta) \rceil + 1) \cdot 2^{6\lambda+3}/n < \sigma < 1$*, be a slack parameter. Then, there exists an algorithm that computes a weighted point set* $X' \subset X$ *with cardinality* $\mathcal{O}(2^{\mathcal{O}(\lambda)} \cdot \log^2(\Delta)/(\varepsilon^\lambda \sigma))$ *which is an implicit representation of an* ε*-WSPD with slack* σ *for* M*. The algorithm has a running time of* $\mathcal{O}(n^2 \cdot \log(\Delta))$ *and a space requirement of* $\mathcal{O}(n \cdot \log(\Delta))$.

Proof. We have $X' \subset X$ since the location of each representative is a point from X. Due to Lemma 7.3.4, X' is an implicit representation of an ε-WSPD with slack σ for M. The cardinality of X' follows from Lemma 7.3.5, and the running time and space requirement to compute X' is due to Lemma 7.3.6. In our construction, we made the assumption that $\sigma > (\lceil \log(\Delta) \rceil + 1) \cdot 2^{6\lambda+3}/n$ to ensure that all the balls in $\mathcal{G}(0)$ are light. □

8 Embeddings with Slack in Data Streams and Applications

This chapter is devoted to the problem of computing low-distortion embeddings in the streaming model. Given a stream of points from an n-point metric space M, our streaming algorithms compute an embedding of M into another n-point metric space M' that preserves a $(1-\sigma)$-fraction of all the pairwise distances with small distortion. The parameter σ is called the *slack* of the embedding. The strict space limitations specified by the streaming model prevent us from storing our embedding explicitly. We bypass this obstacle by computing a compact representation of M' without storing the actual mapping from M into M'.

We present streaming embeddings with low distortion and low slack for n-point Euclidean metric spaces in Section 8.2, doubling metric spaces in Section 8.4, and general metric spaces in Section 8.5. The embeddings for Euclidean and doubling metric spaces are based on the techniques developed in Chapter 7. The embedding for general metric spaces takes advantage of the existence of certain subsets of points called *edge-dense nets*. Intuitively, an edge-dense net $N \subseteq X$ of a metric space $M = (X, \mathrm{D})$ has the property that, for a $(1-\sigma)$-fraction of pairs of points $(x, y) \in X \times X$, the distance between N and both x and y is small compared to $\mathrm{D}(x, y)$. The existence of such nets follows from results on embeddings with beacons by Kleinberg et al. [75]. After some modifications, this allows us to compute a low-distortion embedding with low slack in the streaming model. Our method resembles the construction of spanners with slack of Chan et al. [24]. Finally, we prove some lower bounds on the space requirement of streaming embeddings with slack in Section 8.6.

8.1 Preliminaries

A *metric embedding* is the transformation of one metric space into another metric space.

Definition 8.1.1 (Metric Embedding)**.** *A mapping $\varphi : X \to X'$ from a metric space $M = (X, \mathrm{D})$ into a target metric space $M' = (X', \mathrm{D}')$ is called* metric embedding.

In this thesis, we are only interested in embedding metric spaces $M = (X, \mathrm{D})$, where X is a set of n points. Given such an n-point metric space $M = (X, \mathrm{D})$, our streaming algorithms compute an embedding $\varphi : X \to X'$ into a target metric space $M' = (X', \mathrm{D}')$ whose representation uses only sublinear space. Besides the space requirement, we will measure the quality of φ by the quantity of its distortion and slack.

Definition 8.1.2 (Embedding with Distortion and Slack, [23]). *Let $\varrho \geq 1$ be a precision parameter, and let σ, $0 < \sigma < 1$, be a slack parameter. An embedding $\varphi : X \to X'$ from a finite metric space $M = (X, \mathrm{D})$ into a target metric space $M' = (X', \mathrm{D}')$ has* distortion ϱ *and slack σ if there are two values $\alpha, \beta \geq 1$ with $\alpha \cdot \beta \leq \varrho$ such that*

$$\frac{1}{\alpha} \cdot \mathrm{D}(x, y) \leq \mathrm{D}'(\varphi(x), \varphi(y)) \leq \beta \cdot \mathrm{D}(x, y) \tag{8.1}$$

is true for a $(1 - \sigma)$-fraction of pairs $(x, y) \in X \times X$.

Similar to our definition of slack of a WSPD, the slack σ of an embedding φ is measured by the quantity of the fraction of all *ordered* pairs $(x, y) \in X \times X$ that do not satisfy Inequality (8.1). The assumption of having n^2 (instead of $\binom{n}{2}$) pairwise distances simplifies descriptions and makes our proofs cleaner without changing the results in any significant way. In case that an embedding φ has distortion ϱ and slack σ with $\sigma = 0$, we just say that φ has distortion ϱ.

In the following, we will present streaming embeddings for Euclidean, doubling, and general metric spaces. Our algorithms for general and doubling metric spaces work in the insertion-only data stream model, whereas the ones for Euclidean metric spaces work in the dynamic geometric data stream model. In each case, we assume that the minimum pairwise distance of the given metric space is at least 1, and the maximum pairwise distance is at most Δ. Furthermore, we assume that the parameter n is known in advance by our algorithms.

8.2 Embedding Euclidean Metric Spaces

In this section, we explain how to compute with high probability a low-distortion embedding with low slack for a Euclidean metric space $M = (P, \mathrm{D})$ given as a dynamic geometric data stream. Recall that, in this streaming model, the input is a sequence of m Insert and Delete operations of points from a discrete Euclidean space $\{1, \ldots, \Delta\}^d$. At first, we assume that the dimension d is a constant. We will show in Section 8.2.2 how to get rid of this assumption.

8.2.1 Low Dimensions

Our algorithm for constant-dimensional Euclidean spaces is based on the WSPD construction described in Section 7.2. In order to construct an ε-WSPD with slack σ for a point set P, we first use a certain quadtree partition of the point space into a few cells and an elaborate refinement of this partition, where each cell is further subdivided into a few cubelets. Then, we replace each point by a representative such that all the representatives of points located inside of the same cubelet have the same position. Let $R(p)$ be the representative of a point $p \in P$, then $R : P \to P'$ is an embedding from M into the target metric space $M' = (P', \mathrm{D})$. The advantage of this embedding is that it can be computed by using only

the information about the number of points in certain cells or cubelets and is not reliant on the exact location of the points in P. We will show that we can use a random sampling technique to estimate the number of points in the relevant cells and cubelets. To sum up, the idea of our streaming algorithm is to maintain a random sample of the current point set P given as a dynamic geometric data stream and to apply the algorithm described in Section 7.2 on the sample set.

Now, we explain the sample step in more detail. We read the items of the input stream one by one. Each time, we decide whether we use the associated point for further computations or not. For that purpose, we use the technique described in [43, 42] to maintain a sample set of the current point set P with size $s \in \Theta(2^{\Theta(d)} \cdot d^d \cdot \log(\Delta) \log(n/\delta)/(\varepsilon^d \sigma^4))$, where δ is the error probability of the algorithm. We denote this sample set by S.

After the sample step, we execute the quadtree partitioning for S based on the heavy cells in $\lceil \log(\Delta) \rceil + 1$ nested square grids over S as described in Section 7.2. During this process, a cell is identified as heavy if it contains at least $\sigma s/2^{d+1}$ sample points. A cell containing less sample points is identified as light. Thereafter, we build the balanced quadtree partition and perform the refinement into equal sized cubelets as explained in Section 7.2. For each cubelet $\mathcal{C}$ that contains at least $\lceil \ln(n)/\sigma^2 \rceil$ sample points, we replace the points in $\mathcal{C}$ by one representative. This point is set to the location of an arbitrary sample point inside of $\mathcal{C}$ and weighted by $\lceil |\mathcal{C}| \cdot n/s \rceil$, where $|\mathcal{C}|$ denotes the number of replaced points. To avoid that the total weight of the representatives differs from n, we sum up all weights and increase or decrease the weight of some arbitrary representatives by the required amount. The set of all weighted representatives is our compact representation for M'.

Let us first ignore that we use a sample step. Then, due to the fact that the embedding $R : P \to P'$ from M into the target metric space $M' = (P', \mathrm{D})$ is determined by the construction of an ε-WSPD with slack σ for P, the embedding R has distortion $(1 + 2\varepsilon)^2$ and slack σ. In the following, we will show that the sample step, which enables us to compute a representation of M' in the dynamic geometric data stream model, does not significantly increase the slack.

Maintenance of the Sample Set

By applying the technique described in [43, 42], we are able to maintain a sample set of the current point set P under insertions and deletions such that every point in the sample set is chosen nearly uniformly at random from P. More precisely, by adopting the results in [43, 42], we obtain the following lemma:

Lemma 8.2.1 (Sample Data Structure, [43, 42])**.** *Let δ, $0 < \delta \leq 1$, be an error probability parameter. Given a sequence of* Insert *and* Delete *operations of points from the discrete Euclidean space $\{1, \ldots, \Delta\}^d$, there is a data structure that, with probability $1 - \delta$, returns s points $q_0, \ldots, q_{s-1}$ from the current point set $P := \{p_0, \ldots, p_{n-1}\}$ such that*

$$\mathbf{Pr}\left[q_i = p_j\right] = \frac{1}{n} \pm \frac{\delta}{\Delta^d}$$

is true for every $j \in [n]$ and for every $i \in [s]$. Both update time and space requirement of the algorithm are $\mathcal{O}((s + \log(1/\delta)) \cdot d^2 \cdot \log^2(\Delta/\delta))$.

Slack Induced by the Sample Step

Due to the fact that we use a sample set to estimate the number of points in certain cells and cubelets, we make an error which increases the slack. In order to measure this increase of the slack, we first investigate how much the quadtree partition computed on the sample set S might differ from the one that we would get by taking the whole input point set P into account.

Recall that the algorithm identifies each cell that contains at least $\sigma s/2^{d+1}$ sample points as heavy. Next, we show that if each point in S is chosen uniformly at random from P, then, with high probability, the algorithm identifies every heavy cell as heavy and every cell which contains significantly less points than a heavy cell as light.

Lemma 8.2.2. *If each point in S is chosen uniformly at random from P, then every heavy cell is identified as heavy with probability at least $1 - \delta$.*

Proof. Let $\mathcal{H}$ be the set of all heavy cells that do not have a heavy subcell. Obviously, there are at most n cells in $\mathcal{H}$. Let $\mathcal{C}$ be any such cell. Let Y_i be the indicator random variable for the event that the i-th point in S is contained in cell $\mathcal{C}$. Since $\mathcal{C}$ is heavy, it contains at least $\sigma n/2^d$ points. Thus, we have $\mathbf{E}[Y_i] \geq \sigma/2^d$. By a Chernoff bound and linearity of expectation, we get

$$\mathbf{Pr}\left[\sum_{i=1}^{|S|} Y_i < \left(1 - \frac{1}{2}\right) \cdot \mathbf{E}\left[\sum_{i=1}^{|S|} Y_i\right]\right] \leq \exp\left(-\frac{|S| \cdot \mathbf{E}[Y_i]}{2^3}\right) \leq \exp\left(-\frac{\sigma \cdot |S|}{2^{d+3}}\right) .$$

For $|S| \geq 8 \cdot 2^d \ln(n/\delta)/\sigma$, this probability is at most δ/n. Since we have chosen

$$|S| \in \Theta\left(\frac{2^{\Theta(d)} \cdot d^d \cdot \log(\Delta) \log(n/\delta)}{\varepsilon^d \sigma^4}\right) \subset \Omega\left(\frac{2^d \ln(n/\delta)}{\sigma}\right) ,$$

$\mathcal{C}$ contains at least

$$\left(1 - \frac{1}{2}\right) \cdot \mathbf{E}\left[\sum_{i=1}^{|S|} Y_i\right] = \frac{\sigma s}{2^{d+1}}$$

sample points with probability at least $1 - \delta/n$. By the union bound, the probability that every cell in $\mathcal{H}$ contains at least $\sigma s/2^{d+1}$ sample points is at least $1 - \delta$. Obviously, it then follows that each ancestor cell of a cell in $\mathcal{H}$ also contains at least $\sigma s/2^{d+1}$ sample points. Hence, every heavy cell is identified as heavy with probability at least $1 - \delta$. □

We call a cell that contains at least $\sigma n/2^{d+2}$ points from P *quarter-heavy*. The following lemma proves that, with high probability, no cell which is not quarter-heavy is identified as heavy by the algorithm.

Lemma 8.2.3. *If each point in S is chosen uniformly at random from P, then every cell that is not quarter-heavy is identified as light with probability at least $1 - \delta$.*

Proof. Let $\mathcal{C}$ be any cell that contains $\sigma n/(2^{d+2}k)$ points from P, where $k > 1$. Let Y_i be the indicator random variable for the event that the i-th point in S is contained in cell $\mathcal{C}$. We have $\mathbf{E}[Y_i] = \sigma/(2^{d+2}k)$. By a Chernoff bound and linearity of expectation, we get

$$\mathbf{Pr}\left[\sum_{i=1}^{|S|} Y_i \geq (1+k)\cdot\mathbf{E}\left[\sum_{i=1}^{|S|} Y_i\right]\right] \leq \exp\left(-\frac{k\cdot|S|\cdot\mathbf{E}[Y_i]}{3}\right) = \exp\left(-\frac{|S|\cdot\sigma}{3\cdot 2^{d+2}}\right) .$$

For $|S| \geq 12\cdot 2^d \ln(n/\delta)/\sigma$, this probability is at most δ/n. Since we have chosen

$$|S| \in \Theta\left(\frac{2^{\Theta(d)}\cdot d^d\cdot\log(\Delta)\log(n/\delta)}{\varepsilon^d\sigma^4}\right) \subset \Omega\left(\frac{2^d\ln(n/\delta)}{\sigma}\right) ,$$

$\mathcal{C}$ contains less than

$$(1+k)\cdot\mathbf{E}\left[\sum_{i=1}^{|S|} Y_i\right] < \frac{\sigma s}{2^{d+1}}$$

sample points with probability at least $1 - \delta/n$.

Let us assume that the algorithm is performing the quadtree partitioning and has already identified all cells which are at least quarter-heavy as heavy cells and has not yet tried to identify any cell that is not quarter-heavy. Since a cell must contain at least one point from P to be a potential candidate for being identified as heavy, there are at most n such candidate cells in the current space partition. By the union bound, the probability that all of them contain less than $\sigma s/2^{d+1}$ sample points is at least $1 - \delta$. Since each cell contains at most as many sample points as its parent cell, it then follows that every descendant cell of a cell in the current space partition contains less than $\sigma s/2^{d+1}$ sample points. Thus, every cell that is not quarter-heavy contains less than $\sigma s/2^{d+1}$ sample points and is, hence, identified as light with probability at least $1 - \delta$.

□

Due to Lemmas 8.2.2 and 8.2.3, we know that the quadtree partition on S is fairly close to the quadtree partition on P. By utilizing this fact, we next analyze the slack induced by estimating the number of points in cubelets based on the sample set S.

Lemma 8.2.4. *Let $Z \in \Theta(2^{\Theta(d)}\cdot d^d\cdot\log(\Delta)/(\varepsilon^d\sigma))$. If each point in S is chosen uniformly at random from P, then we can define a set $\mathcal{Z}$ of Z cubelets such that the set of cubelets constructed by the algorithm is a subset of $\mathcal{Z}$ with probability at least $1 - 2\delta$.*

Proof. Due to Lemmas 8.2.2 and 8.2.3, with probability at least $1 - 2\delta$, the algorithm satisfies the condition that it identifies every heavy cell as heavy and every cell which is not quarter-heavy as light. Let us now consider the quadtree partition that we get by splitting exactly the quarter-heavy cells and performing the balancing operations. Let $\mathcal{L}^+$ be the set of cells in the resulting balanced quadtree. Then, the set of cells in any balanced

quadtree partition obtained from a run of our algorithm that satisfies the above condition is a subset of $\mathcal{L}^+$. Furthermore, let $\mathcal{Z}$ be the set of cubelets that we obtain by subdividing each cell in $\mathcal{L}^+$ into cubelets. Then, the set of cubelets constructed during any run of the algorithm that satisfies the above condition is a subset of $\mathcal{Z}$. Next, we upper bound the size of $\mathcal{Z}$.

We proceed exactly as we have done in the proof of Lemma 7.2.6. There are at most $2^{d+2}/\sigma$ quarter-heavy cells which do not have a quarter-heavy subcell in a quadtree partition. For each such cell, there exist at most $2^d(\lceil \log(\Delta) \rceil + 1)$ cells in the quadtree. Hence, the unbalanced quadtree partition contains $\mathcal{O}(2^{2d} \cdot \log(\Delta)/\sigma)$ cells. Due to Lemma 7.2.5, the number of cells in the balanced quadtree partition is $\mathcal{O}(24^d \cdot \log(\Delta)/\sigma)$. Since in the last step of the algorithm a cell is subdivided into $\left\lceil 6\sqrt{d} \right\rceil^d$ cubes and each cube into $\left\lceil 2\sqrt{d}/\varepsilon \right\rceil^d$ cubelets, the set $\mathcal{Z}$ consists of $\mathcal{O}(288^d \cdot d^d \cdot \log(\Delta)/(\varepsilon^d \sigma))$ cubelets. □

Lemma 8.2.5. *Let $Z \in \Theta(2^{\Theta(d)} \cdot d^d \cdot \log(\Delta)/(\varepsilon^d \sigma))$, and let U be the union of all the cubelets that contain at most $\sigma n/(2Z)$ points from P. If each point in S is chosen uniformly at random from P and the space partition consists of at most Z cubelets, then, with probability at least $1 - \delta$, the number of points from P in U is at most $\sigma n/2$ and the number of sample points in U is at most σs.*

Proof. Obviously, if there are at most Z cubelets in the space partition, then the number of points in cubelets that contain at most $\sigma n/(2Z)$ points from P is at most $\sigma n/2$. Thus, we have that, for some $k \geq 1$, the total number of points from P in U is $\sigma n/(2k)$. Let Y_i be the indicator random variable for the event that the i-th point in S is contained in U. We have $\mathbf{E}[Y_i] = \sigma/(2k)$. By a Chernoff bound and linearity of expectation, we get

$$\mathbf{Pr}\left[\sum_{i=1}^{|S|} Y_i \geq (1+k) \cdot \mathbf{E}\left[\sum_{i=1}^{|S|} Y_i\right]\right] \leq \exp\left(-\frac{k \cdot |S| \cdot \mathbf{E}[Y_i]}{3}\right) = \exp\left(-\frac{|S| \cdot \sigma}{6}\right) .$$

For $|S| \geq 6\ln(1/\delta)/\sigma$, this probability is at most δ. Since we have chosen

$$|S| \in \Theta\left(\frac{2^{\Theta(d)} \cdot d^d \cdot \log(\Delta)\log(n/\delta)}{\varepsilon^d \sigma^4}\right) \subset \Omega\left(\frac{\ln(1/\delta)}{\sigma}\right) ,$$

U contains less than

$$(1+k) \cdot \mathbf{E}\left[\sum_{i=1}^{|S|} Y_i\right] \leq \sigma s$$

sample points with probability at least $1 - \delta$. □

Lemma 8.2.6. *Let $Z \in \Theta(2^{\Theta(d)} \cdot d^d \cdot \log(\Delta)/(\varepsilon^d \sigma))$. If each point in S is chosen uniformly at random from P, then the number of points in every cubelet that contains at least $\sigma n/(2Z)$ points from P can be $(1 \pm \sigma)$-approximated by S with probability $1 - \delta$.*

Proof. Let $\mathcal{C}$ be any cubelet that contains at least $\sigma n/(2Z)$ points from P. Let Y_i be the indicator random variable for the event that the i-th point in S is contained in cubelet $\mathcal{C}$. We have $\mathbf{E}[Y_i] \geq \sigma/(2Z)$. By Chernoff bounds and linearity of expectation, we get

$$\mathbf{Pr}\left[\sum_{i=1}^{|S|} Y_i \geq (1+\sigma)\cdot\mathbf{E}\left[\sum_{i=1}^{|S|} Y_i\right]\right] \leq \exp\left(-\frac{\sigma^2\cdot|S|\cdot\mathbf{E}[Y_i]}{3}\right) \leq \exp\left(-\frac{\sigma^3\cdot|S|}{6Z}\right)$$

and

$$\mathbf{Pr}\left[\sum_{i=1}^{|S|} Y_i \leq (1-\sigma)\cdot\mathbf{E}\left[\sum_{i=1}^{|S|} Y_i\right]\right] \leq \exp\left(-\frac{\sigma^2\cdot|S|\cdot\mathbf{E}[Y_i]}{2}\right) \leq \exp\left(-\frac{\sigma^3\cdot|S|}{4Z}\right) \quad .$$

For $|S| \geq 6Z\ln(n/\delta)/\sigma^3$, each of these probabilities is at most δ/n. Since we have chosen

$$|S| \in \Theta\left(\frac{2^{\Theta(d)}\cdot d^d\cdot\log(\Delta)\log(n/\delta)}{\varepsilon^d\sigma^4}\right) \subseteq \Omega\left(\frac{Z\ln(n/\delta)}{\sigma^3}\right) \quad ,$$

the number of points in $\mathcal{C}$ can be $(1\pm\sigma)$-approximated with probability $1-2\delta/n$. By the union bound, the number of points in every cubelet that contains at least $\sigma n/(2Z)$ points is $(1\pm\sigma)$-approximated with probability at least $1-\delta$. □

Based on the lemmas given above, we will show that, with high probability, our streaming algorithm computes an ε-WSPD with slack $\sigma' = 4\sigma$ for P. Unfortunately, in contrast to our WSPD construction in a classical non-streaming model, the property that, for each point, the distances to its $\sigma' n$ closest neighbors can be arbitrarily distorted, but the distances to all the other points are $(1\pm 2\varepsilon)$-preserved (see Remark 7.2.4), does not hold for our streaming embedding. This is caused by the use of the random sampling technique. Simply said, we know *how big* the slack is, but we do not know *where* it arises.

