# Geometric Approximation Algorithms in the Online and Data Stream Models

by

Hamid Zarrabi-Zadeh

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2008

## Abstract

The online and data stream models of computation have recently attracted considerable research attention due to many real-world applications in various areas such as data mining, machine learning, distributed computing, and robotics. In both these models, input items arrive one at a time, and the algorithms must decide based on the partial data received so far, without any secure information about the data that will arrive in the future.

In this thesis, we investigate efficient algorithms for a number of fundamental geometric optimization problems in the online and data stream models. The problems studied in this thesis can be divided into two major categories: geometric clustering and computing various extent measures of a set of points.

In the online setting, we show that the basic unit clustering problem admits non-trivial algorithms even in the simplest one-dimensional case: we show that the naïve upper bounds on the competitive ratio of algorithms for this problem can be beaten using randomization. In the data stream model, we propose a new streaming algorithm for maintaining coresets of a set of points in fixed dimensions, and also, introduce a new simple framework for transforming a class of offline algorithms to their equivalents in the data stream model. These results together lead to improved $(1 + \varepsilon)$-approximation streaming algorithms for a wide variety of geometric optimization problems in fixed dimensions, including diameter, width, $k$-center, smallest enclosing ball, minimum-volume bounding box, minimum enclosing cylinder, minimum-width enclosing spherical shell/annulus, etc. In high-dimensional data streams, where the dimension is not a constant, we propose a simple streaming algorithm for the minimum enclosing ball (the 1-center) problem with an improved approximation factor.

**Acknowledgements**

I wish to express my profound and sincere gratitude to my supervisor, Timothy Chan, for giving me the opportunity to work with him and under his supervision. I am deeply indebted to him for his invaluable guidance, insightful discussions, and generous advice on research and non-research matters. It was a real pleasure for me to do my PhD under his supervision. His continuous encouragement and belief in me were vital in the advancement of my graduate career. For all this and more, I gratefully thank him.

I would also like to thank my thesis committee members for reading the thesis, providing helpful comments, and being present at my defence session. It was my pleasure to have Alex López-Ortiz and Therese Biedl on my advisory committee, and Prosenjit Bose and Shoja Chenouri on the defence committee. I am also thankful to Anna Lubiw for attending several of my talks and providing me with useful comments. I would also like to express my gratitude to my professors at Sharif University of Technology. In particular, I would like to thank my ex-supervisor, Mohammad Ghodsi, for his invaluable advice and insights, and for introducing me to the exciting world of computational geometry.

The working atmosphere in the Algorithms and Complexity group at the University of Waterloo became much more enjoyable with the company of many good friends. I express my best thanks to Peyman Afshani (my officemate), Ehsan Chiniforooshan, Reza Dorrigiv, Arash Farzan, Mehdi Mirzazadeh, Narges Simjour, and Omran Ahmadi, for being so friendly and sharing many enlightening discussions.

Finally, I am deeply thankful to my family for all the things they have done for my sake. I am grateful to my parents for their unconditional support, constant encouragement, and unflinching faith in me throughout my whole life. I am also deeply thankful to my wife, Zahra, who made my study easier with love and patience: without her constant support and encouragement this thesis would have been an impossibility. My lovely daughters, Mahya and Hana, were also supporting me with their little hearts. It is to them that I dedicate this thesis, with love and gratitude.

## Dedication

To my beloved wife, Zahra,
and to my lovely daughters, Mahya and Hana.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

In this thesis, we study a number of geometric optimization problems in the online and data stream models of computation. The problems studied are mainly related to geometric clustering and computing various extent measures of a set of points. In the following section, we provide basic definitions and concepts of the online and data stream models. Sections 1.2 and 1.3 describe two types of problems studied in this thesis, namely geometric clustering and computing extent measures of a set of points. In Section 1.4, we give a brief overview on the coreset framework which is a central tool used in Chapters 4 and 5. Section 1.5 outlines the results obtained in this thesis.

## 1.1 Online and Streaming Algorithms

Over the past two decades, *online algorithms* have received tremendous research attention. While in traditional design and analysis of algorithms we usually assume that the entire input is available to the algorithm before starting to generate the output, online algorithms have only access to a partial set of the input, and must generate their output based on that partial data available at present, without having a secure information about the relevant data that will arrive in the future. Online algorithms naturally arise in many areas such as computer science, operations research, distributed computing, and robotics.

Many problems are inherently online in the sense that they need immediate output for the input (requests) arrived so far in real time. Paging is one of the most well-known problems in this category. There is also a huge body of recent work on designing online algorithms for problems that have been originally studied in the traditional computational model.

In this thesis, we study online algorithms in terms of *competitive analysis*, i.e., we compare an online algorithm to an optimal offline algorithm that has access to the whole input in advance. An online algorithm $\mathcal{A}$ is called *c-competitive*, if for every input sequence $\sigma$, $\mathcal{A}(\sigma) \leq c \cdot \texttt{opt}(\sigma)$, where $\mathcal{A}(\sigma)$ is the cost of the online algorithm $\mathcal{A}$ on the input sequence $\sigma$, and $\texttt{opt}(\sigma)$ is the cost of the optimal offline algorithm on $\sigma$. In the above definition, $c$ is referred to as the *competitive ratio* of $\mathcal{A}$. If $\mathcal{A}$ is a randomized algorithm, then the *expected competitive ratio* of $\mathcal{A}$ is defined as $\max_\sigma \frac{E[\mathcal{A}(\sigma)]}{\texttt{opt}(\sigma)}$, where the expectation is taken over the randomized choices of the online algorithm. See the book by Borodin and El-Yaniv [15] for an in-depth study of competitive analysis through a wide range of examples.

The *data stream* model is another related model of computation that has recently attracted considerable attention. In this model, only one pass over the input is allowed and the algorithm has only a limited amount of working storage. This one-pass streaming model is attractive both in theory and in practice due to emerging applications involving massive data sets, since the entire input need not be stored and can be processed as elements arrive one at a time. See the survey by Muthukrishnan [84] on the growing literature on streaming algorithms.

The data stream and online models of computation have one important property in common: in both these two models input is given as a sequence over time and the algorithm must decide based on the partial information available, without having access to the whole data in advance. However, the main focus in these two models are different. In the online model, the solution must be constructed as each input arrive and decisions made to construct the solution in each step cannot be subsequently revoked. In the data stream model, however, the main concern is the amount of working space; as data items arrive, the streaming algorithm must decide which items should be kept in memory[1]. Since in the data stream model the entire input cannot be stored in the memory, streaming algorithms are restricted to use "sublinear" storage. On the other hand, online algorithms have no such restriction and can store all the data received so far. Moreover, streaming algorithms—unlike online algorithms— do not require immediate output; they should only keep enough information in memory to be able to produce an (approximate) solution on the items received so far upon request.

[1] Alternatively, a streaming algorithm may decide to keep some statistics about the data instead of a subset of the data items.

Some other related models have been considered in the literature, including the *dynamic streaming* model, in which both insertions and deletions are allowed using sublinear working storage (e.g., [46, 47, 67]), and the *sliding window* model, in which both insertions and deletions are allowed, but input items must be deleted in the same order as they are inserted (e.g., [23, 44]).

## 1.2 Clustering Problems

Clustering problems are fundamental and arise in a wide variety of applications such as information retrieval, data mining, unsupervised learning, and image analysis. In this thesis, we investigate two of the most basic and popular versions of clustering, namely, $k$-center and unit covering.

**$k$-Center.** The $k$-center problem is defined as follows: given a set of $n$ points and a parameter $k$, partition the points into $k$ clusters so as to minimize the maximum cluster radius. In one-dimensional Euclidean space, the $k$-center problem can be solved in $O(n \log n)$ time using dynamic programming [16, 83]. For $d \geq 2$ dimensions, the problem is well-known to be NP-hard [45, 82]. Moreover, it is NP-hard to approximate the two-dimensional $k$-center problem to within a factor smaller than 1.822 in the Euclidean metric, or smaller than 2 in the rectilinear metric [43].