Weight of the Representatives

To avoid that the total weight of the representatives differs from n, we adjust the weight of some representatives in the last phase of the algorithm. Now, we prove that this adjustment is small.

Lemma 8.2.7. *Let R be the set of representatives before the adjustment, and let $w(r)$ denote the weight of a representative $r \in R$. Then,*

$$(1-\sigma)\cdot n < \sum_{r\in R} w(r) \leq \left(1+\frac{\sigma^2}{\ln(n)}\right)\cdot n$$

holds with probability at least $1-4\delta$.

Proof. In the third phase, we place one representative in each cubelet $\mathcal{C}$ that contains at least $\lceil\ln(n)/\sigma^2)\rceil$ sample points. The weight of this representative is set to $\lceil|\mathcal{C}|\cdot n/s\rceil$,

where $|\mathcal{C}|$ denotes the number of sample points in $\mathcal{C}$. It follows that the total weight of the representatives can be smaller than n. The sample data structure from Lemma 8.2.1 fails with probability δ. Furthermore, the statistical difference from the exact uniform distribution is at most δ. However, in case that the sample data structure works as required, it follows from Lemma 8.2.4 that, with an error probability of 2δ, the number of cubelets containing less than $\lceil \ln(n)/\sigma^2 \rceil$ sample points is at most $Z \in \mathcal{O}(2^{\mathcal{O}(d)} \cdot d^d \cdot \log(\Delta)/(\varepsilon^d \sigma))$ and

$$Z \leq s \cdot \frac{\sigma^3}{6 \log(n)} \leq \frac{\sigma s}{3 \lceil \ln(n)/\sigma^2 \rceil}$$

for our chosen value of s. Thus, we get

$$\begin{aligned} n - \sum_{r \in R} w(r) &\leq \left\lceil \left\lceil \frac{\ln(n)}{\sigma^2} \right\rceil \cdot \frac{n}{s} \right\rceil \cdot \frac{\sigma s}{3 \lceil \ln(n)/\sigma^2 \rceil} \\ &< 2 \cdot \left\lceil \frac{\ln(n)}{\sigma^2} \right\rceil \cdot \frac{n}{s} \cdot \frac{\sigma s}{3 \lceil \ln(n)/\sigma^2 \rceil} \\ &< \sigma n \end{aligned}$$

with probability at least $1 - 4\delta$, which proves the first inequality of the lemma.

The sum of the weights can be larger than n because the weight of each representative is rounded up to the next integer. Thus, the sum of the weights is at most $n + |R|$. Since $s \leq n$ and every cubelet in which we set a representative contains at least $\lceil \ln(n)/\sigma^2 \rceil$ points of the sample set S, we get

$$\begin{aligned} \sum_{r \in R} w(r) - n &\leq n + |R| - n \\ &\leq \frac{s}{\lceil \ln(n)/\sigma^2 \rceil} \\ &\leq \frac{\sigma^2 n}{\ln(n)} , \end{aligned}$$

which proves the second inequality of the lemma. □

We summarize our results in the following theorem:

Theorem 9. *Given a stream of* Insert *and* Delete *operations of points from a discrete Euclidean space* $\{1, \ldots, \Delta\}^d$*, where* d *is a constant, a precision parameter* ε*,* $0 < \varepsilon < 1$*, a slack parameter* σ*,* $1/o(n) < \sigma < 1$*, and an error probability parameter* δ*,* $0 < \delta < 1$*, there is a randomized streaming algorithm that computes with probability* $1 - \delta$*, for the current point set* P *of size* n*, a point set* $P' \subset P$ *of size* $\mathcal{O}(\log(\Delta)/(\varepsilon^d \sigma))$ *such that* P *embeds into* P' *with distortion* $1 + \varepsilon$ *and slack* σ*. The algorithm has an update time of*

$$\mathcal{O}\left(\frac{\log(\Delta) \cdot \log^2(\Delta/\delta) \cdot \log(n/\delta)}{\varepsilon^{d+1} \sigma^4}\right)$$

and a space requirement of

$$\mathcal{O}\left(\frac{\log(\Delta) \cdot \log^2(\Delta/\delta) \cdot \log(n/\delta)}{\varepsilon^{d} \sigma^4}\right) .$$

Proof. Due to Lemmas 8.2.1 and 8.2.2, with high probability, the algorithm identifies and splits each heavy cell during the quadtree partitioning, which implies that Lemmas 7.2.2 and 7.2.3 are applicable. It follows that, for any two points p_1 and p_2 in P, if the cubelet containing p_1 and the cubelet containing p_2 are not ε-well-separated, then p_2 belongs to the σn closest points of p_1. This induces a slack of σ. Since we estimate the number of points in each cubelet based on the sample set S, we get an additional slack. Due to Lemmas 8.2.4, 8.2.5 and 8.2.6, with high probability, the additional slack induced by this estimation is at most 2σ. Finally, we get more additional slack since we only place representatives in cubelets containing more than a certain threshold of points and we round up the weight of each representative. Due to Lemma 8.2.7, with high probability, this slack is at most σ. Thus, with high probability, our streaming algorithm computes a representation of an ε-WSPD with slack 4σ for P. This ε-WSPD is an implicit embedding of P with distortion $(1+2\varepsilon)^2$ and slack 4σ.

The error probability is given as follows. We use a random sample as given by the data structure from Lemma 8.2.1. This data structure fails with probability δ. Furthermore, the statistical difference from the exact uniform distribution is at most δ. In case that the sample data structure works as required, Lemmas 8.2.2 and 8.2.3 hold with probability at least $1-2\delta$. Thus, the probability that Lemmas 8.2.1, 8.2.2, and 8.2.3 hold is at least $1-4\delta$. If this is the case, then the assertions given in Lemmas 8.2.4, 8.2.7, 7.2.2 and 7.2.3 follow directly and the assertions given in Lemmas 8.2.5 and 8.2.6 hold each with an error probability of at most δ. Thus, the total error probability of our algorithm is at most 6δ.

In summary, if we run our embedding algorithm with a precision parameter $\varepsilon' \leq \varepsilon/5$, a slack parameter $\sigma' \leq \sigma/4$, and an error probability parameter $\delta' \leq \delta/6$, then the embedding has distortion $(1+2\varepsilon')^2 \leq 1+\varepsilon$ and slack $4\sigma' \leq \sigma$ and works with error probability $6\delta' \leq \delta$.

Due to Theorem 7, the size of P' is $\mathcal{O}\left(\log(\Delta)/(\varepsilon^d\sigma)\right)$. Furthermore, it follows from Theorem 7 and $S \subset P$ that we have $P' \subset P$.

The update time is given as follows. At first, we use the data structure of Lemma 8.2.1 to decide if the point is sampled or not. This costs $\mathcal{O}((s+\log(1/\delta)) \cdot \log^2(\Delta/\delta))$ time. Afterwards, we build the balanced quadtree partition and the refinement into a set of $\mathcal{O}(\log(\Delta)/(\varepsilon^d \cdot \sigma))$ cubelets for a set of s points. By applying Theorem 7, this can be done in $\mathcal{O}(s(1/\varepsilon + \log(\Delta)) + \log^2(\Delta)/\sigma + \log(\Delta)/(\varepsilon^d\sigma))$ time. Since the size of the sample set is $s \in \Theta(\log(n/\delta) \cdot \log(\Delta)/(\varepsilon^d \cdot \sigma^4))$, the total update time is as claimed in the theorem.

Due to Lemma 8.2.1, the space required to store the sample data structure is upper bounded by $\mathcal{O}((s+\log(1/\delta)) \cdot \log^2(\Delta/\delta))$. Furthermore, we have to store the partition tree with $\mathcal{O}(\log(\Delta)/(\varepsilon^d \cdot \sigma))$ nodes for the sample set S. This requires $\mathcal{O}(s + \log(\Delta)/(\varepsilon^d\sigma))$ space. Since the size of the sample set is $s \in \Theta(\log(n/\delta) \cdot \log(\Delta)/(\varepsilon^d \cdot \sigma^4))$, the resource for the sampling data structure is dominating. Thus, the total space requirement is as stated in the theorem.

Since we use the WSPD construction given in Section 7.2, we have to make sure that $\sigma' > 2^{2d}/n$ (confer Theorem 7). However, this is implicitly required by the fact that the space requirement of a streaming algorithm has to be sublinear in n and the space requirement of our streaming algorithm is $\omega(1/\sigma)$. □

8.2.2 High Dimensions

If the points in P have a high dimension, we first use the Johnson-Lindenstrauss embedding [71] with $d(\varepsilon, \sigma, \delta) \in \Theta(1/(\varepsilon^2\sigma\delta))$ dimensions to get an embedding into a low-dimensional space that has distortion $1+\varepsilon$ and slack σ with probability $1-\delta$. Afterwards, we apply the techniques described in Sections 7.2 and 8.2 on the low-dimensional point set. This composition of two embeddings, both with distortion $1+\varepsilon$ and slack σ, yields an embedding with distortion $1+3\varepsilon$ and slack 2σ that can be computed in the dynamic geometric data stream model.

In the following, we will give an idea how to use the techniques developed by Johnson and Lindenstrauss [71] to obtain an embedding from a high-dimensional space into a low-dimensional space with distortion $1+\varepsilon$ and slack σ. Overall, we apply the AMS-sketch [6] to get an embedding similar to the Johnson-Lindenstrauss embedding [71]. The main difference between both techniques is as follows. The AMS-sketch computes one random variable for the whole input stream such that the sum of the squared coordinate values is close to the second frequency moment of the input stream with high probability. In contrast, our method computes for each d-dimensional point in the data stream $d(\varepsilon, \sigma, \delta)$ random variables, the $d(\varepsilon, \sigma, \delta)$ coordinates of the embedded point, whose squared values are all equal to the so-called second frequency moment of this d-dimensional input point with high probability. More precisely, we can look upon one d-dimensional point as a stream consisting of d different elements where the frequency of element i, $1 \leq i \leq d$, is given by the value of the i-th coordinate of the d-dimensional point. It is easy to see that, for this definition of frequency moments, the second frequency moment of a point is equal to the squared norm of this point. Because the embedding of a point is given by a linear mapping, the embedded distance vector of two points is equal to the distance vector of the two embedded points. Consequently, our method computes an embedding that preserves an approximation of almost all squared pairwise distances. Hence, the embedding preserves an approximation of almost all simple pairwise distances. Next, we state our result and give a detailed proof of its correctness.

Theorem 10. *Let ε, $0 < \varepsilon < 1$, be a precision parameter, let σ, $0 < \sigma < 1$, be a slack parameter, let δ, $0 < \delta < 1$, be an error probability parameter, and let $d(\varepsilon, \sigma, \delta) := 2/(\varepsilon^2\sigma\delta)$ be a function dependent on ε, σ, and δ. Given a set P of n points in $\mathbb{R}^d$, there exists an embedding $\varphi : P \to \mathbb{R}^{d(\varepsilon,\sigma,\delta)}$ such that*

$$(1-\varepsilon) \cdot \mathrm{D}(p,q) \leq \mathrm{D}(\varphi(p), \varphi(q)) \leq (1+\varepsilon) \cdot \mathrm{D}(p,q)$$

is true for at least $(1-\sigma) \cdot n^2$ pairs of points $(p,q) \in P \times P$ with probability at least $1-\delta$. Each point can be embedded in $\mathcal{O}(d \cdot \log^2(d)/(\varepsilon^2\sigma\delta))$ time using $\mathcal{O}(\log(d)/(\varepsilon^2\sigma\delta))$ space.

Proof. Our proof of the theorem is almost identical to the proof of Theorem 2.2 in [6]. However, since we will present another result that is based on similar techniques, we do not only describe our modifications to the proof of Theorem 2.2 in [6] but include below the full proof.

For each point $p \in P$ and each coordinate $i \in \{1, \ldots, d(\varepsilon, \sigma, \delta)\}$, the algorithm computes a random variable $Y_i(p)$. We define the embedding φ of the point p by

$$\varphi(p) := \frac{1}{\sqrt{d(\varepsilon, \sigma, \delta)}} \cdot (Y_1(p), \ldots, Y_{d(\varepsilon,\sigma,\delta)}(p))^{\mathrm{T}} \ .$$

For each point $p \in P$, the value $Y_i(p)$ is computed in the same way.

Fix an explicit set $V := \{v_1, \ldots, v_Z\}$ of $Z \in \mathcal{O}(d^2)$ vectors of length d with $+1, -1$ entries which are four-wise independent, i.e., for every four distinct coordinates, each of the 16 possible combinations $\{-1, 1\}^4$ occurs uniformly distributed in V. As described in [5, 6], such sets can be constructed with the help of the parity check matrices of BCH codes. The implementation of this construction requires an irreducible polynomial of degree g over the finite field $\mathbb{F}_2$, where 2^g is the smallest power of 2 greater than d. Such a polynomial can be found by using only $\mathcal{O}(\log d)$ space. Then, the construction enables us to compute each coordinate of each vector in V in $\mathcal{O}(\log d)$ space, using a constant number of multiplications in the finite field $\mathbb{F}_{2^g}$ and binary inner products of vectors of length g.

In order to compute $Y_i(p)$, for any $p \in \mathbb{R}^d$, we choose a random vector

$$v_z =: r_i = \left(r_i^{(1)}, r_i^{(2)}, \ldots, r_i^{(d)}\right) \in V \ ,$$

where z is chosen uniformly between 1 and Z. Note that, once we have chosen a random vector r_i to compute the i-th coordinate of $\varphi(p)$ for the first point $p \in P$, we use r_i to compute the i-th coordinate of $\varphi(p')$ for every point $p' \in P$, i.e., we choose $d(\varepsilon, \sigma, \delta)$ random vectors in total. Recall that we denote the i-th coordinate of a point p by $p^{(i)}$. We define

$$Y_i(p) := \sum_{k=1}^{d} r_i^{(k)} \cdot p^{(k)} \ .$$

To compute $\varphi(p) = 1/\sqrt{d(\varepsilon, \sigma, \delta)} \cdot (Y_1(p), \ldots, Y_{d(\varepsilon,\sigma,\delta)}(p))^{\mathrm{T}}$, we have to keep the value z and have to maintain the sum $Y_i(p)$ for each coordinate $i \in \{1, \ldots, d(\varepsilon, \sigma, \delta)\}$. Recall that the bits of each $r_i = v_z$ can be generated from z in $\mathcal{O}(\log(d))$ space, using a constant number of arithmetic and finite field operations on elements of $\mathcal{O}(\log(d))$ bits. Thus, the embedding of one d-dimensional input point requires $\mathcal{O}(\log(d) \cdot d(\varepsilon, \sigma, \delta))$ space and $\mathcal{O}(d \cdot \log^2(d) \cdot d(\varepsilon, \sigma, \delta))$ time. Furthermore, if A denotes the $d(\varepsilon, \sigma, \delta) \times d$ matrix whose rows are the vectors $r_1, \ldots, r_{d(\varepsilon,\sigma,\delta)}$, then we can write $\varphi(p) = 1/\sqrt{d(\varepsilon, \sigma, \delta)} \cdot Ap$. Hence, φ is a linear function, i.e., $\varphi(p - q) = \varphi(p) - \varphi(q)$ for all pairs $(p, q) \in \mathbb{R}^d \times \mathbb{R}^d$.

Let $Y(\nu) := \|\varphi(\nu)\|^2$ be the random variable for the squared length of $\varphi(\nu)$. Due to our definition of $\varphi(\nu)$, we have

$$Y(\nu) = \sum_{i=1}^{d(\varepsilon,\sigma,\delta)} \left(\frac{1}{\sqrt{d(\varepsilon, \sigma, \delta)}} \cdot Y_i(\nu) \right)^2 \ .$$

Next, we show that the expected value of $Y(\nu)$ is $\|\nu\|^2$ and, by bounding the variance of $Y(\nu)$, that $Y(\nu)$ is sharply concentrated.

Due to the fact that the random variables $r_i^{(k)}$ are pairwise independent and $\mathbf{E}\left[r_i^{(k)}\right] = 0$ for all pairs $(i,k) \in \{1,\ldots,d(\varepsilon,\sigma,\delta)\} \times \{1,\ldots,d\}$, we have

$$\begin{aligned}
& \mathbf{E}\left[\left(\frac{1}{\sqrt{d(\varepsilon,\sigma,\delta)}} \cdot Y_i(\nu)\right)^2\right] \\
= \; & \mathbf{E}\left[\left(\frac{1}{\sqrt{d(\varepsilon,\sigma,\delta)}} \cdot \sum_{k=1}^{d} r_i^{(k)} \cdot \nu^{(k)}\right)^2\right] \\
= \; & \sum_{k=1}^{d} \frac{1}{d(\varepsilon,\sigma,\delta)} \cdot \mathbf{E}\left[\left(r_i^{(k)}\right)^2\right] \cdot \left(\nu^{(k)}\right)^2 + \sum_{1 \le k < \ell \le d} \frac{2}{d(\varepsilon,\sigma,\delta)} \cdot \mathbf{E}\left[r_i^{(k)}\right] \cdot \mathbf{E}\left[r_i^{(\ell)}\right] \cdot \nu^{(k)} \cdot \nu^{(\ell)} \\
= \; & \sum_{k=1}^{d} \frac{1}{d(\varepsilon,\sigma,\delta)} \cdot \left(\nu^{(k)}\right)^2 \\
= \; & \frac{1}{d(\varepsilon,\sigma,\delta)} \cdot \|\nu\|^2 \; .
\end{aligned}$$

Due to linearity of expectation, it follows that

$$\mathbf{E}\left[Y(\nu)\right] = \mathbf{E}\left[\sum_{i=1}^{d(\varepsilon,\sigma,\delta)} \left(\frac{1}{\sqrt{d(\varepsilon,\sigma,\delta)}} \cdot Y_i(\nu)\right)^2\right] = \|\nu\|^2 \; .$$

Since the variables $r_i^{(k)}$ are four-wise independent, we have

$$\mathbf{E}\left[\left(\frac{Y_i(\nu)}{\sqrt{d(\varepsilon,\sigma,\delta)}}\right)^4\right] = \sum_{k=1}^{d} \frac{1}{d(\varepsilon,\sigma,\delta)^2} \cdot \left(\nu^{(k)}\right)^4 + \sum_{1 \le k < \ell \le d} \frac{6}{d(\varepsilon,\sigma,\delta)^2} \cdot \left(\nu^{(k)}\right)^2 \cdot \left(\nu^{(\ell)}\right)^2 \; .$$

Furthermore, we obtain

$$\begin{aligned}
\mathbf{E}\left[\left(\frac{Y_i(\nu)}{\sqrt{d(\varepsilon,\sigma,\delta)}}\right)^2\right]^2 &= \left(\sum_{k=1}^{d} \frac{1}{d(\varepsilon,\sigma,\delta)} \cdot \left(\nu^{(k)}\right)^2\right)^2 \\
&= \sum_{k=1}^{d} \frac{1}{d(\varepsilon,\sigma,\delta)^2} \cdot \left(\nu^{(k)}\right)^4 + \sum_{1 \le k < \ell \le d} \frac{2}{d(\varepsilon,\sigma,\delta)^2} \cdot \left(\nu^{(k)}\right)^2 \cdot \left(\nu^{(\ell)}\right)^2 \; .
\end{aligned}$$

It follows that

$$\begin{aligned}
\mathbf{V}\left[\left(\frac{Y_i(\nu)}{\sqrt{d(\varepsilon,\sigma,\delta)}}\right)^2\right] &= \mathbf{E}\left[\left(\frac{1}{\sqrt{d(\varepsilon,\sigma,\delta)}} \cdot Y_i(\nu)\right)^4\right] - \mathbf{E}\left[\left(\frac{1}{\sqrt{d(\varepsilon,\sigma,\delta)}} \cdot Y_i(\nu)\right)^2\right]^2 \\
&= \sum_{1 \le k < \ell \le d} \frac{4}{d(\varepsilon,\sigma,\delta)^2} \cdot \left(\nu^{(k)}\right)^2 \cdot \left(\nu^{(\ell)}\right)^2 \\
&\le 2 \cdot \mathbf{E}\left[\left(\frac{1}{\sqrt{d(\varepsilon,\sigma,\delta)}} \cdot Y_i(\nu)\right)^2\right]^2 \; .
\end{aligned}$$

Now, we can upper bound the variance of $Y[\nu]$ by

$$\begin{aligned}
\mathbf{V}\left[Y(\nu)\right] &= \mathbf{V}\left[\sum_{i=1}^{d(\varepsilon,\sigma,\delta)}\left(\frac{1}{\sqrt{d(\varepsilon,\sigma,\delta)}}\cdot Y_i(\nu)\right)^2\right] \\
&= d(\varepsilon,\sigma,\delta)\cdot\mathbf{V}\left[\left(\frac{1}{\sqrt{d(\varepsilon,\sigma,\delta)}}\cdot Y_i(\nu)\right)^2\right] \\
&\leq d(\varepsilon,\sigma,\delta)\cdot 2\cdot\mathbf{E}\left[\left(\frac{1}{\sqrt{d(\varepsilon,\sigma,\delta)}}\cdot Y_i(\nu)\right)^2\right]^2 \\
&\leq \frac{2\cdot\|\nu\|^4}{d(\varepsilon,\sigma,\delta)} \ .
\end{aligned}$$

Hence, by Chebyshev's inequality and the definition of $d(\varepsilon,\sigma,\delta)$, we get

$$\mathbf{Pr}\left[|Y(\nu)-\mathbf{E}\left[Y(\nu)\right]|>\varepsilon\cdot\|\nu\|^2\right] \leq \frac{\mathbf{V}\left[Y(\nu)\right]}{\varepsilon^2\cdot\|\nu\|^4} = \frac{2}{d(\varepsilon,\sigma,\delta)\cdot\varepsilon^2} = \sigma\delta \ .$$

Let Z_ℓ be the indicator random variable for the event that the distance between the ℓ-th pair of points is not $(1\pm\varepsilon)$-approximated. By the above, we get $\mathbf{E}\left[Z_\ell\right]\leq\sigma\delta$. Then, by Markov's inequality, we have

$$\mathbf{Pr}\left[\sum_{\ell=1}^{n^2} Z_\ell \geq \sigma\cdot n^2\right] \leq \frac{\mathbf{E}\left[\sum_{\ell=1}^{n^2} Z_\ell\right]}{\sigma n^2} \leq \frac{\sigma\delta n^2}{\sigma n^2} = \delta \ .$$

□

By combining the above result with the results from Sections 7.2 and 8.2, we obtain the following theorem:

Theorem 11. *Given a stream of* Insert *and* Delete *operations of points from a discrete Euclidean space* $\{1,\ldots,\Delta\}^d$*, a precision parameter* ε*,* $0<\varepsilon<1$*, a slack parameter* σ*,* $1/o(n)<\sigma<1$*, and an error probability parameter* δ*,* $0<\delta<1/2$*, there is a randomized streaming algorithm that computes with probability* $1-\delta$*, for the current point set* P *of size* n*, a point set* P' *from a discrete Euclidean space* $\{1,\ldots,\Delta'\}^{d'}$ *with spread* $\Delta'\in\mathcal{O}(\sqrt{dn}\Delta/(\varepsilon^2\sqrt{\sigma\delta}))$ *and dimension* $d'\in\Theta(1/(\varepsilon^2\sigma\delta))$ *and of size*

$$\mathcal{O}\left(\left(\frac{1}{\varepsilon\sigma\delta}\right)^{\mathcal{O}(1/(\varepsilon^2\sigma\delta))}\cdot\log(dn\Delta)\right)$$

such that P *embeds into* P' *with distortion* $1+\varepsilon$ *and slack* σ*. The algorithm has an update time of*

$$\mathcal{O}\left(\frac{d\cdot\log^2(d)}{\varepsilon^2\sigma\delta}+\left(\frac{1}{\varepsilon\sigma\delta}\right)^{\mathcal{O}(1/(\varepsilon^2\sigma\delta))}\cdot\log^4(dn\Delta)\right)$$

and a space requirement of

$$\mathcal{O}\left(\log(d)/(\varepsilon^2\sigma\delta) + \left(\frac{1}{\varepsilon\sigma\delta}\right)^{\mathcal{O}(1/(\varepsilon^2\sigma\delta))} \cdot \log^4(dn\Delta)\right) \ .$$

Proof. We combine the embedding from Theorem 10 with the construction described in Sections 7.2 and 8.2. At first, we embed the discrete high-dimensional Euclidean point set P into a low-dimensional Euclidean space. Then, we impose an appropriately fine grid on the target space and move each embedded point to its nearest grid point. This technique is sometimes called *snap rounding*. It follows that, by appropriate scaling of the point space, the resulting point set is from a discrete low-dimensional Euclidean space. On this point set, we apply the construction described in Sections 7.2 and 8.2. Now, we explain our approach in more detail.