Gonzalez [51] proposed a simple *furthest point* greedy algorithm that gives a 2-approximation to the $k$-center in any metric space. The algorithm repeatedly picks the point furthest away from the current selected set of centers as the next center to be added. Hochbaum and Shmoys [64, 65] obtained the same 2-factor approximation using a different approach. Gonzalez's algorithm can be implemented in $O(n \log k)$ deterministic time [43], or $O(n)$ expected time for all $k = O(n^{1/3}/\log n)$ [57].

Charikar *et al.* [25] introduced an online version of the $k$-center problem, called *incremental clustering*, in which every new point is either added to an existing cluster or placed in a new singleton cluster upon arrival. To prevent the number of clusters to exceed $k$, the algorithm is allowed to merge existing clusters at any point of time. Note that in this setting, clustering decisions are irrevocable because once formed, clusters cannot be broken up. Under this incremental setting, Charikar *et al.* proposed a greedy *doubling* algorithm that achieves competitive ratio 8 in any metric space, using $O(k)$ space and $O(k \log k)$ amortized time per update.

The above incremental algorithm can be viewed as a streaming algorithm for the $k$-center problem as well. Very recently, McCutchen and Khuller [80] gave a $(2+\varepsilon)$-approximation algorithm for the streaming $k$-center in any metric space based on a parallelization of Gonzalez's algorithm. In fixed-dimensional geometric space, a $(1+\varepsilon)$-approximation to the $k$-center can be maintained efficiently using the notion of coresets [57, 58].

**Unit Covering.** The unit covering problem is defined as follows: given a set of $n$ points, cover the points by balls of unit radius, so as to minimize the number of balls used. For $d \geq 2$ dimensions, the problems is NP-hard even in the rectilinear space [45]. In one dimension, the problem is equivalent to covering a set of points using a minimum number of unit intervals, and can be solved by a simple greedy algorithm in $O(n \log n)$ time [53]. While there is a lower bound of $\Omega(n \log n)$ on the running time of the algorithms for the one-dimensional unit covering problem [53], several output-sensitive algorithms with optimal $O(n \log k)$ running time are available for this problem [43, 85, 86, 91], where $k$ is the size of the optimal solution.

Hochbaum and Maass [63] gave a polynomial-time approximation scheme (PTAS) for the unit covering problem under the $L_\infty$ metric, using a simple shifted grid strategy. More precisely, for any fixed integer $l \geq 1$, their PTAS gives a factor $(1+1/l)^d$ approximation algorithm with running time $O(l^d n^{dl^d+1})$ for covering a set of points using $d$-dimensional axis-parallel unit boxes. Faster algorithms with constant approximation factors are provided in [43, 86]. Moreover, the problem is further studied under the $L_2$ metric in some recent work [20, 38, 48].

In the online setting, a sequence of points in $\mathbb{R}^d$ arrive over time and the online algorithm must cover each point using a unit $d$-dimensional ball upon arrival so as to minimize the total number of balls used. Charikar *et al.* [25] presented a simple online algorithm with competitive ratio $O(2^d d \log d)$ based on a geometric theorem due to Rogers [89], which states that $\mathbb{R}^d$ can be covered by any convex shape with covering density $O(d \log d)$. They also provided a lower bound of $\Omega(\log d / \log\log\log d)$ on the competitive ratio of any deterministic online algorithm for the unit covering problem in $d$ dimensions.

## 1.3 Extent Measures and Shape Fitting

Another major type of problems studied in this thesis involves computing various extent measures of a set of points. In general, an *extent measure* of a point set

$P \subseteq \mathbb{R}^d$ either computes some statistics about $P$ (such as diameter and the $k$-th largest distance between pairs of points in $P$), or computes some statistics of a shape enclosing $P$ (such as width, radius of the smallest enclosing ball, and volume of the smallest bounding hyperbox). Some of the extent measures studied in this thesis are listed below.

**Diameter.** The *diameter* of a point set is the maximum distance over all pairs of points. Given an $n$-point set $P \subseteq \mathbb{R}^d$, the diameter of $P$ can be computed in $O(dn^2)$ time using a brute-force algorithm. For $d = 2, 3$, there are algorithms that compute the diameter in optimal $O(n \log n)$ time [31, 88]. For $d \geq 4$, the current best running time is $O(n^{2-2/(\lceil d/2 \rceil+1)} \log^{O(1)} n)$ due to Agarwal, Matoušek and Suri [7].

**Width.** The *width* of a point set $P$ is the minimum distance between two parallel hyperplanes enclosing $P$. In two dimensions, the width can be computed in $O(n \log n)$ time by a simple scan over the convex hull of $P$. In three dimensions, Agarwal and Sharir [10] gave a randomized algorithm that runs in $O(n^{3/2+\delta})$ expected time for any constant $\delta > 0$. For $d \geq 4$, the current best algorithm has $O(n^{\lceil d/2 \rceil})$ running time [19].

**Smallest Enclosing Cylinder.** This problem asks for finding a *cylinder* of the minimum radius that encloses $P$. In two dimensions, this problem is identical to the width problem. For $d = 3$, Agarwal *et al.* [2] achieved an algorithm with running time $O(n^{3+\delta})$. In higher dimensions, a running time of $O(n^{2d-1+\delta})$ is attainable as noted in [19].

While exact computation of the extent measures is usually expensive, it has recently been established that a $(1 + \varepsilon)$-factor approximation for many of these extent-related problems can be computed efficiently in $O(n + 1/\varepsilon^c)$ time for some constant $c$ (depending on the problem and the dimension) using the general coreset framework [4, 21]. A brief introduction to this framework is provided in the next section.

Shape fitting is another well-studied problem which is closely related to computing extent measures of points. In this problem, one is asked to find a shape that best "fits" a given point set under some fitting criterion. A typical criterion which is usually used to measure how well a shape $\mathcal{S}$ fits the point set $P$ is the maximum distance between a point of $P$ and its nearest point in $\mathcal{S}$. For example, if the shape to fit is a line, the corresponding shape-fitting problem is equivalent to finding the

Table 1.1: Some basic shape-fitting problems.

| **Shape** | **Corresponding Shape-Fitting Problem** |
|---|---|
| point | smallest enclosing ball |
| line | minimum-radius enclosing cylinder |
| hyperplane | width |
| sphere/circle | minimum-width enclosing spherical shell/annulus |
| cylinder | minimum-width enclosing cylindrical shell |

minimum-radius cylinder enclosing $P$. Some basic shape-fitting problems are listed in Table 1.1. In this table, an *annulus* refers to the region between two concentric circles, with its width being the difference between the two circles radii. Similarly, a *spherical shell* is the region between two concentric spheres, and a *cylindrical shell* is the region between two coaxial cylinders.

Note that the $k$-center problem can be viewed as a shape-fitting problem as well in which the shape to fit is a set of $k$ points. Similarly, for a set of $k$ lines and $k$ hyperplanes, the corresponding shape-fitting problems are referred to as $k$-line-center and $k$-hyperplane-center respectively in the literature [9, 5].

## 1.4 The Coreset Framework

The coreset framework has recently emerged as a powerful tool for approximating various measures of a geometric data set. In this framework, a small subset of the input point set, called a *coreset*, is extracted in such a way that solving the optimization problem on the coreset yields an approximate solution to the entire set.

The main idea behind the coreset framework is simple: suppose we have an optimization problem for which a slow (exact or approximation) algorithm $\mathcal{A}$ is available that runs in $O(n^c)$ time, where $n$ is the size of the input. Suppose we have a mechanism to extract a subset of the input (i.e., the coreset) of size $(1/\varepsilon)^{c'}$ so that the optimal solution on the coreset is within $(1+\varepsilon)$ factor of the optimal solution on the entire set. If the algorithm for extracting the coreset is fast enough (typically linear in $n$), then, we can run the slow algorithm $\mathcal{A}$ on the coreset instead of the entire set to achieve a $(1+\varepsilon)$-approximate solution to the entire set in time $O(n + (1/\varepsilon)^{c''})$, where $c'' = c + c'$. This immediately leads to a linear-time approximation scheme (LTAS) for our optimization problem. This linear-time approximation scheme is of particular interest considering that the exact algorithms for many geometric problems are generally expensive (see for example the current best known exact algorithms mentioned in Section 1.3 for computing extent measures).