By applying the techniques given in the proof of Theorem 10 with a precision parameter $\varepsilon' := \varepsilon/18$, a slack parameter $\sigma' := \sigma/2$, and an error probability parameter $\delta' := \delta/3$, we get an embedding $\varphi : P \to \mathbb{R}^{d(\varepsilon',\sigma',\delta')}$ with $d(\varepsilon',\sigma',\delta') \in \Theta(1/(\varepsilon^2\sigma\delta))$ such that

$$\left(1 - \frac{\varepsilon}{18}\right) \cdot \mathrm{D}(p,q) \leq \mathrm{D}(\varphi(p),\varphi(q)) \leq \left(1 + \frac{\varepsilon}{18}\right) \cdot \mathrm{D}(p,q) \tag{8.2}$$

is true for at least $(1-\sigma/2)\cdot n^2$ pairs of points $(p,q) \in P \times P$ with probability at least $1-\delta/3$. Furthermore, we can upper bound the maximum distance between two embedded points as follows. Let p and q be any two points from P. We define $\nu := p - q$ and $Y(\nu) := \|\varphi(\nu)\|^2$. Then, as explained in the proof of Theorem 10, the expected value of $Y(\nu)$ is $\mathbf{E}[Y(\nu)] = \|\nu\|^2$, and we can upper bound the variance of $Y(\nu)$ by

$$\mathbf{V}[Y(\nu)] \leq \frac{2 \cdot \|\nu\|^4}{d(\varepsilon',\sigma',\delta')} \ .$$

Thus, by Chebyshev's inequality, we get

$$\mathbf{Pr}\left[|Y(\nu) - \mathbf{E}[Y(\nu)]| > n \cdot \|\nu\|^2\right] \leq \frac{\mathbf{V}[Y(\nu)]}{n^2 \cdot \|\nu\|^4} \leq \frac{\delta}{3n^2} \ .$$

Due to the union bound and $\|\nu\|^2 \leq d\Delta^2$, we have that all squared pairwise distances of the embedded points are at most $\mathcal{O}(n \cdot d\Delta^2)$ with probability at least $1-\delta/3$. It follows that the diameter of the embedded point set is $\mathcal{O}(\sqrt{dn}\Delta)$ with probability at least $1-\delta/3$.

Next, we apply the snap-rounding technique. More precisely, we impose a square grid on the target space $\mathbb{R}^{d(\varepsilon',\sigma',\delta')}$, where each cell has side length $\varepsilon/(18\sqrt{d(\varepsilon',\sigma',\delta')})$, and move each embedded point to its nearest grid point. Each point is moved by a distance of at most

$$\frac{\varepsilon}{18\sqrt{d(\varepsilon',\sigma',\delta')}} \cdot \frac{\sqrt{d(\varepsilon',\sigma',\delta')}}{2} = \frac{\varepsilon}{36} \ .$$

Thus, by moving each point to its nearest grid point, the distance between any two points is decreased or increased by at most $\varepsilon/18$. Let $\varphi'(p)$ be the position of an embedded and

moved point $p \in P$. Since the minimum pairwise distance from distinct points in P is 1 and Inequality (8.2) is true for at least $(1 - \sigma/2) \cdot n^2$ pairs of points $(p, q) \in P \times P$ with probability at least $1 - \delta/3$, we have that

$$\left(1 - \frac{\varepsilon}{9}\right) \cdot \mathrm{D}(p, q) \leq \mathrm{D}(\varphi'(p), \varphi'(q)) \leq \left(1 + \frac{\varepsilon}{9}\right) \cdot \mathrm{D}(p, q)$$

is true for at least $(1 - \sigma/2) \cdot n^2$ pairs of points $(p, q) \in P \times P$ with probability at least $1 - \delta/3$. It follows that the embedding φ' has distortion $(1 + \varepsilon/9)/(1 - \varepsilon/9) \leq (1 + \varepsilon/3)$ and slack $\sigma/2$ with probability at least $1 - \delta/3$. Furthermore, each point lies on a grid with cell size $\varepsilon/(18\sqrt{d(\varepsilon', \sigma', \delta')})$ and the maximum pairwise distance of points is $\mathcal{O}(\sqrt{dn}\Delta)$ with probability at least $1 - \delta/3$. Hence, by scaling the point space by $18\sqrt{d(\varepsilon', \sigma', \delta')}/\varepsilon$, we get a set of points from a discrete low-dimensional space $\{1, \ldots, \Delta'\}^{d'}$ with spread $\Delta' \in \mathcal{O}(\sqrt{dn}\Delta/(\varepsilon^2\sqrt{\sigma\delta}))$ and dimension $d' \in \mathcal{O}(1/(\varepsilon^2\sigma\delta))$.

On the obtained point set, we run our construction from Sections 7.2 and 8.2 with a precision parameter $\varepsilon'' := \varepsilon/3$, a slack parameter $\sigma'' := \sigma/2$, and an error probability parameter $\delta'' := \delta/3$. Then, with a total error probability of δ, the resulting point set P' embeds P with distortion $(1+\varepsilon/3)\cdot(1+\varepsilon/3) \leq (1+\varepsilon)$ and slack σ. It follows from the above and Theorem 7 that we also have $P' \subset \{1, \ldots, \Delta'\}^{d'}$ with spread $\Delta' \in \mathcal{O}(\sqrt{dn}\Delta/(\varepsilon^2\sqrt{\sigma\delta}))$ and dimension $d' \in \mathcal{O}(1/(\varepsilon^2\sigma\delta))$.

As explained before in the proof of Theorem 9, we have to ensure that $\sigma'' > 2^{2d}/n$ (confer Theorem 7) since we use the construction given in Section 7.2. However, this is implicitly required by the fact that the space requirement of a streaming algorithm has to be sublinear in n and the space requirement of our streaming algorithm is $\omega(1/\sigma)$.

Finally, we analyze the complexity of our construction. Due to Theorem 10, each point in P can be embedded into the low-dimensional space $\mathbb{R}^{d(\varepsilon', \sigma', \delta')}$ in $\mathcal{O}(d \cdot \log^2(d)/(\varepsilon^2\sigma\delta))$ time using $\mathcal{O}(\log(d)/(\varepsilon^2\sigma\delta))$ space. Due to Theorems 7 and 9, the construction from Sections 7.2 and 8.2 applied on a set of points with dimension $\mathcal{O}(1/(\varepsilon^2\sigma\delta))$ and spread $\mathcal{O}(\sqrt{dn}\Delta/(\varepsilon^2\sqrt{\sigma\delta}))$ has both an update time and space requirement of

$$\mathcal{O}\left(\left(\frac{1}{\varepsilon\sigma\delta}\right)^{\mathcal{O}(1/(\varepsilon^2\sigma\delta))} \cdot \log^4(dn\Delta)\right) .$$

The size of the set of representatives follows from Lemma 7.2.7. □

8.3 Max-Cut in High Dimensions

In this section, we show how to embed a set of high-dimensional Euclidean points into a low-dimensional Euclidean space such that the sum of the pairwise distances is well preserved. Afterwards, we use this result to design a streaming algorithm that implicitly computes a $(1 \pm \varepsilon)$-approximation of the max-cut problem for a dynamic data stream of high-dimensional Euclidean points.

Let $\varphi : P \to \mathbb{R}^{d(\varepsilon,\delta)}$ be the Johnson-Lindenstrauss embedding where each point is mapped into a Euclidean space with dimension $d(\varepsilon,\delta) \in \Theta(1/(\varepsilon^2\delta^2))$. Then, we will show that, for a pair of points $(p,q) \in P \times P$, the expected value of $|\mathrm{D}(\varphi(p),\varphi(q)) - \mathrm{D}(p,q)|$ is $\delta\,\varepsilon \cdot \mathrm{D}(p,q)$ and $|\mathrm{D}(\varphi(p),\varphi(q)) - \mathrm{D}(p,q)|$ is sharply concentrated around its expected value with probability $1-\delta$. This leads to the following lemma:

Lemma 8.3.1. *Let ε, $0 < \varepsilon < 1$, be a precision parameter, let δ, $0 < \delta < 1$, be an error probability parameter, and let $d(\varepsilon,\delta) := 50/(\varepsilon^2\delta^2)$ be a function dependent on ε and δ. Given a set P of n points in $\mathbb{R}^d$, there exists an embedding $\varphi : P \to \mathbb{R}^{d(\varepsilon,\delta)}$ such that*

$$\sum_{(p,q)\in P\times P} |\mathrm{D}(\varphi(p),\varphi(q)) - \mathrm{D}(p,q)| \leq \varepsilon \cdot \sum_{(p,q)\in P\times P} \mathrm{D}(p,q)$$

is true with probability at least $1-\delta$. Each point can be embedded in $\mathcal{O}(d \cdot \log^2(d)/(\varepsilon^2\delta^2))$ time using $\mathcal{O}(\log(d)/(\varepsilon^2\delta^2))$ space.

Proof. For each point $p \in P$ and each coordinate $i \in \{1,\ldots,d(\varepsilon,\delta)\}$, we compute a random variable $Y_i(p)$ as explained in the proof of Theorem 10. We define the embedding φ for the point p by

$$\varphi(p) := \frac{1}{\sqrt{d(\varepsilon,\delta)}} \cdot (Y_1(p),\ldots,Y_{d(\varepsilon,\delta)}(p))^{\mathrm{T}} \quad .$$

Following the construction in the proof of Theorem 10, each point can be embedded using a space of $\mathcal{O}(\log(d)/(\varepsilon^2\delta^2))$ and by performing $\mathcal{O}(d/(\varepsilon^2\delta^2))$ arithmetic and finite field operations on elements of $\mathcal{O}(\log(d))$ bits. Furthermore, since φ is a linear function, we have $\varphi(p-q) = \varphi(p) - \varphi(q)$ for all pairs $(p,q) \in \mathbb{R}^d \times \mathbb{R}^d$.

Now, let p and q be any two points in $\mathbb{R}^d$. We define $\nu := p - q$ and $Y(\nu) := \|\varphi(\nu)\|^2$ to be the random variable for the squared length of $\varphi(\nu)$. Then, as explained in the proof of Theorem 10, the expected value of $Y(\nu)$ is $\mathbf{E}[Y(\nu)] = \|\nu\|^2$, and we can upper bound the variance of $Y(\nu)$ by

$$\mathbf{V}[Y(\nu)] \leq \frac{2 \cdot \|\nu\|^4}{d(\varepsilon,\delta)} \quad .$$

Let $\mathrm{err}(p,q)$ be the error that occurs due to the estimation of $\|\nu\| = \|p-q\|$, i.e.,

$$\mathrm{err}(p,q) := \left| \sqrt{Y(\nu)} - \|\nu\| \right| \quad .$$

The expected value of $\mathrm{err}(p,q)$ is given by

$$\begin{aligned}
& \mathbf{E}[\mathrm{err}(p,q)] \\
\leq\ & \frac{\varepsilon\,\delta}{5} \cdot \|\nu\| + \sum_{i=0}^{\infty} \mathbf{Pr}\left[\frac{\varepsilon\,\delta}{5} \cdot 2^i \cdot \|\nu\| < \left| \sqrt{Y(\nu)} - \|\nu\| \right| \leq \frac{\varepsilon\,\delta}{5} \cdot 2^{i+1} \cdot \|\nu\| \right] \cdot \frac{\varepsilon\,\delta}{5} \cdot 2^{i+1} \cdot \|\nu\| \\
\leq\ & \frac{\varepsilon\,\delta}{5} \cdot \|\nu\| + \sum_{i=0}^{\infty} \mathbf{Pr}\left[\left| \sqrt{Y(\nu)} - \|\nu\| \right| > \frac{\varepsilon\,\delta}{5} \cdot 2^i \cdot \|\nu\| \right] \cdot \frac{\varepsilon\,\delta}{5} \cdot 2^{i+1} \cdot \|\nu\| \quad .
\end{aligned}$$

It follows that, in order to upper bound the expected value of $\mathrm{err}(p,q)$, we have to upper bound the probability that $\mathrm{err}(p,q) > \varepsilon\delta/5 \cdot 2^i \cdot \|\nu\|$ for each $i \in \mathbb{N}_0$. Let ℓ be any fixed power of 2. Then, for $0 \leq \varepsilon\,\delta\,\ell/\,5 \leq 1$, we get

$$\begin{aligned}
& \mathbf{Pr}\left[\left|\sqrt{Y(\nu)} - \|\nu\|\right| > \frac{\varepsilon\,\delta\,\ell}{5} \cdot \|\nu\|\right] \\
= \; & \mathbf{Pr}\left[\sqrt{Y(\nu)} < \left(1 - \frac{\varepsilon\,\delta\,\ell}{5}\right) \cdot \|\nu\| \text{ or } \sqrt{Y(\nu)} > \left(1 + \frac{\varepsilon\,\delta\,\ell}{5}\right) \cdot \|\nu\|\right] \\
= \; & \mathbf{Pr}\left[Y(\nu) < \left(1 - \frac{\varepsilon\,\delta\,\ell}{5}\right)^2 \cdot \|\nu\|^2 \text{ or } Y(\nu) > \left(1 + \frac{\varepsilon\,\delta\,\ell}{5}\right)^2 \cdot \|\nu\|^2\right] \\
\leq \; & \mathbf{Pr}\left[Y(\nu) < \left(1 - \frac{\varepsilon\,\delta\,\ell}{5}\right) \cdot \|\nu\|^2 \text{ or } Y(\nu) > \left(1 + \frac{\varepsilon\,\delta\,\ell}{5}\right) \cdot \|\nu\|^2\right] \\
= \; & \mathbf{Pr}\left[\left|Y(\nu) - \|\nu\|^2\right| > \frac{\varepsilon\,\delta\,\ell}{5} \cdot \|\nu\|^2\right] \quad .
\end{aligned}$$

Similarly, for $\varepsilon\,\delta\,\ell/\,5 > 1$, we get

$$\begin{aligned}
\mathbf{Pr}\left[\left|\sqrt{Y(\nu)} - \|\nu\|\right| > \frac{\varepsilon\,\delta\,\ell}{5} \cdot \|\nu\|\right] \; &= \; \mathbf{Pr}\left[\sqrt{Y(\nu)} > \left(1 + \frac{\varepsilon\,\delta\,\ell}{5}\right) \cdot \|\nu\|\right] \\
&= \; \mathbf{Pr}\left[Y(\nu) > \left(1 + \frac{\varepsilon\,\delta\,\ell}{5}\right)^2 \cdot \|\nu\|^2\right] \\
&\leq \; \mathbf{Pr}\left[Y(\nu) > \left(1 + \frac{\varepsilon\,\delta\,\ell}{5}\right) \cdot \|\nu\|^2\right] \\
&= \; \mathbf{Pr}\left[Y(\nu) - \|\nu\|^2 > \frac{\varepsilon\,\delta\,\ell}{5} \cdot \|\nu\|^2\right] \\
&\leq \; \mathbf{Pr}\left[\left|Y(\nu) - \|\nu\|^2\right| > \frac{\varepsilon\,\delta\,\ell}{5} \cdot \|\nu\|^2\right] \quad ,
\end{aligned}$$

where the first equality follows from the fact that the case $\sqrt{Y(\nu)} < (1 - \varepsilon\,\delta\,\ell/\,5) \cdot \|\nu\| < 0$ cannot occur. Thus, for any value $\varepsilon\,\delta\,\ell/\,5 \in \mathbb{R}$, we have

$$\mathbf{Pr}\left[\left|\sqrt{Y(\nu)} - \|\nu\|\right| > \frac{\varepsilon\,\delta\,\ell}{5} \cdot \|\nu\|\right] \leq \mathbf{Pr}\left[\left|Y(\nu) - \|\nu\|^2\right| > \frac{\varepsilon\,\delta\,\ell}{5} \cdot \|\nu\|^2\right] \quad .$$

By Chebyshev's inequality, we can upper bound this probability by

$$\mathbf{Pr}\left[|Y(\nu) - \|\nu\|^2| > \frac{\varepsilon\,\delta\,\ell}{5} \cdot \|\nu\|^2\right] \leq \frac{25 \cdot \mathbf{V}\left[Y(\nu)\right]}{\varepsilon^2\,\delta^2\,\ell^2\,\|\nu\|^4} \leq \frac{50}{d(\varepsilon,\delta) \cdot \varepsilon^2\,\delta^2\,\ell^2} = \frac{1}{\ell^2} \quad .$$

Now, the expected value of $\mathrm{err}(p,q)$ can be upper bounded by

$$\begin{aligned}
\mathbf{E}\left[\mathrm{err}(p,q)\right] &\leq \frac{\varepsilon\,\delta}{5}\cdot\|\nu\| + \sum_{i=0}^{\infty}\mathbf{Pr}\left[\left|\sqrt{Y(\nu)}-\|\nu\|\right| > \frac{\varepsilon\,\delta}{5}\cdot 2^i\cdot\|\nu\|\right]\cdot\frac{\varepsilon\,\delta}{5}\cdot 2^{i+1}\cdot\|\nu\| \\
&\leq \frac{\varepsilon\,\delta}{5}\cdot\|\nu\| + \sum_{i=0}^{\infty}\frac{1}{2^{2i}}\cdot\frac{\varepsilon\,\delta}{5}\cdot 2^{i+1}\cdot\|\nu\| \\
&= \frac{\varepsilon\,\delta}{5}\cdot\|\nu\| + 2\cdot\sum_{i=0}^{\infty}\frac{1}{2^{i}}\cdot\frac{\varepsilon\,\delta}{5}\cdot\|\nu\| \\
&\leq \varepsilon\,\delta\,\|\nu\| \ .
\end{aligned}$$

Due to Markov's inequality, it follows that

$$\mathbf{Pr}\left[\sum_{p\in P}\sum_{q\in P}\mathrm{err}(p,q) \geq \frac{1}{\delta}\cdot\mathbf{E}\left[\sum_{p\in P}\sum_{q\in P}\mathrm{err}(p,q)\right]\right] \leq \delta \ .$$

Due to linearity of expectation, we have

$$\begin{aligned}
&\mathbf{Pr}\left[\sum_{p\in P}\sum_{q\in P}\mathrm{err}(p,q) \leq \varepsilon\cdot\sum_{p\in P}\sum_{q\in P}\|p-q\|\right] \\
\geq\ &\mathbf{Pr}\left[\sum_{p\in P}\sum_{q\in P}\mathrm{err}(p,q) \leq \frac{1}{\delta}\cdot\sum_{p\in P}\sum_{q\in P}\mathbf{E}\left[\mathrm{err}(p,q)\right]\right] \\
=\ &\mathbf{Pr}\left[\sum_{p\in P}\sum_{q\in P}\mathrm{err}(p,q) \leq \frac{1}{\delta}\cdot\mathbf{E}\left[\sum_{p\in P}\sum_{q\in P}\mathrm{err}(p,q)\right]\right] \\
\geq\ &1-\delta \ .
\end{aligned}$$

□

Given any Euclidean point set P, the embedding described above is useful for all geometric problems that satisfy the following four properties:

(*i*) The cost of an optimal solution for P is a function whose set of input parameters is a subset of all pairwise distances of P.