The main assumption made in the coreset framework is the existence of a coreset whose size is independent of the input size. Fortunately, it has been shown over the past years that for a wide range of geometric optimization problems, such a small-size coreset exists and can be extracted efficiently. One of the major improvements in this area is the unified approach introduced by Agarwal *et al.* [4]. They showed that computing coresets for many geometric problems reduces to computing a specific coreset which they call an $\varepsilon$-kernel. Roughly speaking, a subset $Q \subseteq P$ is called an $\varepsilon$-*kernel* of $P$ if for every slab $S$ containing $Q$, the $(1 + \varepsilon)$-expansion of $S$ contains $P$. Agarwal *et al.* showd that for any point set in $d$ dimensions, an $\varepsilon$-kernel of size $O(1/\varepsilon^{(d-1)/2})$ exists and can be extracted in linear time. The $\varepsilon$-kernel notion is extremely fundamental as it immediately yields coresets for many optimization problems such as diameter, width, smallest enclosing ball, minimum-volume enclosing box, and minimum-radius enclosing cylinder. Combined with the lifting technique proposed by Agarwal *et al.* [4], the $\varepsilon$-kernel notion also yields coresets for some other non-convex shape-fitting problems such as minimum-width enclosing spherical shell/annulus and minimum-width enclosing cylindrical shell.

The coreset framework is a fundamental tool for designing algorithms in the data stream model as well, as it allows to compute a measure approximately over the data stream by keeping only a small-size "sketch" of the input. See [4, 21, 12, 47, 60] on the growing literature of streaming algorithms developed using the notion of coresets. The coreset notion has been also proved to be useful for clustering problems such as $k$-center, $k$-median, $k$-mean, $k$-line center, etc. [60, 59, 58, 37, 61]. Other types of coresets have been studied in the literature, including coresets for moving points [3, 57], dynamic coresets [22, 47], coresets dealing with outliers [62, 6], and coresets for problems in higher (non-constant) dimensions [17, 18].

## 1.5 Results in This Thesis

In this thesis, we obtain new online and streaming algorithms for several geometric problems. An outline of our results is as follows. In Chapter 2, we began investigating an online clustering problem we call *online unit clustering*, which is extremely simple to state but turns out to be surprisingly nontrivial. The problem in one dimension is as follows: given a sequence of $n$ points on the real line, partition the points into clusters (subsets), each enclosable by a unit interval, with the objective of minimizing the number of clusters used.

In Chapter 2, we show that the naïve upper bound of 2 on the competitive ratio of the algorithms for online unit clustering in one dimension can be beaten using randomization. In particular, we present a randomized algorithm with expected competitive ratio $15/8 = 1.875$ using only two random bits. This is an interesting result, considering that ratio 2 is known to be tight (among both deterministic and randomized algorithms) for the related online unit covering problem where the position of each enclosing unit interval is fixed upon its creation. We further improve the competitive ratio of our algorithm to $11/6 = 1.833$ by adding an extra level of randomization. Our one-dimensional results can be extended to higher dimensions using a "dimension-reduction" approach, yielding a competitive ratio of $(\frac{11}{12}) \cdot 2^d$ for the $d$-dimensional problem under the $L_\infty$ metric.

The one-dimensional unit clustering problem is theoretically appealing because of its utter simplicity and its connection to some other well-known problems. In Chapter 3, we consider a closely-related problem of coloring co-interval graphs, and prove some lower bounds on the competitive ratio of algorithms for this problem. In particular, we show that no deterministic online algorithm for coloring unit co-interval graphs can be better than 3/2-competitive. Moreover, we show a lower bound of 4/3 on the competitive ratio of any randomized algorithm for coloring unit co-interval graphs. Both these two lower bounds hold for the online unit clustering problem as well. We also prove that for the class of (arbitrary rather than unit) co-interval graphs no randomized algorithm has competitive ratio better than 3/2.