(*ii*) The cost of an optimal solution for P is at least $\sum_{p\in P}\sum_{q\in P} 1/c\cdot\mathrm{D}(p,q)$, where $c\geq 1$ is any small constant.

(*iii*) If the distance $\mathrm{D}(p,q)$ between any two points $p,q\in P$ is increased or decreased by any value $\alpha>0$, the cost of an optimal solution for P is increased or decreased by at most $\mathcal{O}(\alpha)$.

(*iv*) The complexity of all known $(1\pm\varepsilon)$-approximation algorithms depends exponentially on the dimension of P.

To handle these problems, we first embed the input points and afterwards apply any efficient $(1 \pm \varepsilon)$-approximation algorithm on the embedded points.

One suitable problem is the *max-cut problem* in the dynamic geometric data stream model.

Definition 8.3.2 (Euclidean Max-Cut Problem)**.** *For a set $P \subset \mathbb{R}^d$, the* Euclidean max-cut problem *is to find a partition of P into two subsets C_1 and C_2 such that the sum*

$$\mathrm{Cut}(P, C_1, C_2) := \sum_{(p,q) \in C_1 \times C_2} \mathrm{D}(p, q)$$

of inter-cluster distances is maximized.

Obviously, the max-cut problem satisfies Properties (*i*) and (*iii*). Furthermore, it is shown in [44] that Property (*ii*) is satisfied for $c = 4$. Concerning Property (*iv*), the authors of [44] gave an efficient $(1 \pm \varepsilon)$-approximation for the max-cut problem in low-dimensions that has the following properties:

Lemma 8.3.3 ([44])**.** *Let ε, $0 < \varepsilon < 1$, be a precision parameter. Given a stream of m* Insert *and* Delete *operations of points from a discrete Euclidean space $\{1, \ldots, \Delta\}^d$, where d is a constant, there exists a streaming algorithm that computes with probability at least $2/3$, for the current point set P with cardinality n, a data structure of size $\mathcal{O}(\log^3(\Delta m) \cdot \log^4(\Delta)/\varepsilon^{2d+4})$ from which an implicit $(1 \pm \varepsilon)$-approximate solution for the max-cut problem can be extracted in poly$(\exp(1/\varepsilon)^{\mathcal{O}(1)}, (1/\varepsilon)^d, \log(\Delta), \log(n), \log(m))$ time. An update can be processed in $\mathcal{O}(\log^2(\Delta) \cdot \log(\Delta m))$ time.*

By combining the embedding given in Lemma 8.3.1 with the approximation algorithm presented in [44], we can implicitly compute a $(1 \pm \varepsilon)$-approximation for the max-cut problem on dynamic geometric data streams of high-dimensional points.

Theorem 12. *Let ε, $0 < \varepsilon < 1$, be a precision parameter. Given a stream of m* Insert *and* Delete *operations of points from a discrete high-dimensional Euclidean space $\{1, \ldots, \Delta\}^d$, there is a randomized streaming algorithm that has a space requirement of $\mathcal{O}(\log^7(d\Delta mn)/\varepsilon^{\mathcal{O}(1/\varepsilon^2)})$ and computes with probability at least $5/8$, for the current point set P of size n, a data structure from which an implicit $(1 \pm \varepsilon)$-approximation for the max-cut problem can be extracted in poly$(\exp(1/\varepsilon)^{\mathcal{O}(1)}, (1/\varepsilon)^{1/\varepsilon^2}, \log(d), \log(\Delta), \log(n), \log(m))$ time. An update requires $\mathcal{O}(d \cdot \log^2(d)/\varepsilon^2 + \log^3(d\Delta nm/\varepsilon))$ time.*

Proof. We proceed in a similar way as we have done in the proof of Theorem 11. At first, we embed the discrete high-dimensional Euclidean point set P into a low-dimensional Euclidean space. This embedding induces a small multiplicative error on the cost of a maximum cut. Then, we apply the snap-rounding technique, i.e., we impose an appropriately fine grid on the target space and move each embedded point to its nearest grid point. This movement of the points induces an additive error, which can be charged against a lower bound on the cost of a maximum cut for P to get a small multiplicative error. Finally, by

applying the techniques described in [44] on the embedded and moved points, we obtain the results stated in the theorem. Next, we explain our construction in more detail.

In the first step, we apply the embedding $\varphi : P \to P'$ given in Lemma 8.3.1 with precision parameter $\varepsilon' := \varepsilon/16$ and error probability parameter $\delta' := 1/24$ on P. Then, we have that

$$\sum_{(p,q)\in P\times P} |\mathrm{D}(\varphi(p),\varphi(q)) - \mathrm{D}(p,q)| \leq \varepsilon' \cdot \sum_{(p,q)\in P\times P} \mathrm{D}(p,q)$$

is true with probability at least $1-\delta'$. Since Property (*ii*) (on page 160) is satisfied for $c = 4$ [44], we have $\mathrm{MaxCut}(P) \geq 1/4 \cdot \sum_{(p,q)\in P\times P} \mathrm{D}(p,q)$. Due to the fact that each cut of P is a subset of $(p,q) \in P \times P$, we obtain

$$\begin{aligned}
\left| \sum_{(p,q)\in C_1\times C_2} \mathrm{D}(\varphi(p),\varphi(q)) - \sum_{(p,q)\in C_1\times C_2} \mathrm{D}(p,q) \right| &\leq \sum_{(p,q)\in C_1\times C_2} |\mathrm{D}(\varphi(p),\varphi(q)) - \mathrm{D}(p,q)| \\
&\leq \sum_{(p,q)\in P\times P} |\mathrm{D}(\varphi(p),\varphi(q)) - \mathrm{D}(p,q)| \\
&\leq \varepsilon' \cdot \sum_{(p,q)\in P\times P} \mathrm{D}(p,q) \\
&\leq 4\varepsilon' \cdot \mathrm{MaxCut}(P) \\
&= \frac{\varepsilon}{4} \cdot \mathrm{MaxCut}(P)
\end{aligned}$$

for all cuts (C_1, C_2) of P with probability 23/24. Let (C_1', C_2') be a maximum cut of P, and let (C_1'', C_2'') be any cut of P such that the embedded point sets of C_1'' and C_2'' build a maximum cut of P'. It follows from the above that

$$\begin{aligned}
\sum_{(p,q)\in C_1''\times C_2''} \mathrm{D}(\varphi(p),\varphi(q)) &\geq \sum_{(p,q)\in C_1'\times C_2'} \mathrm{D}(\varphi(p),\varphi(q)) \\
&\geq \sum_{(p,q)\in C_1'\times C_2'} \mathrm{D}(p,q) - \frac{\varepsilon}{4} \cdot \mathrm{MaxCut}(P) \\
&= \left(1 - \frac{\varepsilon}{4}\right) \cdot \mathrm{MaxCut}(P)
\end{aligned}$$

and

$$\begin{aligned}
\sum_{(p,q)\in C_1''\times C_2''} \mathrm{D}(\varphi(p),\varphi(q)) &\leq \sum_{(p,q)\in C_1''\times C_2''} \mathrm{D}(p,q) + \frac{\varepsilon}{4} \cdot \mathrm{MaxCut}(P) \\
&\leq \left(1 + \frac{\varepsilon}{4}\right) \cdot \mathrm{MaxCut}(P) \ .
\end{aligned}$$

Thus, we have

$$\left(1 - \frac{\varepsilon}{4}\right) \cdot \mathrm{MaxCut}(P) \leq \mathrm{MaxCut}(P') \leq \left(1 + \frac{\varepsilon}{4}\right) \cdot \mathrm{MaxCut}(P) \tag{8.3}$$

with probability at least 23/24.

In the second step, we apply the snap-rounding technique. We impose a square grid on the target space $\mathbb{R}^{d(\varepsilon',\delta')}$ with $d(\varepsilon',\delta') \in \Theta(1/(\varepsilon^2\delta^2))$, where each cell has side length $\varepsilon/(16 \cdot \sqrt{d(\varepsilon',\delta')})$, and move each point in P' to its nearest grid point. Let P'' be the set of points that we obtain after moving the points in P'. Each point is moved by a distance of at most

$$\frac{\varepsilon}{16 \cdot \sqrt{d(\varepsilon',\delta')}} \cdot \frac{\sqrt{d(\varepsilon',\delta')}}{2} = \frac{\varepsilon}{32} \ .$$

Thus, the movement of the points induces an additive error of at most $\varepsilon n^2/16$ on the sum of the pairwise distances. Since Property (*ii*) (on page 160) is satisfied for $c = 4$ [44] and the minimum pairwise distance of P is 1, a lower bound on the cost of a maximum cut for P is $n^2/4$. Hence, we have $\varepsilon n^2/16 \leq \varepsilon/4 \cdot \text{MaxCut}(P)$. Due to Inequality (8.3), we get

$$\left(1 - \frac{\varepsilon}{2}\right) \cdot \text{MaxCut}(P) \leq \text{MaxCut}(P'') \leq \left(1 + \frac{\varepsilon}{2}\right) \cdot \text{MaxCut}(P)$$

with probability at least $1 - 1/24$. Besides, we can upper bound the diameter of P'' as follows. Since the maximum pairwise distance of P is $\sqrt{d}\Delta$, the value $n^2 \cdot \sqrt{d}\Delta$ is an upper bound on the cost of a maximum cut for P. Since the diameter of a point set is a lower bound on the cost of a maximum cut of the point set, we get

$$\text{diam}(P') \leq \text{MaxCut}(P') \leq \left(1 + \frac{\varepsilon}{4}\right) \cdot \text{MaxCut}(P) \leq \left(1 + \frac{\varepsilon}{4}\right) \cdot n^2 \cdot \sqrt{d}\Delta \ ,$$

where the second inequality follows from Inequality (8.3). As a result, the diameter of P'' is $\mathcal{O}(\sqrt{d}\Delta n^2)$. Furthermore, each point in P'' lies on a grid with cell size $\varepsilon/(16 \cdot \sqrt{d(\varepsilon',\delta')})$. Thus, by scaling the point space by $16 \cdot \sqrt{d(\varepsilon',\delta')}/\varepsilon$, we get a set of points from a discrete low-dimensional space $\{1, \ldots, \Delta'\}^{d'}$ with $\Delta' \in \mathcal{O}(\sqrt{d}\Delta n^2/\varepsilon^2)$ and $d' \in \mathcal{O}(1/\varepsilon^2)$.

On the scaled point set, we run the approximation algorithm of [44] with precision parameter $\varepsilon'' := \varepsilon/3$. Due to Lemma 8.3.3 and our calculations above, with probability at least $23/24 - 1/3 = 5/8$, we can compute a point set P''' such that

$$\left(1 - \frac{\varepsilon}{2}\right)\left(1 - \frac{\varepsilon}{3}\right) \cdot \text{MaxCut}(P) \leq \frac{\varepsilon \cdot \text{MaxCut}(P''')}{16 \cdot \sqrt{d(\varepsilon',\delta')}} \leq \left(1 + \frac{\varepsilon}{2}\right)\left(1 + \frac{\varepsilon}{3}\right) \cdot \text{MaxCut}(P) \ .$$

Since $(1 - \varepsilon/2)(1 - \varepsilon/3) \geq (1 - \varepsilon)$ and $(1 + \varepsilon/2)(1 + \varepsilon/3) \leq (1 + \varepsilon)$, after rescaling, our construction computes an implicit $(1 \pm \varepsilon)$-approximate solution for the max-cut problem with probability at least 5/8.

Note that our construction works in the streaming model, where the first two steps are used to transform a stream of high-dimensional points into a stream of low-dimensional points. Due to Lemma 8.3.1, the transformation of one high-dimensional input point requires $\mathcal{O}(\log(d)/\varepsilon^2)$ space and $\mathcal{O}(d \cdot \log^2(d)/\varepsilon^2)$ time. Finally, since we apply the approximation algorithm of [44] on a stream of points with dimension $\mathcal{O}(1/\varepsilon^2)$ and spread $\mathcal{O}(\sqrt{d}\Delta n^2/\varepsilon^2)$, the complexity of our construction is as claimed in the theorem. □

8.4 Embedding Doubling Metric Spaces

In this section, we show how to compute a low-distortion embedding with low slack for an n-point doubling metric space $M = (X, \mathrm{D})$ with bounded dimension λ given as a stream of points in the insertion-only data stream model. We assume that the minimum pairwise distance between two points in X is at least 1, and the maximum pairwise distance is at most Δ. Furthermore, we assume access to a distance oracle that, given any two points from X, can compute in constant time the distance between these two points.

The idea of our streaming algorithm is based on the results obtained in Section 7.3. Recall that our WSPD construction from Section 7.3 works as follows. It computes the uniform cut decompositions $\mathcal{G}(0), \ldots, \mathcal{G}(\lceil \log(\Delta) \rceil)$ and decomposes each heavy ball in the uniform cut decompositions into a set of mini balls. Then, the weighted centers of these mini balls are the representatives of the WSPD. The idea of our streaming algorithm is to take two sample sets from the input stream. The first sample set is our set of representatives and is supposed to hit every mini ball that contains more than a certain threshold of points with high probability. The second sample set is supposed to approximate the weight of the mini ball centers. Next, we explain our streaming algorithm in more detail (see Algorithm 8.4.1 for a description in pseudocode).

We take two sample sets from the input stream denoted by R and S. For that purpose, each input point is chosen at random with probability $\mathbf{Pr}[\text{point is taken into } R] := ((\lceil \log(\Delta) \rceil + 1)^2 \cdot 2^{6\lambda+5}) \cdot \ln(n/\delta)/(\varepsilon^\lambda \sigma^2 \cdot n)$ to be a sample point in R, where δ is the error probability of the algorithm. Similarly, each input point is taken at random with probability $\mathbf{Pr}[\text{point is taken into } S] := 6 \cdot (\lceil \log(\Delta) \rceil + 1)^2 \cdot 2^{6\lambda+5} \cdot \ln(n/\delta)/(\varepsilon^\lambda \sigma^4 \cdot n)$ into the sample set S. Let $R := \{y_0, \ldots, y_{k-1}\}$ and $S := \{s_0, \ldots, s_{\ell-1}\}$ be the sample sets after having read the whole input stream. Then, R is our set of representatives for X, and S determines the weight of the representatives in R. More precisely, each point in S is assigned to the nearest representative in R. For each representative $y_i \in R$, let c_i be the total number of points in S that have been assigned to y_i. Then, if $c_i > 0$, we set the weight of y_i to $\lceil c_i \cdot n/|S| \rceil$. Otherwise, we remove y_i from the set of representatives. To avoid that the total weight of the representatives is larger than n, we sum up all weights and decrease the weight of some arbitrary representatives by the required amount. The set of all weighted representatives is our compact representation for M'.

Slack Induced by the Sample Step

Let $\mathcal{M} := \{\mathcal{B}_0, \ldots, \mathcal{B}_{m-1}\}$ be the set of mini balls that we would obtain by running the WSPD construction from Section 7.3 on the n-point metric space $M = (X, \mathrm{D})$. Then, we show that, with high probability, there is at least one sample point from R in each mini ball that contains at least a certain fraction of points from X. Furthermore, for each ball in the same set of mini balls, we show that, with high probability, the number of points inside the ball can be $(1 \pm \sigma)$-approximated by S. Finally, we prove that the remaining mini balls contain only a few points from X as well as from the sample set S.

Algorithm 8.4.1 EMBEDDOUBLINGMETRIC$(n, \Delta, \varepsilon, \sigma, \delta)$

initialize empty point sets R and S
$i \leftarrow 0$
for each point x in the stream **do**
  flip a coin that shows head with probability **Pr** [point is taken into R]
  if coin shows head **then**
    $y_i \leftarrow x$
    $R \leftarrow R \cup y_i$
    initialize counter c_i with 0
    $i \leftarrow i + 1$
  flip a coin that shows head with probability **Pr** [point is taken into S]
  if coin shows head **then**
    $S \leftarrow S \cup x$
for each point $x \in S$ **do**
  compute nearest neighbor y_i in R
  increment counter c_i by 1
for each point $y_i \in R$ **do**
  set weight of y_i to $\lceil c_i \cdot n/|S| \rceil$
return R

In some proofs, we need to know good estimators for the number of points in the sample sets R and S. For this reason, we first give appropriate lower and upper bounds on the sizes of R and S.

Lemma 8.4.1. *If each point in X is taken with probability*

$$\mathbf{Pr}\left[\textit{point is taken into } R\right] := \frac{((\lceil \log(\Delta) \rceil + 1)^2 \cdot 2^{6\lambda+5}) \cdot \ln(n/\delta)}{\varepsilon^\lambda \sigma^2 \cdot n}$$

into the set R, then we have

$$\frac{((\lceil \log(\Delta) \rceil + 1)^2 \cdot 2^{6\lambda+4}) \cdot \ln(n/\delta)}{\varepsilon^\lambda \sigma^2} < |R| < \frac{3 \cdot ((\lceil \log(\Delta) \rceil + 1)^2 \cdot 2^{6\lambda+4}) \cdot \ln(n/\delta)}{\varepsilon^\lambda \sigma^2}$$

with probability $1 - \delta/n$.

Proof. Let Y_i be the indicator random variable for the event that the i-th point in X is taken into the sample set R. We have

$$\mathbf{E}\left[Y_i\right] = \frac{((\lceil \log(\Delta) \rceil + 1)^2 \cdot 2^{6\lambda+5}) \cdot \ln(n/\delta)}{\varepsilon^\lambda \sigma^2 \cdot n} \quad .$$

By a Chernoff bound and linearity of expectation, we get

$$\begin{aligned}\mathbf{Pr}\left[\sum_{i=1}^{|X|} Y_i \geq \left(1+\frac{1}{2}\right)\cdot\mathbf{E}\left[\sum_{i=1}^{|X|} Y_i\right]\right] &\leq \exp\left(-\frac{n\cdot\mathbf{E}\left[Y_i\right]}{12}\right)\\ &\leq \exp\left(-\frac{((\lceil\log(\Delta)\rceil+1)^2\cdot 2^{6\lambda+3})\cdot\ln(n/\delta)}{3\cdot\varepsilon^{\lambda}\sigma^2}\right)\\ &\leq \frac{\delta}{2n}\end{aligned}$$

and

$$\begin{aligned}\mathbf{Pr}\left[\sum_{i=1}^{|X|} Y_i \leq \left(1-\frac{1}{2}\right)\cdot\mathbf{E}\left[\sum_{i=1}^{|X|} Y_i\right]\right] &\leq \exp\left(-\frac{n\cdot\mathbf{E}\left[Y_i\right]}{8}\right)\\ &\leq \exp\left(-\frac{((\lceil\log(\Delta)\rceil+1)^2\cdot 2^{6\lambda+2})\cdot\ln(n/\delta)}{\varepsilon^{\lambda}\sigma^2}\right)\\ &\leq \frac{\delta}{2n}\ .\end{aligned}$$

Thus, we have

$$\left(1-\frac{1}{2}\right)\cdot\mathbf{E}\left[\sum_{i=1}^{|X|} Y_i\right] < \sum_{i=1}^{|X|} Y_i < \left(1+\frac{1}{2}\right)\cdot\mathbf{E}\left[\sum_{i=1}^{|X|} Y_i\right]$$

with probability at least $1-\delta/n$. Now, the assertion follows from $\sum_{i=1}^{|X|} Y_i = |R|$ and $\mathbf{E}\left[\sum_{i=1}^{|X|} Y_i\right] = n\cdot\mathbf{E}\left[Y_i\right]$. □

Lemma 8.4.2. *If each point in X is taken with probability*

$$\mathbf{Pr}\left[\textit{point is taken into } S\right] := \frac{6\cdot(\lceil\log(\Delta)\rceil+1)^2\cdot 2^{6\lambda+5}\cdot\ln(n/\delta)}{\varepsilon^{\lambda}\sigma^4\cdot n}$$

into the set S, then we have

$$\frac{3\cdot(\lceil\log(\Delta)\rceil+1)^2\cdot 2^{6\lambda+5}\cdot\ln(n/\delta)}{\varepsilon^{\lambda}\sigma^4} < |S| < \frac{9\cdot(\lceil\log(\Delta)\rceil+1)^2\cdot 2^{6\lambda+5}\cdot\ln(n/\delta)}{\varepsilon^{\lambda}\sigma^4}$$

with probability $1-\delta/n$.

Proof. The proof runs through with the same approach as used in the proof of Lemma 8.4.1. Let Y_i be the indicator random variable for the event that the i-th point in X is taken into the sample set S. We have

$$\mathbf{E}\left[Y_i\right] = \frac{6\cdot(\lceil\log(\Delta)\rceil+1)^2\cdot 2^{6\lambda+5}\cdot\ln(n/\delta)}{\varepsilon^{\lambda}\sigma^4\cdot n}\ .$$

By a Chernoff bound and linearity of expectation, we obtain

$$\begin{aligned}\mathbf{Pr}\left[\sum_{i=1}^{|X|} Y_i \geq \left(1+\frac{1}{2}\right)\cdot\mathbf{E}\left[\sum_{i=1}^{|X|} Y_i\right]\right] &\leq \exp\left(-\frac{n\cdot\mathbf{E}\left[Y_i\right]}{12}\right)\\ &\leq \exp\left(-\frac{(\lceil\log(\Delta)\rceil+1)^2\cdot 2^{6\lambda+4}\cdot\ln(n/\delta)}{\varepsilon^\lambda\sigma^4}\right)\\ &\leq \frac{\delta}{2n}\end{aligned}$$

and

$$\begin{aligned}\mathbf{Pr}\left[\sum_{i=1}^{|X|} Y_i \leq \left(1-\frac{1}{2}\right)\cdot\mathbf{E}\left[\sum_{i=1}^{|X|} Y_i\right]\right] &\leq \exp\left(-\frac{n\cdot\mathbf{E}\left[Y_i\right]}{8}\right)\\ &\leq \exp\left(-\frac{3\cdot(\lceil\log(\Delta)\rceil+1)^2\cdot 2^{6\lambda+3}\cdot\ln(n/\delta)}{\varepsilon^\lambda\sigma^4}\right)\\ &\leq \frac{\delta}{2n}\ .\end{aligned}$$

Hence, we get

$$\left(1-\frac{1}{2}\right)\cdot\mathbf{E}\left[\sum_{i=1}^{|X|} Y_i\right] < \sum_{i=1}^{|X|} Y_i < \left(1+\frac{1}{2}\right)\cdot\mathbf{E}\left[\sum_{i=1}^{|X|} Y_i\right]$$

with probability at least $1-\delta/n$. Now, the assertion is due to $\sum_{i=1}^{|X|} Y_i = |S|$ and $\mathbf{E}\left[\sum_{i=1}^{|X|} Y_i\right] = n\cdot\mathbf{E}\left[Y_i\right]$. □

The following lemma shows that, with high probability, there is at least one sample point from R in each mini ball that contains at least a certain fraction of points from X.

Lemma 8.4.3. *With probability $1-\delta$, there is at least one sample point from R in each mini ball that contains at least $\varepsilon^\lambda\sigma^2 n/((\lceil\log(\Delta)\rceil+1)^2\cdot 2^{6\lambda+5})$ points from X.*

Proof. Let $\mathcal{B}$ be any mini ball that contains at least $\varepsilon^\lambda\sigma^2 n/((\lceil\log(\Delta)\rceil+1)^2\cdot 2^{6\lambda+5})$ points from X. Then, we have

$$\begin{aligned}&\mathbf{Pr}\left[\mathcal{B}\text{ contains no sample point from }R\right]\\ \leq\ &\left(1-\mathbf{Pr}\left[\text{point is taken into }R\right]\right)^{\frac{\varepsilon^\lambda\sigma^2 n}{(\lceil\log(\Delta)\rceil+1)^2\cdot 2^{6\lambda+5}}}\\ =\ &\left(1-\frac{((\lceil\log(\Delta)\rceil+1)^2\cdot 2^{6\lambda+5})\cdot\ln(n/\delta)}{\varepsilon^\lambda\sigma^2\cdot n}\right)^{\frac{\varepsilon^\lambda\sigma^2 n}{(\lceil\log(\Delta)\rceil+1)^2\cdot 2^{6\lambda+5}}}\\ \leq\ &\delta/n\ ,\end{aligned}$$

where the second inequality is due to a bound on Euler's number (see Inequality (B.2)). Finally, the assertion of the lemma follows by the union bound since we can assume that the number of mini balls is less than n. □

Now, we can show that, with high probability, there are just a few sample points from S located in mini balls that contain less than a certain fraction of points from X. Furthermore, the number of points in each of the remaining mini balls can be $(1\pm\sigma)$-approximated by S with high probability.

Lemma 8.4.4. *Let U be the union of all the mini balls in $\mathcal{M}$ that contain less than $\varepsilon^\lambda\sigma^2 n/((\lceil\log(\Delta)\rceil+1)^2\cdot 2^{6\lambda+5})$ points from X. If $|S|\geq 6\ln(1/\delta)/\sigma$, then, with probability at least $1-\delta$, the number of points from X in U is less than $\sigma n/2$ and the number of sample points from S that are contained in U is at most $\sigma|S|$.*

Proof. As we have shown in the proof of Lemma 7.3.5, the number of mini balls is at most $(\lceil\log(\Delta)\rceil+1)^2\cdot 2^{6\lambda+4}/(\varepsilon^\lambda\sigma)$. Thus, the total number of points in mini balls contained in U is less than

$$\frac{(\lceil\log(\Delta)\rceil+1)^2\cdot 2^{6\lambda+4}}{\varepsilon^\lambda\sigma}\cdot\frac{\varepsilon^\lambda\sigma^2 n}{(\lceil\log(\Delta)\rceil+1)^2\cdot 2^{6\lambda+5}}=\frac{\sigma n}{2}\ .$$

It follows that, for some $t\geq 1$, the total number of points in mini balls from U is $\sigma n/(2t)$. Let Y_i be the indicator random variable for the event that the i-th point in S is contained in U. We have $\mathbf{E}[Y_i]=\sigma/(2t)$. By a Chernoff bound and linearity of expectation, we get

$$\mathbf{Pr}\left[\sum_{i=1}^{|S|}Y_i\geq(1+t)\cdot\mathbf{E}\left[\sum_{i=1}^{|S|}Y_i\right]\right]\leq\exp\left(-\frac{t\cdot|S|\cdot\mathbf{E}[Y_i]}{3}\right)=\exp\left(-\frac{|S|\cdot\sigma}{6}\right).$$

Since we assume that $|S|\geq 6\ln(1/\delta)/\sigma$, this probability is at most δ. Thus, U contains less than

$$(1+t)\cdot\mathbf{E}\left[\sum_{i=1}^{|S|}Y_i\right]\leq\sigma|S|$$

sample points with probability at least $1-\delta$. □

Lemma 8.4.5. *If $|S|\geq 3\cdot(\lceil\log(\Delta)\rceil+1)^2\cdot 2^{6\lambda+5}\cdot\ln(n/\delta)/(\varepsilon^\lambda\sigma^4)$, then the number of points in every mini ball that contains at least $\varepsilon^\lambda\sigma^2 n/((\lceil\log(\Delta)\rceil+1)^2\cdot 2^{6\lambda+5})$ points from X can be $(1\pm\sigma)$-approximated by S with probability $1-\delta$.*

Proof. Let $\mathcal{B}$ be any mini ball that contains at least $\varepsilon^\lambda\sigma^2 n/((\lceil\log(\Delta)\rceil+1)^2\cdot 2^{6\lambda+5})$ points from X. Let Y_i be the indicator random variable for the event that the i-th point in S is contained in the mini ball $\mathcal{B}$. We have $\mathbf{E}[Y_i]\geq\varepsilon^\lambda\sigma^2/((\lceil\log(\Delta)\rceil+1)^2\cdot 2^{6\lambda+5})$. By a Chernoff bound and linearity of expectation, we get

$$\begin{aligned}\mathbf{Pr}\left[\sum_{i=1}^{|S|}Y_i\geq(1+\sigma)\cdot\mathbf{E}\left[\sum_{i=1}^{|S|}Y_i\right]\right] &\leq \exp\left(-\frac{\sigma^2\cdot|S|\cdot\mathbf{E}[Y_i]}{3}\right)\\ &\leq \exp\left(-\frac{\varepsilon^\lambda\sigma^4\cdot|S|}{3\cdot(\lceil\log(\Delta)\rceil+1)^2\cdot 2^{6\lambda+5}}\right)\end{aligned}$$

and

$$\begin{aligned}\mathbf{Pr}\left[\sum_{i=1}^{|S|} Y_i \leq (1-\sigma)\cdot\mathbf{E}\left[\sum_{i=1}^{|S|} Y_i\right]\right] &\leq \exp\left(-\frac{\sigma^2\cdot|S|\cdot\mathbf{E}\left[Y_i\right]}{2}\right)\\ &\leq \exp\left(-\frac{\varepsilon^\lambda\sigma^4\cdot|S|}{(\lceil\log(\Delta)\rceil+1)^2\cdot 2^{6\lambda+6}}\right)\ .\end{aligned}$$

Since we assume that $|S| \geq 3\cdot(\lceil\log(\Delta)\rceil+1)^2\cdot 2^{6\lambda+5}\cdot\ln(n/\delta)/(\varepsilon^\lambda\sigma^4)$, each of these probabilities is at most δ/n. Hence, the number of points in B can be $(1\pm\sigma)$-approximated with probability $1-2\delta/n$. By the union bound, the number of points in every mini ball that contains at least $\varepsilon^\lambda\sigma^2 n/((\lceil\log(\Delta)\rceil+1)^2\cdot 2^{6\lambda+5})$ points from X is $(1\pm\sigma)$-approximated with probability at least $1-\delta$. □

Weight of the Representatives

To avoid that the total weight of the representatives differs from n, we adjust the weight of some representatives. We adopt the result given in Section 8.2 to show that the adjustment is small.

Lemma 8.4.6. *Let R be the set of representatives before the adjustment, and let $w(y_i)$ denote the weight of a representative $y_i \in R$. Then,*

$$n \leq \sum_{y_i\in R} w(y_i) < \left(1+\frac{\sigma^2}{2}\right)\cdot n$$

holds with probability at least $1-2\delta/n$.

Proof. Since the sum of the counters over all representatives is equal to $|S|$ and the weight of a representative is at least its counter multiplied by $n/|S|$, the total weight of the representatives is at least n. This proves the first inequality of the lemma.

The sum of the weights can be larger than n because the weight of each representative is rounded up to the next integer. Thus, the sum of the weights is at most $n+|R|$. Due to Lemma 8.4.1, we have

$$|R| < \frac{3\cdot((\lceil\log(\Delta)\rceil+1)^2\cdot 2^{6\lambda+4})\cdot\ln(n/\delta)}{\varepsilon^\lambda\sigma^2}$$

with probability $1-\delta/n$. Furthermore, due to Lemma 8.4.2, we have

$$|S| > \frac{3\cdot(\lceil\log(\Delta)\rceil+1)^2\cdot 2^{6\lambda+5}\cdot\ln(n/\delta)}{\varepsilon^\lambda\sigma^4}$$

with probability $1-\delta/n$. It follows that $|R| < |S|\cdot\sigma^2/2$ with probability $1-2\delta/n$. Since $|S| \leq n$, we obtain

$$\sum_{y_i\in R} w(y_i) - n \leq n + |R| - n < \frac{\sigma^2}{2}\cdot|S| \leq \frac{\sigma^2}{2}\cdot n\ ,$$

which proves the second inequality of the lemma. □

We summarize our results in the following theorem:

Theorem 13. *Given a stream of points from an n-point doubling metric space $M = (X, \mathrm{D})$ with bounded dimension λ, a precision parameter ε, $0 < \varepsilon < 1$, a slack parameter σ, $1/o(n) < \sigma < 1$, and an error probability parameter δ, $0 < \delta < 1$, there is a randomized streaming algorithm that computes with probability $1 - \delta$ a set $X' \subset X$ of cardinality $\mathcal{O}(\log^2(\Delta) \cdot \log(n/\delta)/(\varepsilon^\lambda \sigma^2))$ such that $M = (X, \mathrm{D})$ embeds into $M' = (X', \mathrm{D})$ with distortion $1 + \varepsilon$ and slack σ. The algorithm requires $\mathcal{O}(\log^2(\Delta) \cdot \log(n/\delta)/(\varepsilon^\lambda \sigma^4))$ space and has a constant update time. The set X' can be extracted in $\mathcal{O}(\log^4(\Delta) \cdot \log^2(n/\delta)/(\varepsilon^{2\lambda} \sigma^6))$ time.*

Proof. Due to Lemma 8.4.3, with high probability, there is at least one representative in each mini ball that contains at least $\varepsilon^\lambda \sigma^2 n/((\lceil \log(\Delta) \rceil + 1)^2 \cdot 2^{6\lambda+5})$ points from X. Let $\mathcal{B}(x, r)$ be such a mini ball. Then, each point $x' \in S \cap \mathcal{B}(x, r)$ is either assigned to a representative in $\mathcal{B}(x, r)$ or to another representative which is closer to x'. It follows that the representative to which we assign x' is contained in the ball $\mathcal{B}(x, 3r)$. Hence, due to Lemmas 7.3.3 and 7.3.4, the representatives for points located in mini balls that contain at least $\varepsilon^\lambda \sigma^2 n/((\lceil \log(\Delta) \rceil + 1)^2 \cdot 2^{6\lambda+5})$ points from X form an 3ε-WSPD with slack σ for X. Since we estimate the number of points in the mini balls based on the sample set S, we get an additional slack. Due to Lemmas 8.4.2, 8.4.4 and 8.4.5, with high probability, the additional slack induced by this estimation is at most 2σ. Furthermore, we get another additional slack of $\sigma^2/2$ (see Lemma 8.4.6) since we round up the weight of each representative. Thus, with high probability, our streaming algorithm computes an 3ε-WSPD with slack 4σ for X. Due to our construction, this WSPD is an embedding for X with distortion $(1 + 3\varepsilon)^2$ and slack 4σ.

Due to Lemma 8.4.1, we can assume with high probability that the set of representatives R and, hence, the set X' has a cardinality of $\mathcal{O}(\log^2(\Delta) \cdot \log(n/\delta)/(\varepsilon^\lambda \sigma^2))$. Furthermore, due to Lemma 8.4.2, we can assume with high probability that the set S has a cardinality of $\mathcal{O}(\log^2(\Delta) \cdot \log(n/\delta)/(\varepsilon^\lambda \sigma^4))$. Since we only store the two sample sets R and S, the total space requirement of our algorithm is $\mathcal{O}(\log^2(\Delta) \cdot \log(n/\delta)/(\varepsilon^\lambda \sigma^4))$.

The error probability is given as follows. Lemmas 8.4.1 and 8.4.2 hold with a total error probability of at most $2\delta/n$. If this is the case, then the assertions given in Lemmas 8.4.4 and 8.4.5 follow with a total error probability of 2δ. The assertion of Lemma 8.4.3 is true with probability $1 - \delta$. Thus, the total error probability of our algorithm is at most 4δ.

Overall, if we run our embedding algorithm with a precision parameter $\varepsilon' \leq \varepsilon/9$, a slack parameter $\sigma' \leq \sigma/4$, and an error probability parameter $\delta' \leq \delta/4$, then the embedding has distortion $(1 + 3\varepsilon')^2 \leq 1 + \varepsilon$ and slack $4\sigma' \leq \sigma$ and works with error probability $4\delta' \leq \delta$.

Since we can decide in constant time whether a point is taken into one or both of the two sample sets, the algorithm has a constant update time. To extract the weighted set X', we assign each point in S to its nearest neighbor in R. Since we assume access to a distance oracle that, given any two points from X, can compute in constant time the distance between these two points, the assignment of S can be done in $|R| \cdot |S|$ time.

Since we use the construction from Section 7.3 in the analysis of our streaming algorithm, we have to ensure that $\sigma' > (\lceil \log(\Delta) \rceil + 1) \cdot 2^{3\lambda}/n$ (confer Theorem 8). However, this is

implicitly required by the fact that the space requirement of a streaming algorithm has to be sublinear in n and the space requirement of our streaming algorithm is $\omega(1/\sigma)$. □

8.5 Embedding General Metric Spaces

This section deals with a streaming algorithm that embeds a general n-point metric space $M = (X, \mathrm{D})$ with constant distortion and slack σ into a metric space $M' = (X', \mathrm{D}')$. As in the previous sections, we assume that the minimum pairwise distance of M is at least 1, and the maximum pairwise distance is at most Δ. Furthermore, we assume access to a distance oracle that, given any two points from X, can compute in constant time the distance between these two points.

Our algorithm works in the insertion-only data stream model and resembles the construction of spanners with slack proposed by Chan et al. [24]. A spanner with slack σ for M is a sparse graph G whose vertices are the points in X and whose shortest-path metric approximates a $(1-\sigma)$-fraction of all pairwise distances of M with small distortion. The first step of the spanner construction presented in [24] is the computation of a small *edge-dense net* $N \subset X$ of M. Intuitively, N has the property that, for a large fraction of pairs of points $(x, y) \in X \times X$, the distance between N and both x and y is small compared to $\mathrm{D}(x, y)$. Based on N, the edges of G are constructed as follows. For each pair of points $(x, y) \in N \times N$, an edge with length $\mathrm{D}(x, y)$ is added to G. For each point $x \in X \backslash N$, its closest neighbor y in N is determined and an edge with length $\mathrm{D}(x, y)$ is added to G.

Now, we transform the construction to the streaming model. Our first modification is that we replace the edge-dense net N by a sample set S drawn uniformly at random from X, i.e., G contains a clique S and each point $x \in X \backslash S$ is connected to its closest neighbor in S. We will show that, if the size of S is chosen carefully, this modification changes the properties of G only slightly. Secondly, instead of storing for each point $x \in X \backslash S$ an edge to a point in S, we store for each point $x' \in S$ the number of points at each distance scale that have x' as their nearest neighbor. This technique has been earlier applied by Czumaj and Sohler [32] to obtain 2-pass streaming algorithms for clustering problems. Since our streaming algorithm has to get along with one pass and after having read only a part of the input stream one cannot know the nearest neighbor of a point $x \in X$ in the final sample set S, we compute the nearest neighbor in the current sample set S, at the point of time when x appears in the stream. In this way, we are able to compute a compact representation of a spanner with slack for M in the streaming model. Next, we describe our streaming algorithm in more detail. A description in pseudocode is given by Algorithm 8.5.1.

We read the points of the input stream one by one and sample each point with probability $\mathbf{Pr}\left[\text{point is sampled}\right] := m \log(n/\delta)(\lceil \log(\Delta) \rceil + 1)/(\sigma^2 n)$, where δ is the error probability of the algorithm and m is the size of the edge-dense net N mentioned above (which is a constant depending on σ). Let $S := \{s_0, \ldots, s_{k-1}\}$ be the set of sampled points. For each $i \in [k]$, we maintain counters $c_{i,0}, c_{i,1}, \ldots, c_{i,\lceil \log(\Delta) \rceil}$, which are initially set to 0. Moreover, for each point $x \in X \backslash S$, we compute its nearest neighbor $\tau(x) = s_i$ in S, at the point of time when x appears in the stream, and we increment $c_{i,j}$, where $j = \lceil \log(\mathrm{D}(x, s_i)) \rceil$. By

storing the points in S and the counters $c_{i,j}$, we implicitly store the following metric space M'. The metric M' is the shortest-path metric of a graph G with vertex set X. For each pair of points $(s_i, s_j) \in S \times S$, the graph G contains an edge $\{s_i, s_j\}$ of length $\mathrm{D}(s_i, s_j)$. For each point $x \in X \backslash S$, the graph G contains an edge $\{x, \tau(x)\}$ of length $2^{\lceil \log(\mathrm{D}(x,\tau(x))) \rceil}$. We denote the resulting embedding by φ. Note that we do not store the mapping $\varphi : X \to X'$ since this would require $\Omega(n)$ space.

Algorithm 8.5.1 EmbedGeneralMetric$(n, \Delta, \sigma, \delta)$

```
initialize empty point set S
i ← 0
for each point x in the stream do
    flip a coin that shows head with probability Pr[point is sampled]
    if coin shows head then
        s_i ← x
        S ← S ∪ s_i
        for j ← 0 to ⌈log(Δ)⌉ do
            initialize counter c_{i,j} with 0
        i ← i + 1
    else
        compute nearest neighbor τ(x) = s_{i'} in S
        increment counter c_{i',⌈log(D(x,s_{i'}))⌉} by 1
return points in S together with their counters
```

Analysis of the Embedding

In order to prove that the embedding φ has constant distortion and slack σ, we first show that M indeed contains some small edge-dense net N.

Definition 8.5.1 (Edge-Dense Net)**.** *Let $M = (X, \mathrm{D})$ be any general metric space, let $\gamma > 0$ be any precision parameter, and let σ, $0 < \sigma < 1$, be any slack parameter. We say that a subset $N \subset X$ is a* (σ, γ)-edge-dense net *for M if, for at least a $(1 - \sigma)$-fraction of pairs $(x, y) \in X \times X$, there exists a pair $(b_x, b_y) \in N \times N$ such that*

$$\max\{\mathrm{D}(x, b_x), \mathrm{D}(y, b_y)\} \leq \gamma \cdot \mathrm{D}(x, y) \ .$$

Lemma 8.5.2 ([75])**.** *For any general metric space $M = (X, \mathrm{D})$ and for any slack parameter σ, $0 < \sigma < 1$, there exists a subset $N \subset X$ with $|N| = C(\sigma)$, where $C(\sigma)$ is a constant depending on σ, such that*

$$\min_{b \in N}\{\mathrm{D}(x, b), \mathrm{D}(y, b)\} \leq \mathrm{D}(x, y)$$

is true for at least a $(1 - \sigma)$-fraction of pairs $(x, y) \in X \times X$.

We reformulate the above lemma as follows.

Lemma 8.5.3. *For any general metric space $M = (X, \mathrm{D})$ and for any slack parameter σ, $0 < \sigma < 1$, there exists a $(\sigma, 2)$-edge-dense net $N \subset X$ with $|N| = C(\sigma)$, where $C(\sigma)$ is a constant depending on σ.*

Proof. Let N be the set given by Lemma 8.5.2. Then, for at least a $(1-\sigma)$-fraction of pairs $(x, y) \in X \times X$, there exists an element $b \in N$ such that

$$\min_{b \in N} \{\mathrm{D}(x, b), \mathrm{D}(y, b)\} \leq \mathrm{D}(x, y) \ .$$

Without loss of generality, we assume that $\mathrm{D}(x, b) \leq \mathrm{D}(x, y)$. By triangle inequality, we have

$$\mathrm{D}(y, b) \leq \mathrm{D}(y, x) + \mathrm{D}(x, b) \leq 2 \cdot \mathrm{D}(x, y) \ ,$$

and the assertion follows. □

Now, let $N := \{z_0, \ldots, z_{m-1}\}$ be a $(\sigma, 2)$-edge-dense net for the input metric space M. For each $\ell \in [m]$, let X_ℓ be the set of points in X for which the nearest neighbor in N is z_ℓ (breaking ties arbitrarily). Furthermore, for each $j \in [\lceil \log(\Delta) \rceil + 1]$, we define

$$X_{\ell,j} := \left\{ x \in X_\ell \mid \mathrm{D}(x, z_\ell) \in \left(2^{j-1}, 2^j\right] \right\} \ .$$

We say that $X_{\ell,j}$ is *good* if after $\sigma |X_{\ell,j}|$ points from $X_{\ell,j}$ have appeared in the stream, the set S contains at least one point from $X_{\ell,j}$. In case this condition fails, we say that $X_{\ell,j}$ is *bad*. The next lemma shows that each set $X_{\ell,j}$ that contains more than a certain threshold of points is good with high probability.