In Chapter 4, we switch to the data stream model and employ the idea of coresets to develop efficient streaming algorithm for approximating various descriptors of the extent of a point set in fixed dimensions. The key tool that we use is the $\varepsilon$-kernel notion introduced by Agarwal *et al.* [4] as described in Section 1.4. We present a new streaming algorithm for maintaining an $\varepsilon$-kernel of a point set in $\mathbb{R}^d$ using $O((1/\varepsilon^{(d-1)/2})\log(1/\varepsilon))$ space. The space used by our algorithm is optimal up to a small logarithmic factor. This substantially improves (for any fixed dimension $d \geq 3$) the best previous algorithm for this problem that uses $O(1/\varepsilon^{d-(3/2)})$ space, presented by Agarwal and Yu [12]. Our algorithm immediately improves the space complexity of the best previous streaming algorithms for a number of fundamental geometric optimization problems in fixed dimensions, including width, minimum enclosing cylinder, minimum-width enclosing annulus, minimum-width enclosing cylindrical shell, etc.

In Chapter 5, we define a new class of core-preserving algorithms and present a simple framework for converting each algorithm in this class to an efficient algorithm with the same space complexity in the data stream model. As an application of this framework, we show that for a set of points in fixed dimensions, additive and multiplicative $\varepsilon$-coresets for the $k$-center problem can be maintained in $O(1)$ and $O(k)$ time respectively, using a data structure whose size is independent of the size of the input. This independence on the input size is of particular interest as the input size in the data streams is typically huge. We also provide a faster streaming algorithm for maintaining $\varepsilon$-coresets for fat extent-related problems such as diameter and minimum enclosing ball in fixed dimensions.

In Chapter 6, we focus on the minimum enclosing ball (the 1-center) problem in high-dimensional data streams, where the dimension $d$ may be arbitrarily large. Though our results in Chapters 4 and 5 provide efficient streaming algorithms for this problem in any $d$ dimensions, the dependency of their space complexity on $d$ is exponential (i.e., $O(1/\varepsilon^{O(d)})$). Therefore, our main concern here is to obtain approximation algorithms that avoid the exponential (or superpolynomial) dependency on the dimension. The only previous known result for this problem was a naïve 2-approximation algorithm. We improve this constant factor to 3/2 by a simple one-pass algorithm that uses $O(d)$ time per point and just $O(d)$ space. Chapter 7 concludes with some open problems and directions for future research.

**Remark.** Results of this thesis have been published in the proceedings of the following conferences: ESA 2008 [99], CCCG 2008 [100], COCOON 2007 [102], CSICC 2007 [98], WAOA 2006 [24], and CCCG 2006 [101].

# Chapter 2

# Online Unit Clustering

In this chapter, we consider the online unit clustering problem and show that in the one-dimensional case, the naïve upper bound of 2 on the competitive ratio can be beaten using randomization. We present two new randomized algorithms for this problem: the first algorithm achieves a competitive ratio of 15/8 using only 2 random bits, and the second one yields an improved competitive ratio of 11/6 using $k$ random bits, where $k$ is the size of the optimal solution. We also show how our algorithm can be extended to two and higher dimensions.

## 2.1 Introduction

Clustering problems—dividing a set of points into groups to optimize various objective functions—are fundamental and arise in a wide variety of applications such as information retrieval, data mining, and facility location. The followings are two of the most basic and popular versions of clustering:

**Problem 1 ($k$-Center)** *Given a set of $n$ points and a parameter $k$, cover the set by $k$ congruent balls, so as to minimize the radius of the balls.*

**Problem 2 (Unit Covering)** *Given a set of $n$ points, cover the set by balls of unit radius, so as to minimize the number of balls used.*

Extended abstracts of this chapter have been previously presented at WAOA 2006 [24] and COCOON 2007 [102]. The full versions are to appear in special issues of *Theory of Computing Systems* and *Algorithmica*.

Both problems are NP-hard in the Euclidean plane [45, 82]. Factor-2 algorithms are known for the $k$-center problem [43, 51] in any dimension, while polynomial-time approximation schemes are known for the unit covering problem [63] in fixed dimensions. A more detailed background study of these two problems has been provided in Section 1.2.