Lemma 8.5.4. *With probability at least $1 - \delta$, for each $\ell \in [m]$ and each $j \in [\lceil \log(\Delta) \rceil + 1]$ with $|X_{\ell,j}| \geq \sigma n / (m(\lceil \log(\Delta) \rceil + 1))$, $X_{\ell,j}$ is good.*

Proof. Pick any $\ell \in [m]$ and any $j \in [\lceil \log(\Delta) \rceil + 1]$ such that

$$|X_{\ell,j}| \geq \frac{\sigma n}{m(\lceil \log(\Delta) \rceil + 1)} \ .$$

Then, we have

$$\begin{aligned} \mathbf{Pr}\left[X_{\ell,j} \text{ is bad}\right] &= \left(1 - \mathbf{Pr}\left[\text{point is sampled}\right]\right)^{\sigma |X_{\ell,j}|} \\ &\leq \left(1 - \frac{m \log(n/\delta)(\lceil \log(\Delta) \rceil + 1)}{\sigma^2 n}\right)^{\frac{\sigma^2 n}{m(\lceil \log(\Delta) \rceil + 1)}} \\ &< \delta / n \ , \end{aligned}$$

where the second inequality is due to a bound on Euler's number (see Inequality (B.2)). Finally, we obtain the assertion of the lemma by applying the union bound over all pairs $(\ell, j) \in [m] \times [\lceil \log(\Delta) \rceil + 1]$. Recall that m is a constant depending on σ (see Lemma 8.5.3), so we can assume that $m \cdot (\lceil \log(\Delta) \rceil + 1) \leq n$. □

The results above facilitates us to bound the distortion and the slack of our embedding φ in a sufficient way.

Lemma 8.5.5. *With probability at least $1-\delta$, for at least a $(1-3\sigma)$-fraction of pairs $(x,y) \in X \times X$, we have*

$$\mathrm{D}(x,y) \leq \mathrm{D}'(\varphi(x),\varphi(y)) \leq 46 \cdot \mathrm{D}(x,y) \ .$$

Proof. The number of points contained in sets $X_{\ell,j}$ with $|X_{\ell,j}| < \sigma n/(m(\lceil \log(\Delta) \rceil + 1))$ is at most σn. Now, by Lemmas 8.5.2, 8.5.3, and 8.5.4 and since N is a $(\sigma, 2)$-edge-dense net, it follows with probability at least $1-\delta$ that, for at least a $(1-3\sigma)$-fraction of pairs $(x,y) \in X \times X$, there exist $b_x, b_y \in N$, and $x', y' \in S$ (see Figure 8.1) such that:

- x' and y' appear in the stream before x and y, respectively,
- $\mathrm{D}(x,b_x) \leq \mathrm{D}(x,y)$ or $\mathrm{D}(y,b_y) \leq \mathrm{D}(x,y)$,
- $\max\{\mathrm{D}(x,b_x), \mathrm{D}(y,b_y)\} \leq 2 \cdot \mathrm{D}(x,y)$,
- $\mathrm{D}(x',b_x) \leq 2 \cdot \mathrm{D}(x,b_x)$, and $\mathrm{D}(y',b_y) \leq 2 \cdot \mathrm{D}(y,b_y)$.

Without loss of generality, we assume that $\mathrm{D}(x,b_x) \leq \mathrm{D}(x,y)$. Now, for a pair $(x,y) \in X \times X$, we get

$$\begin{aligned}
& \mathrm{D}'(\varphi(x),\varphi(y)) \\
= \ & 2^{\lceil \log(\mathrm{D}(x,\tau(x))) \rceil} + \mathrm{D}(\tau(x),\tau(y)) + 2^{\lceil \log(\mathrm{D}(y,\tau(y))) \rceil} \\
\leq \ & 2^{\lceil \log(\mathrm{D}(x,x')) \rceil} + \mathrm{D}(\tau(x),x) + \mathrm{D}(x,x') + \mathrm{D}(x',y') + \mathrm{D}(y',y) + \mathrm{D}(y,\tau(y)) + 2^{\lceil \log(\mathrm{D}(y,y')) \rceil} \\
\leq \ & 2^{\lceil \log(\mathrm{D}(x,x')) \rceil} + 2 \cdot \mathrm{D}(x,x') + \mathrm{D}(x',y') + 2 \cdot \mathrm{D}(y',y) + 2^{\lceil \log(\mathrm{D}(y,y')) \rceil} \\
\leq \ & 4 \cdot \mathrm{D}(x,x') + \mathrm{D}(x',y') + 4 \cdot \mathrm{D}(y,y') \\
\leq \ & 4 \cdot (\mathrm{D}(x,b_x) + \mathrm{D}(b_x,x')) + \mathrm{D}(x',y') + 4 \cdot (\mathrm{D}(y,b_y) + \mathrm{D}(b_y,y')) \\
\leq \ & 12 \cdot \mathrm{D}(x,b_x) + \mathrm{D}(x',y') + 12 \cdot \mathrm{D}(y,b_y) \\
\leq \ & 12 \cdot \mathrm{D}(x,b_x) + \mathrm{D}(x',b_x) + \mathrm{D}(b_x,x) + \mathrm{D}(x,y) + \mathrm{D}(y,b_y) + \mathrm{D}(b_y,y') + 12 \cdot \mathrm{D}(y,b_y) \\
\leq \ & 15 \cdot \mathrm{D}(x,b_x) + \mathrm{D}(x,y) + 15 \cdot \mathrm{D}(y,b_y) \\
\leq \ & 46 \cdot \mathrm{D}(x,y) \ .
\end{aligned}$$

□

By combining our results, we obtain the following theorem:

Theorem 14. *Let σ, $0 < \sigma < 1$, be a slack parameter, and let δ, $0 < \delta < 1$, be an error probability parameter. Given a stream of points from any general n-point metric space M, there exists a randomized streaming algorithm that computes with probability at least $1-\delta$ an implicit representation of an n-point metric space M' such that M embeds into M' with distortion $\mathcal{O}(1)$ and slack σ. The algorithm requires $\mathcal{O}(C(\sigma) \cdot \log(n/\delta) \cdot \log(n) \cdot \log^2(\Delta)/\sigma^2)$ space, where $C(\sigma)$ is a constant depending on σ.*

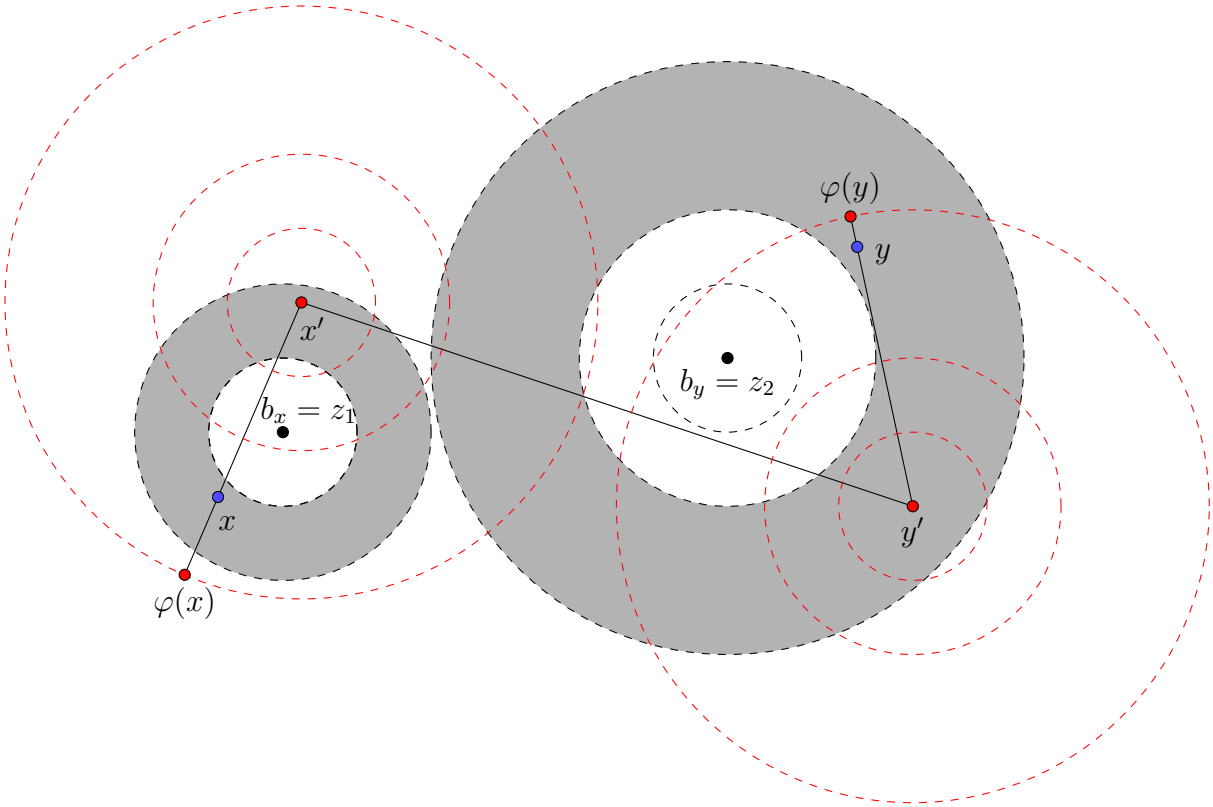


Figure 8.1: Illustration of the embedding φ for a set of points in the Euclidean plane. The points z_1 and z_2 belong to the edge-dense net N. The areas which contain the sets $X_{1,1}$ and $X_{2,2}$ are colored in gray. Both sets are good, which implies that $X_{1,1}$ contains a sample point x' and $X_{2,2}$ contains a sample point y'. The distance scale for each sample point is indicated by the dashed red circles. The distance between x and y is represented by $\mathrm{D}(\varphi(x), x') + \mathrm{D}(x', y') + \mathrm{D}(y', \varphi(y))$, which is indicated by the line segments.

Proof. First, we compute an upper bound on the space requirement of our algorithm. Let Y_i be the indicator random variable for the event that the i-th point in the data stream is sampled. Recall that each point is sampled with probability $\mathbf{Pr}[\text{point is sampled}] = C(\sigma)\log(n/\delta)(\lceil\log(\Delta)\rceil + 1)/(\sigma^2 n)$. Thus, we have $\mathbf{E}[Y_i] = \mathbf{Pr}[\text{point is sampled}]$. By a Chernoff bound, we get

$$\begin{aligned}\mathbf{Pr}\left[\sum_{i=1}^{|X|} Y_i \geq 4 \cdot \mathbf{E}\left[\sum_{i=1}^{|X|} Y_i\right]\right] &\leq \exp\left(-\frac{3 \cdot n \cdot \mathbf{E}[Y_i]}{3}\right) \\ &\leq \exp\left(-n \cdot \mathbf{Pr}[\text{point is sampled}]\right) \\ &= \delta \ .\end{aligned}$$

It follows that, with probability at least $1 - \delta$, the size of the sample set S is

$$|S| \in \mathcal{O}\left(\frac{C(\sigma) \cdot \log(n/\delta) \cdot \log(\Delta)}{\sigma^2}\right) \ .$$

Since our algorithm only stores each point in S together with its $\mathcal{O}(\log(\Delta))$ counters and each counter is set to at most n, the total space requirement is as claimed.

Due to Lemma 8.5.5 and the considerations above, the embedding φ has distortion $\mathcal{O}(1)$ and slack 3σ and works with a total error probability of at most 2δ. Thus, if we run our algorithm with a slack parameter $\sigma' \leq \sigma/3$ and an error probability parameter of $\delta' \leq \delta/2$, the resulting embedding has slack $3\sigma' \leq \sigma$ and an error probability of $2\delta' \leq \delta$. □

8.6 Lower Bounds

In this section, we derive two lower bounds. First, we show that any algorithm that embeds an n-point metric space M into another metric space M' with distortion $\varrho < 2$ and slack $\sigma < 1/4$ requires $\Omega(n/\log n)$ bits of memory. The second lower bound depends on the spread Δ of M. More precisely, we prove that any algorithm that embeds a metric space M into another metric space M' with distortion $\varrho < 2$ and slack $\sigma < 1/4$ needs $\Omega(\log\log\Delta)$ bits of memory. Both proofs are based on the pigeonhole principle. We show that if we restrict the memory space to a certain number of bits, there cannot exist a so-called (σ, ϱ)-*net*.

Definition 8.6.1 (Net for Metric Spaces)**.** *Let $\varrho \geq 1$ be a precision parameter, and let σ, $0 < \sigma < 1$, be a slack parameter. A set of n-point metric spaces N is called a (σ, ϱ)-net if every n-point metric space M embeds into some metric space $M' \in N$ with distortion ϱ and slack σ.*

Theorem 15. *Let ϱ, $1 \leq \varrho < 2$, be a precision parameter, and let σ, $0 \leq \sigma < 1/4$, be a slack parameter. Then, any algorithm that computes for every arbitrary n-point metric space $M = (X, \mathrm{D})$ with positive probability an (implicit or explicit) representation of another metric space $M' = (X', \mathrm{D}')$ such that there is an embedding from M to M' with distortion ϱ and slack σ requires $\Omega(n/\log n)$ bits of memory.*

Proof. We use the probabilistic method to prove the assertion. In general, the probabilistic method says that if an object chosen at random from a given universe satisfies a certain property with positive probability, then there must exist an object in the universe that satisfies this property. Applied to our problem, the universe is a set of n-point metric spaces and the desired property of an n-point metric space M is that M cannot be embedded by an algorithm using $\mathcal{O}(n/\log n)$ bits of memory such that the embedding has distortion ϱ and slack σ. If a randomly chosen n-point metric space has this property with positive probability, then there must exist an n-point metric space that cannot be embedded by an algorithm using $\mathcal{O}(n/\log n)$ bits of memory such that the embedding has distortion ϱ and slack σ.

Now, let us consider any algorithm for embedding n-point metric spaces that uses at most k bits of memory. Then, this algorithm has at most 2^k distinct states. Each of these states can correspond to at most one target metric space. Let us denote the set of these target metric spaces by N. By using the probabilistic method, we show that, for a certain value of k, N is not a (σ, ϱ)-net, i.e., there exists an n-point metric space that cannot be embedded into any metric space in N with distortion ϱ and slack σ. This proves the assertion of the lemma for any algorithm using at most k bits of memory.

Let $M = (X, \mathrm{D})$ be a random n-point 1-2-metric space, i.e., every distance is chosen uniformly at random from $\{1, 2\}$. Without loss of generality, we assume that the computed embedding is non-expanding, i.e., the distance between any two points in X is at least as big as the corresponding distance of the embedded points in the target metric space. Let us consider an arbitrary target metric space $M' \in N$. There are $n!$ ways to embed M into M'. We fix one of these possible embeddings. Without loss of generality, we can assume that $X = X'$, i.e., our fixed embedding is the identity function. For the moment, let us assume that the embedding must have slack 0. Then, since our embedding is non-expanding, we must map all 1-distances in M to distances of length at most 1 in M'. Furthermore, since our embedding has distortion $\varrho < 2$, we must map all 2-distances in M to a value greater than 1 in M'. We call an assignment that violates these conditions a *wrong assignment*. Let $\mathrm{err}(M, M')$ be the total number of wrong assignments that occur by embedding M into M'. Since we allow a slack of σ, we are allowed to make at most $\sigma \cdot \binom{n}{2}$ wrong assignments. For two arbitrary points $x, y \in X$ in M', the distance $\mathrm{D}'(x, y)$ is either bigger than 1 or at most 1. Since $\mathrm{D}(x, y)$ is chosen uniformly at random from $\{1, 2\}$, the chance that the assignment of $\mathrm{D}(x, y)$ to $\mathrm{D}'(x, y)$ belongs to the wrong assignments is $1/2$. Therefore, by a Chernoff bound, we have

$$\begin{aligned}
\mathbf{Pr}\left[\mathrm{err}(M, M') \leq \sigma \cdot \binom{n}{2}\right] &< \mathbf{Pr}\left[\mathrm{err}(M, M') \leq \frac{1}{4} \cdot \binom{n}{2}\right] \\
&= \mathbf{Pr}\left[\mathrm{err}(M, M') \leq \left(1 - \frac{1}{2}\right) \cdot \mathbf{E}\left[\mathrm{err}(M, M')\right]\right] \\
&\leq \exp\left(-\frac{\mathbf{E}\left[\mathrm{err}(M, M')\right]}{8}\right) = \exp\left(-\frac{1}{16} \cdot \binom{n}{2}\right) \quad .
\end{aligned}$$

Since $|N| \leq 2^k$ and there are $n!$ ways to embed M into one metric space from N, the union bound implies that the overall probability that M embeds into any metric space from N with distortion ϱ and slack σ is at most

$$n! \cdot 2^k \cdot \exp\left(-\frac{1}{16} \cdot \binom{n}{2}\right) \ ,$$

which is less than 1 for certain $k = cn/\log n$ with sufficiently small constant c. Thus, the randomly chosen n-point metric space M cannot be embedded into any metric space from N with positive probability. It follows that, for such k, there must exist an n-point metric space that cannot be embedded into any metric space from N with distortion ϱ and slack σ, which completes the proof. □

Theorem 16. *Let ϱ, $1 \leq \varrho < 2$, be a precision parameter, and let σ, $0 \leq \sigma < 1/4$, be a slack parameter. Then, any algorithm that computes with positive probability for every metric space $M = (P, \mathrm{D})$, where P is a set of points from the discrete Euclidean space $\{0, \ldots, \Delta\} \subseteq \mathbb{R}$ and D is the Euclidean distance function defined on P, an (implicit or explicit) representation of another metric space $M' = (X', \mathrm{D}')$ such that there is an embedding from M to M' with distortion ϱ and slack σ requires $\Omega(\log(\log(\Delta)))$ bits of memory.*

Proof. For each $i \in \{1, \ldots, \lfloor \log \Delta \rfloor\}$, let M_i be the metric space obtained by placing $|P|/2$ points at the coordinate 0 and $|P|/2$ points at the coordinate 2^i. Any pair of these metric spaces differs by a factor of at least 2 in $|P|^2/4$ of its distances. Thus, there is no metric space M' such that both M_i and M_j, $i \neq j$, embed into M' with distortion less than 2 and slack less than $1/4$. Hence, for each of these metric spaces, there must exist a unique state of an algorithm that computes such an embedding. It follows that the algorithm has at least $\lfloor \log \Delta \rfloor$ states and so it needs $\Omega(\log(\log(\Delta)))$ bits of memory to distinguish them. □

9 Conclusions and Future Work

In this thesis, we developed facility location algorithms and embeddings with slack for huge datasets.

Chapter 3

We presented a randomized distributed algorithm for the facility location problem, considering both metric spaces and powers of metric spaces. For the special case of uniform costs and demands, our algorithm provides a constant-factor approximation using three communication rounds. We believe that our algorithm is particularly well-suited for facility location types of problems in wireless networks. This is because the algorithm uses only a few broadcasts (in every communication round each node sends the same message to its neighbors), which can be easily done in wireless networks.

In the analysis, we used the fact that the sum of the radii of the points is a constant-factor approximation of the expected total cost. This fact is not directly applicable to the non-uniform case, which means that our result cannot directly be generalized to the non-uniform metric facility location problem. However, motivated by our result, Pandit and Pemmaraju [97] obtained a constant-factor approximation in $\mathcal{O}(\log(n))$ communication rounds for the variant of our considered metric facility location problem where the opening cost of facilities are non-uniform. It would be interesting to find out whether $\Omega(\log(n))$ communication rounds are required or not.

Chapter 4

We initiated the study on a KDS for the mobile facility location problem. In particular, we proposed a KDS that maintains a subset of the moving input points as open facilities such that, at any time, the associated total cost is at most a constant factor larger than the current optimal cost. We showed that our KDS is compact, local, responsive, and efficient.

Note that the complexity of our KDS is polylogarithmic in R, which is a value depending on the opening cost and demand values of the input points. Hence, the compactness, locality, responsiveness, and efficiency are not fully polylogarithmic, but only pseudo-polylogarithmic. It would be nice future work to reduce this pseudo-polylogarithmic term to a real polylogarithmic term. Furthermore, future work in the area of mobile facility location problems could include to consider additional opening cost that arises at the point of time when a point changes its status from client to open facility. Here, we point out that in our scenario the additional opening cost per event would be already bounded because we open at most a logarithmic number of facilities per event.

Chapter 5

Chapter 5 addresses one of the central results in this thesis. In this chapter, we developed a randomized algorithm that computes a constant-factor approximation of the cost for the uniform facility location problem over dynamic geometric data streams. Our streaming algorithm strongly improves the best previous one, which guarantees an approximation factor of $\mathcal{O}(\log^2(\Delta))$.

We think that it is worthwhile to further investigate the uniform facility location problem over dynamic geometric data streams. In particular, we are optimistic that one can obtain a $(1 \pm \varepsilon)$-approximation algorithm for the facility location cost. This might also provide new insights into handling other problems in the dynamic geometric data stream model, like computing the earth mover distance or the minimum length of a traveling-salesperson tour, for instance. Obviously, future work could also include to generalize our results to the non-uniform facility location variant.

Chapter 6

We presented a streaming implementation of a k-means clustering algorithm that is based on a new coreset construction. We have shown that this algorithm is capable of efficiently clustering huge amounts of data in the insertion-only data stream model. To evaluate our algorithm, we ran a series of experiments on large real-world datasets. We found empirical evidence that in terms of the cost of the clustering, our algorithm is comparable with StreamLS and significantly better than BIRCH. In terms of the running time, our algorithm outperforms StreamLS, especially for a large number of centers k.

From a theoretical point of view, we showed that, for a precision parameter ε with $0 < \varepsilon < 1$, our adaptive sampling approach computes a (k, ε)-coreset in constant-dimensional Euclidean space. However, the bound on the coreset size depends exponentially on the dimension d (see Theorem 6). In compliance with our experiments, we suggest that one can prove a size bound with much lower dependency on the dimension. Also, from an experimental point of view, it would be interesting to examine the effect of the dimension on an appropriate coreset size more extensively.

Chapters 7 and 8

We considered compact representations of metric spaces. In Chapter 7, we introduced the notion of a WSPD with slack and gave constructions of WSPDs with slack for Euclidean and doubling metric spaces. In Chapter 8, we presented streaming algorithms to compute low-distortion embeddings with low slack for Euclidean, doubling, and general metric spaces. Furthermore, we used an embedding to obtain a randomized algorithm that computes a $(1 \pm \varepsilon)$-approximation of the max-cut problem for a dynamic geometric data stream of high-dimensional Euclidean points.

Metric embeddings with slack preserve much information about the original pairwise distances and can be stored in small space. For this reason, we believe that they are an important tool in the analysis of data streams and deserve further investigation.

A Additional Tables for Chapter 6

A.1 Parameters of Algorithm BIRCH

	Covertype	*Tower*	*Census 1990*	*BigCross*
$p =$	10	5	5	25

Table A.1: Manual adjustment of the parameter TotalMemSize as percentage of the dataset size for algorithm BIRCH

parameter	value
CorD	0
TotalMemSize (in bytes)	p% of dataset size
TotalBufferSize (in bytes)	5% of TotalMemSize
TotalQueueSize (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

Table A.2: Setting of the parameters of algorithm BIRCH

A.2 Running Times of the Algorithms

		running time (in sec)				
dataset	k	StreamKM++	StreamLS	BIRCH	k-Means++	k-Means
Spambase	10	3.06	-	-	3.57	19.02
	20	7.04	-	-	8.22	59.85
	30	16.45	-	-	19.05	88.8
	40	28.93	-	-	20.54	132.03
	50	44.48	-	-	25.9	182.08
Intrusion	10	74.1	-	-	50.6	408.8
	20	103.1	-	-	262.4	2711.3
	30	143.8	-	-	1973.3	4389.1
	40	197.6	-	-	1257.0	10733.7
	50	250.5	-	-	1339.5	14282.0
Covertype	10	245	147	44	3389	-
	20	297	460	44	5160	-
	30	378	1027	44	14933	-
	40	454	1773	44	16713	-
	50	617	2588	44	25803	-
Tower	20	157	679	77	2960	-
	40	168	1989	78	6902	-
	60	187	3849	77	11247	-
	80	211	6212	77	19206	-
	100	248	8946	77	17161	-
Census 1990	10	1571	631	271	-	-
	20	1724	2362	271	-	-
	30	1839	5504	271	-	-
	40	1956	10054	272	-	-
	50	2057	11842	272	-	-
BigCross	15	5486	6239	1006	-	-
	20	5738	10502	998	-	-
	25	5933	15780	996	-	-
	30	6076	22779	996	-	-
Normdata	100	14.5	178.2	-	-	-
($m = 500$)	125	14.9	401.8	-	-	-
	150	15.1	569.3	-	-	-
	175	15.1	659.3	-	-	-
	200	15.6	731.8	-	-	-
Normdata	100	16.7	44.8	-	-	-
($m = 1000$)	125	17.1	92.6	-	-	-
	150	17.5	176.9	-	-	-
	175	17.6	378.1	-	-	-
	200	18.3	586.7	-	-	-

Table A.3: Average running times of the algorithms

A.3 Clustering Cost of the Algorithms

		cost				
dataset	k	StreamKM++	StreamLS	BIRCH	k-Means++	k-Means
Spambase	10	$7.85 \cdot 10^7$	-	-	$8.71 \cdot 10^7$	$1.70 \cdot 10^8$
	20	$2.27 \cdot 10^7$	-	-	$2.45 \cdot 10^7$	$1.53 \cdot 10^8$
	30	$1.24 \cdot 10^7$	-	-	$1.34 \cdot 10^7$	$1.51 \cdot 10^8$
	40	$8.64 \cdot 10^6$	-	-	$9.01 \cdot 10^6$	$1.49 \cdot 10^8$
	50	$6.29 \cdot 10^6$	-	-	$6.68 \cdot 10^6$	$1.48 \cdot 10^8$
Intrusion	10	$1.27 \cdot 10^{13}$	-	-	$1.75 \cdot 10^{13}$	$9.52 \cdot 10^{14}$
	20	$1.26 \cdot 10^{12}$	-	-	$1.55 \cdot 10^{12}$	$9.51 \cdot 10^{14}$
	30	$4.29 \cdot 10^{11}$	-	-	$4.96 \cdot 10^{11}$	$9.51 \cdot 10^{14}$
	40	$1.95 \cdot 10^{11}$	-	-	$2.25 \cdot 10^{11}$	$9.50 \cdot 10^{14}$
	50	$1.11 \cdot 10^{11}$	-	-	$1.29 \cdot 10^{11}$	$9.50 \cdot 10^{14}$
Covertype	10	$3.43 \cdot 10^{11}$	$3.42 \cdot 10^{11}$	$4.24 \cdot 10^{11}$	$3.42 \cdot 10^{11}$	-
	20	$2.06 \cdot 10^{11}$	$2.05 \cdot 10^{11}$	$2.97 \cdot 10^{11}$	$2.03 \cdot 10^{11}$	-
	30	$1.57 \cdot 10^{11}$	$1.56 \cdot 10^{11}$	$1.89 \cdot 10^{11}$	$1.54 \cdot 10^{11}$	-
	40	$1.31 \cdot 10^{11}$	$1.32 \cdot 10^{11}$	$1.59 \cdot 10^{11}$	$1.29 \cdot 10^{11}$	-
	50	$1.15 \cdot 10^{11}$	$1.18 \cdot 10^{11}$	$1.41 \cdot 10^{11}$	$1.13 \cdot 10^{11}$	-
Tower	20	$6.24 \cdot 10^8$	$6.16 \cdot 10^8$	$9.26 \cdot 10^8$	$6.51 \cdot 10^8$	-
	40	$3.34 \cdot 10^8$	$3.34 \cdot 10^8$	$4.75 \cdot 10^8$	$3.30 \cdot 10^8$	-
	60	$2.43 \cdot 10^8$	$2.37 \cdot 10^8$	$3.89 \cdot 10^8$	$2.40 \cdot 10^8$	-
	80	$1.95 \cdot 10^8$	$1.91 \cdot 10^8$	$3.47 \cdot 10^8$	$1.92 \cdot 10^8$	-
	100	$1.65 \cdot 10^8$	$1.63 \cdot 10^8$	$2.98 \cdot 10^8$	$1.63 \cdot 10^8$	-
Census 1990	10	$2.48 \cdot 10^8$	$2.40 \cdot 10^8$	$3.98 \cdot 10^8$	-	-
	20	$1.90 \cdot 10^8$	$1.85 \cdot 10^8$	$3.17 \cdot 10^8$	-	-
	30	$1.59 \cdot 10^8$	$1.53 \cdot 10^8$	$2.94 \cdot 10^8$	-	-
	40	$1.41 \cdot 10^8$	$1.35 \cdot 10^8$	$2.78 \cdot 10^8$	-	-
	50	$1.28 \cdot 10^8$	$1.24 \cdot 10^8$	$2.73 \cdot 10^8$	-	-
BigCross	15	$5.05 \cdot 10^{12}$	$5.23 \cdot 10^{12}$	$6.69 \cdot 10^{12}$	-	-
	20	$4.15 \cdot 10^{12}$	$4.23 \cdot 10^{12}$	$4.85 \cdot 10^{12}$	-	-
	25	$3.59 \cdot 10^{12}$	$3.54 \cdot 10^{12}$	$4.45 \cdot 10^{12}$	-	-
	30	$3.18 \cdot 10^{12}$	$3.18 \cdot 10^{12}$	$3.83 \cdot 10^{12}$	-	-
Normdata	100	$1.50 \cdot 10^6$	$1.50 \cdot 10^6$	-	-	-
($m = 500$)	125	$1.50 \cdot 10^6$	$1.50 \cdot 10^6$	-	-	-
	150	$1.50 \cdot 10^6$	$1.50 \cdot 10^6$	-	-	-
	175	$1.50 \cdot 10^6$	$1.50 \cdot 10^6$	-	-	-
	200	$1.50 \cdot 10^6$	$1.50 \cdot 10^6$	-	-	-
Normdata	100	$1.50 \cdot 10^6$	$1.50 \cdot 10^6$	-	-	-
($m = 1000$)	125	$1.50 \cdot 10^6$	$1.50 \cdot 10^6$	-	-	-
	150	$1.50 \cdot 10^6$	$1.50 \cdot 10^6$	-	-	-
	175	$1.50 \cdot 10^6$	$1.50 \cdot 10^6$	-	-	-
	200	$1.50 \cdot 10^6$	$1.50 \cdot 10^6$	-	-	-

Table A.4: Average clustering cost of the algorithms

A.4 Standard Deviation of Running Time and Cost

		running time (in sec)			
dataset	k	StreamKM++	StreamLS	k-Means++	k-Means
Spambase	10	0.29	-	1.5	3.33
	20	1.09	-	3.88	6.36
	30	1.52	-	11.27	17.61
	40	6.56	-	6.97	26.95
	50	6.59	-	12.83	68.1
Intrusion	10	0.68	-	40.81	58.84
	20	3.22	-	98.11	499.7
	30	6.07	-	1263.44	345.6
	40	24.91	-	563.20	1306.2
	50	31.58	-	706.00	1190.78
Covertype	10	0.88	2.43	2295.85	-
	20	6.93	18.18	1249.18	-
	30	4.15	2.14	9653.06	-
	40	14.02	7.64	6838.93	-
	50	39.28	123.28	12231.98	-
Tower	20	0.58	14.11	1594.76	-
	40	1.79	50.83	2085.12	-
	60	3.96	58.27	3656.87	-
	80	7.95	122.65	5162.60	-
	100	11.34	315.31	1795.07	-
Census 1990	10	2.04	9.08	-	-
	20	5.16	54.3	-	-
	30	5.38	98.03	-	-
	40	23.31	193.00	-	-
	50	17.43	533.39	-	-
BigCross	15	10.49	93.6	-	-
	20	11.49	162.44	-	-
	25	15.69	226.38	-	-
	30	16.66	200.68	-	-
Normdata	100	0.07	1.22	-	-
($m = 500$)	125	0.05	1.14	-	-
	150	0.05	2.19	-	-
	175	0.03	2.89	-	-
	200	0.03	4.05	-	-
Normdata	100	0.06	0.6	-	-
($m = 1000$)	125	0.06	1.32	-	-
	150	0.04	2.56	-	-
	175	0.08	3.96	-	-
	200	0.2	2.41	-	-

Table A.5: Standard deviation of the running time

		cost			
dataset	k	StreamKM++	StreamLS	k-Means++	k-Means
Spambase	10	$2.05 \cdot 10^6$	-	$9.57 \cdot 10^6$	$1.06 \cdot 10^6$
	20	$6.49 \cdot 10^5$	-	$1.73 \cdot 10^6$	$8.78 \cdot 10^4$
	30	$3.14 \cdot 10^5$	-	$9.51 \cdot 10^5$	$8.81 \cdot 10^4$
	40	$1.93 \cdot 10^5$	-	$5.31 \cdot 10^5$	$3.42 \cdot 10^6$
	50	$1.49 \cdot 10^5$	-	$2.47 \cdot 10^5$	$2.91 \cdot 10^6$
Intrusion	10	$1.39 \cdot 10^{12}$	-	$6.61 \cdot 10^{12}$	$3.09 \cdot 10^{11}$
	20	$8.54 \cdot 10^{10}$	-	$3.70 \cdot 10^{11}$	$8.20 \cdot 10^9$
	30	$3.13 \cdot 10^{10}$	-	$6.85 \cdot 10^{10}$	$2.54 \cdot 10^{10}$
	40	$7.03 \cdot 10^9$	-	$3.25 \cdot 10^{10}$	$1.53 \cdot 10^8$
	50	$6.01 \cdot 10^9$	-	$1.61 \cdot 10^{10}$	$6.82 \cdot 10^8$
Covertype	10	$2.47 \cdot 10^9$	$2.70 \cdot 10^{10}$	$3.63 \cdot 10^9$	-
	20	$1.08 \cdot 10^9$	$1.03 \cdot 10^{10}$	$9.17 \cdot 10^8$	-
	30	$1.49 \cdot 10^9$	$6.61 \cdot 10^9$	$6.12 \cdot 10^8$	-
	40	$8.38 \cdot 10^8$	$5.63 \cdot 10^9$	$6.64 \cdot 10^8$	-
	50	$5.68 \cdot 10^8$	$3.90 \cdot 10^9$	$2.92 \cdot 10^8$	-
Tower	20	$7.31 \cdot 10^6$	$2.71 \cdot 10^7$	$4.39 \cdot 10^7$	-
	40	$1.85 \cdot 10^6$	$1.65 \cdot 10^7$	$4.37 \cdot 10^6$	-
	60	$1.52 \cdot 10^6$	$1.55 \cdot 10^7$	$1.61 \cdot 10^6$	-
	80	$1.03 \cdot 10^6$	$9.63 \cdot 10^6$	$1.54 \cdot 10^6$	-
	100	$7.73 \cdot 10^5$	$1.03 \cdot 10^7$	$1.17 \cdot 10^6$	-
Census 1990	10	$5.02 \cdot 10^6$	$1.45 \cdot 10^5$	-	-
	20	$3.66 \cdot 10^6$	$3.14 \cdot 10^6$	-	-
	30	$1.61 \cdot 10^6$	$9.34 \cdot 10^5$	-	-
	40	$1.21 \cdot 10^6$	$8.13 \cdot 10^5$	-	-
	50	$1.01 \cdot 10^6$	$6.80 \cdot 10^5$	-	-
BigCross	15	$3.22 \cdot 10^{10}$	$1.75 \cdot 10^{11}$	-	-
	20	$2.46 \cdot 10^{10}$	$3.36 \cdot 10^{11}$	-	-
	25	$1.86 \cdot 10^{10}$	$1.76 \cdot 10^{11}$	-	-
	30	$1.94 \cdot 10^{10}$	$1.29 \cdot 10^{11}$	-	-
Normdata	100	0	0	-	-
($m = 500$)	125	0	0	-	-
	150	0	0	-	-
	175	0	0	-	-
	200	0	0	-	-
Normdata	100	0	0	-	-
($m = 1000$)	125	0	0	-	-
	150	0	0	-	-
	175	0	0	-	-
	200	0	0	-	-

Table A.6: Standard deviation of the clustering cost

B Mathematical Fundamentals

This appendix deals with some mathematical fundamentals which are assumed to be common knowledge throughout this thesis. In Section B.1, we specify partial sums of some classical series and state some useful inequalities concerning Euler's number. Section B.2 addresses probability theory.

B.1 Sequences, Series, and Inequalities

Arithmetic Series

Let $(a_1, a_2, \ldots)$ be any infinite *arithmetic sequence*, i.e., there is a fixed constant $d \in \mathbb{R}$, called the *common difference*, such that $a_i - a_{i-1} = d$ for all $i \in \mathbb{N}\backslash\{1\}$. It is well-known and can be proven by simple induction that the n-th partial sum of the associated infinite series is equal to

$$\sum_{i=1}^{n} a_i = \frac{n}{2} \cdot (a_1 + a_n) \quad .$$

In particular, the sum of the first n natural numbers is equal to

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \quad .$$

Geometric Series

Let $(a_1, a_2, \ldots)$ be any infinite *geometric sequence*, i.e., there is a fixed constant $q \in \mathbb{R}$ with $q \neq 1$, called the *common ratio*, such that $a_i / a_{i-1} = q$ for all $i \in \mathbb{N}\backslash\{1\}$. It is well-known and can be proven by simple induction that the n-th partial sum of the associated infinite series is equal to

$$\sum_{i=1}^{n} a_i = a_1 \cdot \sum_{i=0}^{n-1} q^i = a_1 \cdot \frac{q^n - 1}{q - 1} \quad .$$

In case that $|q| < 1$, the sum of the infinite geometric series is

$$\sum_{i=1}^{\infty} a_i = a_1 \cdot \frac{1}{1 - q} \quad .$$

Bounds on Euler's Number

In the following, we will state some useful inequalities concerning Euler's number (see [91, 99]). For all $n \in \mathbb{N}$, we have

$$\left(1+\frac{1}{n}\right)^n \leq e \leq \left(1+\frac{1}{n}\right)^{n+1} \tag{B.1}$$

and

$$\left(1-\frac{1}{n}\right)^n \leq \frac{1}{e} \leq \left(1-\frac{1}{n}\right)^{n-1} \quad . \tag{B.2}$$

These inequalities imply

$$\lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n = e \quad \text{and} \quad \lim_{n\to\infty}\left(1-\frac{1}{n}\right)^n = \frac{1}{e} \ .$$

Furthermore, it is known that, for all $n \in \mathbb{N}$, we have

$$\left(\frac{n}{e}\right)^n \leq n! \leq n^n \ .$$

B.2 Probability Theory

This section addresses some basics in probability theory which are assumed to be common knowledge throughout this thesis. The interested reader can find a more general introduction in [89, 99].

The set Ω of all possible outcomes of a random experiment is called a *sample space*. In this thesis, we only consider discrete sample spaces, i.e., any considered sample space is a countable set of elementary events of the form $\Omega = \{\omega_1, \omega_2, \ldots, \omega_n\}$. The probability distribution on Ω is a function $p : \Omega \to \mathbb{R}$ which satisfies the following two conditions:

(*i*) the probability associated with any elementary event is non-negative, i.e.,

$$p(\omega_i) \geq 0 \text{ , for any } \omega_i \in \Omega \ ,$$

(*ii*) the sum of probabilities over all elementary events is equal to 1, i.e.,

$$\sum_{\omega_i \in \Omega} p(\omega_i) = 1 \ .$$

A subset of Ω is called an *event*. Any event $E \subseteq \Omega$ is said to be true if the outcome of the random experiment is any $\omega \in E$. Otherwise, E is said to be false. The probability of E is defined by

$$\mathbf{Pr}\left[E\right] := \sum_{\omega \in E} p(\omega) \ .$$

The probability of an event E_2 assuming an event E_1 with $\mathbf{Pr}\left[E_1\right] > 0$ is called *conditional probability* and is defined as

$$\mathbf{Pr}\left[E_2 \mid E_1\right] := \frac{\mathbf{Pr}\left[E_1 \cap E_2\right]}{\mathbf{Pr}\left[E_1\right]} \ .$$

Random Variables, Expectation, and Variance

A *random variable* is a function $X := X(\omega)$ defined on a sample space Ω. In this thesis, we only consider *discrete* and *real-valued* random variables. Such a random variable is a function $X : \Omega \to \mathbb{R}$ which only takes isolated values with non-zero probabilities. A random variable X is called an *indicator random variable* for an event E in case that, for all $\omega \in \Omega$, we have $X(\omega) \in \{0, 1\}$ and $X(\omega) = 1$ if and only if $\omega \in E$.

For any discrete and real-valued random variable X and any real $k \in \mathbb{R}$, we define $[X = k] := \{\omega \in \Omega \mid X(\omega) = k\}$. Based on this definition, we use the abbreviations

$$\mathbf{Pr}\left[X \leq k\right] := \sum_{\ell \leq k : \ell \in X(\Omega)} \mathbf{Pr}\left[X = \ell\right] \quad \text{and} \quad \mathbf{Pr}\left[X \geq k\right] := \sum_{\ell \geq k : \ell \in X(\Omega)} \mathbf{Pr}\left[X = \ell\right] \ .$$

Furthermore, for two random variables X and Y, we use the abbreviations

$$\mathbf{Pr}\left[[X = k] \cap [Y = \ell]\right] := \mathbf{Pr}\left[X = k \wedge Y = \ell\right]$$

and

$$\mathbf{Pr}\left[[X = k] \cup [Y = \ell]\right] := \mathbf{Pr}\left[X = k \vee Y = \ell\right] \ .$$

Two random variables X and Y are called *independent* if, for all $x, y \in \mathbb{R}$,

$$\mathbf{Pr}\left[X = x \mid Y = y\right] = \mathbf{Pr}\left[X = x\right] \ .$$

A set $X_0, X_1, \ldots, X_{n-1}$ of random variables is called *independent* if, for all $i \in [n]$ and $I \subseteq [n] \backslash \{i\}$,

$$\mathbf{Pr}\left[X_i = x_i \mid \bigwedge_{j \in I} X_j = x_j\right] = \mathbf{Pr}\left[X_i = x_i\right] \tag{B.3}$$

for all $x_i \in \mathbb{R}$ and $x_j \in \mathbb{R}$ with $j \in I$. A set $X_0, X_1, \ldots, X_{n-1}$ of random variables is called *k-wise independent* if (B.3) holds for all $I \subseteq [n] \backslash \{i\}$ with $|I| \leq k$.

The *expectation* of any discrete and real-valued random variable X is defined as

$$\mathbf{E}\left[X\right] := \sum_{k \in X(\Omega)} k \cdot \mathbf{Pr}\left[X = k\right] \ .$$

The following three properties of the expectation are often used in the analysis of randomized algorithms. Proofs can be found in [89].

(*i*) For any random variable X and any real $k \in \mathbb{R}$, we have $\mathbf{E}\left[k \cdot X\right] = k \cdot \mathbf{E}\left[X\right]$.