Recently, many researchers have considered clustering problems in the online and data stream models [25, 26, 52], where the input is given as a sequence of points over time. In the online model, the solution must be constructed as points arrive and decisions made cannot be subsequently revoked; for example, in the unit covering problem, after a ball is opened to cover an incoming point, the ball cannot be removed later. The online version of the unit covering problem is one of the problems addressed in the paper by Charikar *et al.* [25]. They have given an upper bound of $O(2^d d \log d)$ and a lower bound of $\Omega(\log d/ \log\log\log d)$ on the competitive ratio of deterministic online algorithms in $d$ dimensions; for $d = 1$ and 2, the lower bounds are 2 and 4 respectively.

In this chapter, we address the online version of the following variant:

**Problem 3 (Unit Clustering)** *Given a set of $n$ points, partition the set into clusters (subsets), each of radius at most one, so as to minimize the number of clusters used. Here, the* radius *of a cluster refers to the radius of its smallest enclosing ball.*

At first glance, Problem 3 might look eerily similar to Problem 2; in fact, in the usual offline setting, they are identical. However, in the online setting, there is one important difference: as a point $p$ arrives, the unit clustering problem only requires us to decide on the choice of the cluster containing $p$, not the ball covering the cluster; the point cannot subsequently be reassigned to another cluster, but the position of the ball may be shifted.

We show that it is possible to get better results for Problem 3 than Problem 2. Interestingly we show that even in one dimension, the unit clustering problem admits a nontrivial algorithm with competitive ratio better than 2, albeit by using randomization. In contrast, we show that such a result is not possible for unit covering. To be precise, we present an online algorithm for one-dimensional unit clustering that achieves expected competitive ratio $15/8 = 1.875$ against oblivious adversaries. Our algorithm is not complicated but does require a combination of ideas and a careful case analysis. We further improve our result and obtain a randomized online algorithm with expected competitive ratio at most $11/6 \approx$

1.8333, using an extra level of randomization. We also extend our algorithm for the problem in higher dimensions under the $L_\infty$ metric.

We believe that the one-dimensional unit clustering problem itself is theoretically appealing because of its utter simplicity and its connection to well-known problems. For example, in the exact offline setting, one-dimensional unit clustering/covering is known to be equivalent to the dual problem of finding a largest subset of disjoint intervals among a given set of unit intervals—i.e., finding maximum independent sets in unit interval graphs. Higher-dimensional generalizations of this dual independent set problem have been explored in the map labeling and computational geometry literature [11, 20, 42], and online algorithms for various problems about geometric intersection graphs have been considered (such as [77]). The one-dimensional independent set problem can also be viewed as a simple scheduling problem (dubbed "activity selection" by Cormen et al. [32]), and various online algorithms about intervals and interval graphs (such as [1, 40, 71, 74]) have been addressed in the literature on scheduling and resource allocation. In the online setting, one-dimensional unit clustering is equivalent to clique partitioning in unit interval graphs, and thus equivalent to coloring unit co-interval graphs. More details on this related coloring problem will be provided in Chapter 3.

## 2.2 Naïve Algorithms

In this section, we begin our study of the unit clustering problem in one dimension by pointing out the deficiencies of some natural strategies.

Recall that the goal is to assign points to clusters so that each cluster has length at most 1, where the *length* of a cluster refers to the length of its smallest enclosing interval. (Note that we have switched to using lengths instead of radii in one dimension; all intervals are closed.) We say that a point *lies* in a cluster if inserting it to the cluster would not increase the length of the cluster. We say that a point *fits* in a cluster if inserting it to the cluster would not cause the length to exceed 1. The following are three simple online algorithms, all easily provable to have competitive ratio at most 2:

**Algorithm 1** (CENTERED) *For each new point $p$, if it is covered by an existing interval, put $p$ in the corresponding cluster, else open a new cluster for the unit interval centered at $p$.*

**Algorithm 2** (Grid) *Build a uniform unit grid on the line (where cells are intervals of the form $[i, i+1)$). For each new point $p$, if the grid cell containing $p$ is nonempty, put $p$ in the corresponding cluster, else open a new cluster for the grid cell.*

**Algorithm 3** (Greedy) *For each new point $p$, if $p$ fits in some existing cluster, put $p$ in such a cluster, else open a new cluster for $p$.*