(*ii*) For any two random variables X and Y, we have $\mathbf{E}\left[X + Y\right] = \mathbf{E}\left[X\right] + \mathbf{E}\left[Y\right]$. This property is called *linearity of expectation*.

(*iii*) For any two independent random variables X and Y, $\mathbf{E}\left[X \cdot Y\right] = \mathbf{E}\left[X\right] \cdot \mathbf{E}\left[Y\right]$.

The *variance* of any random variable X is defined as

$$\mathbf{V}[X] := \mathbf{E}\left[(X - \mathbf{E}[X])^2\right] \quad .$$

By expanding the term $(X - \mathbf{E}[X])^2$, we get

$$\begin{aligned} \mathbf{V}[X] &= \mathbf{E}\left[(X - \mathbf{E}[X])^2\right] \\ &= \mathbf{E}\left[X^2 - 2X \cdot \mathbf{E}[X] + \mathbf{E}[X]^2\right] \\ &= \mathbf{E}\left[X^2\right] - 2 \cdot \mathbf{E}[X] \cdot \mathbf{E}[X] + \mathbf{E}[X]^2 \\ &= \mathbf{E}\left[X^2\right] - \mathbf{E}[X]^2 \quad . \end{aligned}$$

The following three properties of the variance are often used in the analysis of randomized algorithms. Proofs can be found in [89].

(*i*) For any random variable X, we have $\mathbf{V}[X] \geq 0$.

(*ii*) For any random variable X and any two reals $a, b \in \mathbb{R}$, $\mathbf{V}[a + b \cdot X] = b^2 \cdot \mathbf{V}[X]$.

(*iii*) For any two independent random variables X and Y, $\mathbf{V}[X + Y] = \mathbf{V}[X] + \mathbf{V}[Y]$.

The *standard deviation* of a random variable X is defined as $\sigma := \sqrt{\mathbf{V}[X]}$.

Useful Inequalities

The following inequalities are frequently used in the analysis of our randomized algorithms.

Union Bound

Let E_1 and E_2 be any two events. Then, we have

$$\mathbf{Pr}[E_1 \vee E_2] \leq \mathbf{Pr}[E_1] + \mathbf{Pr}[E_2] \quad .$$

The union bound is implied by the inclusion-exclusion principle from combinatorics.

Markov's Inequality

Let $X : \Omega \to \mathbb{R}_{\geq 0}$ be a non-negative random variable. Then, for any $k \in \mathbb{R}$ with $k > 0$, we have

$$\mathbf{Pr}[X \geq k] \leq \frac{\mathbf{E}[X]}{k} \quad .$$

A proof of Markov's inequality can be found in [89].

Chebyshev's Inequality

Let $X : \Omega \to \mathbb{R}$ be a random variable. Then, for any $k \in \mathbb{R}$ with $k > 0$, we have

$$\mathbf{Pr}\left[|X - \mathbf{E}\left[X\right]| \geq k\right] \leq \frac{\mathbf{V}\left[X\right]}{k^2} \quad .$$

Chebyshev's inequality follows from Markov's inequality. A formal proof can be found in [89].

Chernoff Bounds

Let $X_1, X_2, \ldots, X_n : \Omega \to \{0, 1\}$ be a set of independent 0-1-random variables. Then, it holds for all $\varepsilon \geq 0$ that

$$\mathbf{Pr}\left[\sum_{i=1}^{n} X_i \geq (1+\varepsilon) \cdot \mathbf{E}\left[\sum_{i=1}^{n} X_i\right]\right] \leq \exp\left(-\frac{1}{3} \cdot \min\{\varepsilon, \varepsilon^2\} \cdot \mathbf{E}\left[\sum_{i=1}^{n} X_i\right]\right) \quad .$$

Furthermore, it holds for all ε, $0 \leq \varepsilon \leq 1$, that

$$\mathbf{Pr}\left[\sum_{i=1}^{n} X_i \leq (1-\varepsilon) \cdot \mathbf{E}\left[\sum_{i=1}^{n} X_i\right]\right] \leq \exp\left(-\frac{1}{2} \cdot \varepsilon^2 \cdot \mathbf{E}\left[\sum_{i=1}^{n} X_i\right]\right) \quad .$$

A formal proof can be found in [86].

Bibliography

[1] I. Abraham, Y. Bartal, and O. Neiman. Advances in metric embedding theory. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC '06)*, pages 271–286. Association for Computing Machinery, 2006.

[2] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Approximating extent measures of points. *Journal of the ACM (JACM)*, 51(4):606–635, July 2004.

[3] A. Aggarwal, A. Deshpande, and R. Kannan. Adaptive sampling for k-means clustering. In *Proceedings of the 12th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX '10)*, pages 15–28. Springer, 2009.

[4] N. Ailon, R. Jaiswal, and C. Monteleoni. Streaming k-means approximation. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 10–18. MIT press, 2009.

[5] N. Alon, L. Babai, and A. Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of Algorithms*, 7(4):567–583, 1986.

[6] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences (JCSS)*, 58(1):137–147, February 1999.

[7] S. Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM (JACM)*, 45(5):753–782, September 1998.

[8] S. Arora, P. Raghavan, and S. Rao. Approximation schemes for Euclidean k-medians and related problems. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC '98)*, pages 106–113. Association for Computing Machinery, 1998.

[9] D. Arthur and S. Vassilvitskii. `k-means++`: The advantages of careful seeding. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '07)*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[10] S. Arya, G. Das, D. M. Mount, J. S. Salowe, and M. H. M. Smid. Euclidean spanners: Short, thin, and lanky. In *Proceedings of the 27th Annual ACM Symposium on Theory*

of Computing (STOC '95), pages 489–498. Association for Computing Machinery, 1995.

[11] S. Arya, D. M. Mount, and M. H. M. Smid. Randomized and deterministic algorithms for geometric spanners of small diameter. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS '94)*, pages 703–712. IEEE Computer Society, 1994.

[12] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local search heuristics for k-median and facility location problems. *SIAM Journal on Computing*, 33(3):544–562, 2004.

[13] A. Asuncion and D. J. Newman. UCI machine learning repository, 2007. University of California, Irvine, School of Information and Computer Sciences, available at `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

[14] M. Bădoiu, A. Czumaj, P. Indyk, and C. Sohler. Facility location in sublinear time. In *Proceedings of the 32nd Annual International Colloquium on Automata, Languages and Programming (ICALP '05)*, volume 3580, pages 866–877. Springer, 2005.

[15] J. Basch, L. J. Guibas, and J. Hershberger. Data structures for mobile data. *Journal of Algorithms*, 31(1):1–28, 1999.

[16] J. L. Bentley and J. B. Saxe. Decomposable searching problems I. Static-to-dynamic transformation. *Journal of Algorithms*, 1(4):301–358, December 1980.

[17] S. Bereg, B. K. Bhattacharya, D. G. Kirkpatrick, and M. Segal. Competitive algorithms for maintaining a mobile center. *MONET*, 11(2):177–186, April 2006.

[18] J. Byrka and K. Aardal. An optimal bifactor approximation algorithm for the metric uncapacitated facility location problem. *SIAM Journal on Computing*, 39(6):2212–2231, March 2010.

[19] P. B. Callahan. Optimal parallel all-nearest-neighbors using the well-separated pair decomposition (preliminary version). In *Proceedings of the 34th Annual Symposium on Foundations of Computer Science (FOCS '93)*, pages 332–340. IEEE Computer Society, 1993.

[20] P. B. Callahan. *Dealing with higher dimensions: The well-separated pair decomposition and its applications.* PhD thesis, Johns Hopkins University, Baltimore, Maryland, 1995.

[21] P. B. Callahan and S. R. Kosaraju. Algorithms for dynamic closest pair and n-body potential fields. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '95)*, pages 263–272. Society for Industrial and Applied Mathematics, 1995.

[22] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Journal of the ACM (JACM)*, 42(1):67–90, 1995.

[23] T.-H. H. Chan, K. Dhamdhere, A. Gupta, J. Kleinberg, and A. Slivkins. Metric embeddings with relaxed guarantees. *SIAM Journal on Computing*, 38(6):2303–2329, March 2009.

[24] T.-H. H. Chan, M. Dinitz, and A. Gupta. Spanners with slack. In *Proceedings of the 14th Annual European Symposium on Algorithms (ESA '06)*, pages 196–207. Springer, 2006.

[25] T. M. Chan. Well-separated pair decomposition in linear time? *Information Processing Letters*, 107(5):138–141, August 2008.

[26] K. L. Chang. Pass-efficient algorithms for facility location. Technical Report YALEU/DCS/TR-1337, Yale University, November 2005.

[27] M. Charikar and S. Guha. Improved combinatorial algorithms for facility location problems. *SIAM Journal on Computing*, 34(4):803–824, 2005.

[28] K. Chen. On k-median clustering in high dimensions. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '06)*, pages 1177–1185. Association for Computing Machinery, 2006.

[29] K. Chen. On coresets for k-median and k-means clustering in metric and Euclidean spaces and their applications. *SIAM Journal on Computing*, 39(3):923–947, August 2009.

[30] F. A. Chudak and D. B. Shmoys. Improved approximation algorithms for the uncapacitated facility location problem. *SIAM Journal on Computing*, 33(1):1–25, 2003.

[31] A. Czumaj, G. Frahling, and C. Sohler. Efficient kinetic data structures for max-cut. In *Proceedings of the 19th Canadian Conference on Computational Geometry (CCCG '07)*, pages 157–160. Carleton University, Ottawa, Canada, 2007.

[32] A. Czumaj and C. Sohler. Small space representations for metric min-sum k-clustering and their applications. *Theory of Computing Systems*, 46(3):416–442, April 2010.

[33] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 3rd edition, 2008.

[34] J. Erickson. Dense point sets have sparse Delaunay triangulations or "... but not too nasty". *Discrete & Computational Geometry*, 33(1):83–115, January 2005.

[35] J. Fakcharoenphol, S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *Journal of Computer and System Sciences (JCSS)*, 69(3):485–497, November 2004.

[36] D. Feldman, A. Fiat, and M. Sharir. Coresets for weighted facilities and their applications. In *Proceedings of the 47th IEEE Symposium on Foundations of Computer Science (FOCS '06)*, pages 315–324. IEEE Computer Society, 2006.

[37] D. Feldman, M. Monemizadeh, and C. Sohler. A PTAS for k-means clustering based on weak coresets. In *Proceedings of the 23rd ACM Symposium on Computational Geometry (SCG '07)*, pages 11–18. Association for Computing Machinery, 2007.

[38] J. Fischer and S. Har-Peled. Dynamic well-separated pair decomposition made easy. In *Proceedings of the 17th Canadian Conference on Computational Geometry (CCCG '05)*, pages 235–238. University of Windsor, Ontario, Canada, 2005.

[39] E. W. Forgy. Cluster analysis of multivariate data: Efficiency versus interpretability of classifications. *Biometrics*, 21:768–780, 1965.

[40] D. Fotakis. Incremental algorithms for facility location and k-median. *Theoretical Computer Science*, 361(2–3):275–313, September 2006.

[41] D. Fotakis. Memoryless facility location in one pass. In *Proceedings of the 23rd Annual Symposium on Theoretical Aspects of Computer Science (STACS '06)*, pages 608–620. Springer, 2006.

[42] G. Frahling. *Algorithms for Dynamic Geometric Data Streams*. PhD thesis, University of Paderborn, 2006.

[43] G. Frahling, P. Indyk, and C. Sohler. Sampling in dynamic data streams and applications. *International Journal of Computational Geometry and Applications (IJCGA)*, 18(1–2):3–28, 2008.

[44] G. Frahling and C. Sohler. Coresets in dynamic geometric data streams. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC '05)*, pages 209–217. Association for Computing Machinery, 2005.

[45] S. A. Friedler and D. M. Mount. Approximation algorithm for the kinetic robust k-center problem. *Computational Geometry*, 43(6–7):572–586, August 2010.

[46] J. Gao, L. J. Guibas, J. Hershberger, L. Zhang, and A. Zhu. Discrete mobile centers. *Discrete & Computational Geometry*, 30(1):45–63, July 2003.

[47] J. Gao, L. J. Guibas, and A. T. Nguyen. Deformable spanners and applications. *Computational Geometry*, 35(1–2):2–19, August 2006.

[48] J. Gao and L. Zhang. Well-separated pair decomposition for the unit-disk graph metric and its applications. *SIAM Journal on Computing*, 35(1):151–169, 2005.

[49] B. Gfeller and E. Vicari. A randomized distributed algorithm for the maximal independent set problem in growth-bounded graphs. In *Proceedings of the 26th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC '07)*, pages 53–60. Association for Computing Machinery, 2007.

[50] J. Gudmundsson, C. Levcopoulos, G. Narasimhan, and M. H. M. Smid. Approximate distance oracles for geometric graphs. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '02)*, pages 828–837. Society for Industrial and Applied Mathematics, 2002.

[51] S. Guha and S. Khuller. Greedy strikes back: Improved facility location algorithms. *Journal of Algorithms*, 31(1):228–248, April 1999.

[52] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams: Theory and practice. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 15(3):515–528, January/February 2003.

[53] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *Proceedings of the 41st Symposium on Foundations of Computer Science (FOCS '00)*, pages 359–366. IEEE Computer Society, 2000.

[54] L. J. Guibas. Kinetic data structures: A state of the art report. In *Proceedings of the 3rd Workshop on the Algorithmic Foundations of Robotics (WAFR '98)*, pages 191–209. A. K. Peters, Ltd., 1998.

[55] L. J. Guibas. Kinetic data structures. In D. P. Mehta and S. Sahni, editors, *Handbook of Data Structures and Applications*. Chapman and Hall/CRC, 2004.

[56] S. Har-Peled. Clustering motion. *Discrete & Computational Geometry*, 31(4):545–565, March 2004.

[57] S. Har-Peled and A. Kushal. Smaller coresets for k-median and k-means clustering. *Discrete & Computational Geometry*, 37(1):3–19, January 2007.

[58] S. Har-Peled and S. Mazumdar. On coresets for k-means and k-median clustering. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC '04)*, pages 291–300. Association for Computing Machinery, 2004.

[59] S. Har-Peled and M. Mendel. Fast construction of nets in low-dimensional metrics and their applications. *SIAM Journal on Computing*, 35(5):1148–1184, 2006.

[60] J. Hershberger. Smooth kinetic maintenance of clusters. *Computational Geometry*, 31(1–2):3–30, May 2005.

[61] P. Indyk. Sublinear time algorithms for metric space problems. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC '99)*, pages 428–434. Association for Computing Machinery, 1999.

[62] P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *Proceedings of the 41st IEEE Symposium on Foundations of Computer Science (FOCS '00)*, pages 189–197. IEEE Computer Society, 2000.

[63] P. Indyk. Algorithmic applications of low-distortion geometric embeddings. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS '01)*, pages 10–33. IEEE Computer Society, 2001.

[64] P. Indyk. Algorithms for dynamic geometric problems over data streams. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC '04)*, pages 373–380. Association for Computing Machinery, 2004.

[65] P. Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *Journal of the ACM (JACM)*, 53(3):307–323, May 2006.

[66] P. Indyk. Sketching, streaming and sub-linear space algorithms. Graduate course notes, available at `http://stellar.mit.edu/S/course/6/fa07/6.895/`, 2007.

[67] P. Indyk and J. Matousek. Low-distortion embeddings of finite metric spaces. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*. Chapman and Hall/CRC, 2004.

[68] K. Jain, M. Mahdian, E. Markakis, A. Saberi, and V. V. Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP. *Journal of the ACM (JACM)*, 50(6):795–824, November 2003.

[69] K. Jain, M. Mahdian, and A. Saberi. A new greedy approach for facility location problems. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC '02)*, pages 731–740. Association for Computing Machinery, 2002.

[70] K. Jain and V. V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation. *Journal of the ACM (JACM)*, 48(2):274–296, March 2001.

[71] W. Johnson and J. Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. In *Conference in Modern Analysis and Probability*, volume 26 of Contemporary Mathematics, pages 189–206. American Mathematical Society, 1984.

[72] D. M. Kane, J. Nelson, and D. P. Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings of the 29th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '10)*, pages 41–52. Association for Computing Machinery, 2010.

[73] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. A local search approximation algorithm for k-means clustering. *Computational Geometry*, 28(2–3):89–112, June 2004.

[74] D. R. Karger and M. Ruhl. Finding nearest neighbors in growth-restricted metrics. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC '02)*, pages 741–750. Association for Computing Machinery, 2002.

[75] J. M. Kleinberg, A. Slivkins, and T. Wexler. Triangulation and embedding using small sets of beacons. *Journal of the ACM (JACM)*, 56(6), September 2009.

[76] S. G. Kolliopoulos and S. Rao. A nearly linear-time approximation scheme for the Euclidean k-median problem. *SIAM Journal on Computing*, 37(3):757–782, 2007.

[77] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman. Analysis of a local search heuristic for facility location problems. *Journal of Algorithms*, 37(1):146–188, October 2000.

[78] C. Levcopoulos, G. Narasimhan, and M. H. M. Smid. Improved algorithms for constructing fault-tolerant spanners. *Algorithmica*, 32(1):144–156, 2002.

[79] J.-H. Lin and J. S. Vitter. Approximation algorithms for geometric median problems. *Information Processing Letters*, 44(5):245–249, 1992.

[80] S. P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, March 1982.

[81] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[82] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

[83] M. Mahdian, Y. Ye, and J. Zhang. Approximation algorithms for metric facility location problems. *SIAM Journal on Computing*, 36(2):411–432, 2006.

[84] J. Matousek. *Lectures on Discrete Geometry*. Graduate Texts in Mathematics. Springer, 1st edition, 2002.

[85] M. Matsumoto and T. Nishimura. Mersenne Twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, January 1998.

[86] C. J. H. McDiarmid. Concentration. In *Probabilistic Methods for Algorithmic Discrete Mathematics*, volume 16 of Algorithms and Combinatorics, pages 195–248. Springer, 1998.

[87] R. R. Mettu and C. G. Plaxton. The online median problem. *SIAM Journal on Computing*, 32(3):816–832, 2003.

[88] A. Meyerson. Online facility location. In *Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science (FOCS '01)*, pages 426–431. IEEE Computer Society, 2001.

[89] M. Mitzenmacher and E. Upfal. *Probability and Computing*. Cambridge University Press, 2005.

[90] T. Moscibroda and R. Wattenhofer. Facility location: Distributed approximation. In *Proceedings of the 24th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC '05)*, pages 108–117. Association for Computing Machinery, 2005.

[91] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[92] S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005.

[93] S. Muthukrishnan. Data stream algorithms. Notes from Barbados Complexity Theory Meeting, available at `http://sites.google.com/site/algoresearch/datastreamalgorithms`, 2009.

[94] G. Narasimhan and M. H. M. Smid. Approximating the stretch factor of Euclidean graphs. *SIAM Journal on Computing*, 30(3):978–989, 2000.

[95] N. Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, December 1992.

[96] L. O'Callaghan, A. Meyerson, R. Motwani, N. Mishra, and S. Guha. Streaming-data algorithms for high-quality clustering. In *Proceedings of the 18th International Conference on Data Engineering (ICDE '02)*, pages 685–696. IEEE Computer Society, 2002.

[97] S. Pandit and S. V. Pemmaraju. Return of the primal-dual: Distributed metric facility location. In *Proceedings of the 28th Annual ACM Symposium on Principles of Distributed Computing (PODC '09)*, pages 180–189. Association for Computing Machinery, 2009.

[98] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM Monographs on Discrete Mathematics and Applications, 2000.

[99] C. Scheideler. *Probabilistic Methods for Coordination Problems*. Habilitation thesis, University of Paderborn, 2000.

[100] S. Z. Selim and M. A. Ismail. k-means-type algorithms: A generalized convergence theorem and characterization of local optimality. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 6(1):81–87, January 1984.

[101] D. B. Shmoys. Approximation algorithms for facility location problems. In *Proceedings of the 3rd International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX '00)*, pages 27–33. Springer, 2000.

[102] D. B. Shmoys, É. Tardos, and K. Aardal. Approximation algorithms for facility location problems. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC '97)*, pages 265–274. Association for Computing Machinery, 1997.

[103] M. H. M. Smid. The well-separated pair decomposition and its applications. In T. F. Gonzalez, editor, *Handbook of Approximation Algorithms and Metaheuristics*. Chapman & Hall/CRC, 2007.

[104] M. Sviridenko. An improved approximation algorithm for the metric uncapacitated facility location problem. In *Proceedings of the 9th International Conference on Integer Programming and Combinatorial Optimization (IPCO '02)*, pages 240–257. Springer, 2002.

[105] K. Talwar. Bypassing the embedding: Algorithms for low dimensional metrics. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC '04)*, pages 281–290. Association for Computing Machinery, 2004.

[106] M. Thorup. Quick k-median, k-center, and facility location for sparse graphs. *SIAM Journal on Computing*, 34(2):405–432, 2004.

[107] B. Von Herzen and A. H. Barr. Accurate triangulations of deformed, intersecting surfaces. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '87)*, pages 103–110. Association for Computing Machinery, 1987.

[108] J. Vygen. Approximation algorithms for facility location problems (Lecture notes). Technical Report 05950-OR, Research Institute for Discrete Mathematics, University of Bonn, 2005. Available at `http://www.or.uni-bonn.de/~vygen/files/fl.pdf`.

[109] D. E. Willard and G. S. Lueker. Adding range restriction capability to dynamic data structures. *Journal of the ACM (JACM)*, 32(3):597–617, July 1985.

[110] B. Yao, F. Li, and P. Kumar. Reverse furthest neighbors in spatial databases. In *Proceedings of the 25th IEEE International Conference on Data Engineering (ICDE '09)*, pages 664–675. IEEE Computer Society, 2009.

[111] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: A new data clustering algorithm and its applications. *Data Mining and Knowledge Discovery*, 1(2):141–182, June 1997.