The first two algorithms actually solve the stronger unit covering problem (Problem 2). No such algorithms can break the 2 bound, as we can easily prove:

**Theorem 2.1** *There is a lower bound of 2 on the competitive ratio of any randomized (and deterministic) algorithm for the online unit covering problem in one dimension.*

*Proof.* To show the lower bound for randomized algorithms, we use Yao's technique [96] and provide a probability distribution on the input sequences such that the resulting expected competitive ratio for any deterministic online algorithm is at least 2. The adversary provides a sequence of 3 points at position 1, $x$, and $1 + x$, where $x$ is uniformly distributed in $[0, 1]$. The probability that a deterministic algorithm produces the optimal solution (of size 1 instead of 2 or more) is 0. Thus, the expected value of the competitive ratio is at least 2. $\square$

The 2 bound on the competitive ratio is also tight for Algorithm 3: just consider the sequence $\langle \frac{1}{2}, \frac{3}{2}, \ldots, 2k - \frac{1}{2} \rangle$ followed by $\langle 0, 2, \ldots, 2k \rangle$ (where the greedy algorithm uses $2k+1$ clusters and the optimal solution needs only $k+1$ clusters). No random combination of Algorithms 1–3 can lead to a better competitive ratio, as we can easily see by the same bad example. New ideas are needed to beat 2.

## 2.3 The New Algorithm

In this section, we present a new randomized algorithm for the online unit clustering problem. While the competitive ratio of this algorithm is not necessarily less than 2, the algorithm is designed so that when combined with Algorithm 2 we get a competitive ratio strictly less than 2.

Our algorithm builds upon the simple grid strategy (Algorithm 2). To guard against a bad example like $\langle \frac{1}{2}, \frac{3}{2}, \ldots \rangle$, the idea is to allow two points in different grid cells to be put in a common cluster “occasionally” (as controlled by randomization). Doing so might actually hurt, not help, in many cases, but fortunately we can still show that there is a net benefit (in expectation), at least in the most critical case.

To implement this idea, we form *windows* each consisting of two grid cells and permit clusters crossing the two cells within a window but try to “discourage” clusters crossing two windows. There are two ways to form windows over the grid; we choose which according to an initial random bit. The details of the algorithm are delicate and are described below[1].

---

**Algorithm 4** (RANDWINDOW) Partition the line into windows each of the form $[2i, 2i+2)$. With probability $1/2$, shift all windows one unit to the right. For each new point $p$, find the window $w$ containing $p$, and do the following:

```
1: if $p$ fits in a cluster intersecting $w$ then
2:     put $p$ in the “closest” such cluster
3: else if $p$ fits in a cluster $u$ inside a neighboring window $w'$ then
4:     if $w$ contains at least 1 cluster and $w'$ contains at least 2 clusters then
5:         put $p$ in $u$
6: if $p$ is not put in any cluster then
7:     open a new cluster for $p$
```

---

The algorithm is greedy-like and opens a new cluster only if no existing cluster fits. The main exception arises from the (seemingly mysterious) condition in line 4.

When more than one cluster fits in line 1, we put the point in the “closest” such cluster, specified by the following two preference rules:

- RULE I. If $p$ lies in a cluster $u$, then $u$ is the closest cluster to $p$.
- RULE II. If $p$ lies in a cell $c$, then any cluster intersecting $c$ is closer to $p$ than any cluster contained in a neighboring cell.

The first preference rule prevents clusters from overlapping each other, and the second rule prevents clusters from unnecessarily crossing the boundary of two neighboring cells. (In the second rule, if more than one intersecting cluster exists, any

[1] The pseudocode provided here is slightly different from the original one presented in [24]. Here, we have tried to make our algorithm compatible with the algorithm presented in the next section, in order to make proofs presented here reusable in the next section.

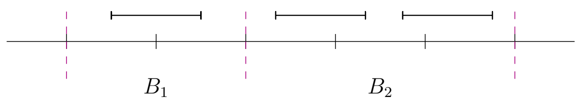

Figure 2.1: Two blocks of sizes 2 and 3.

of them can be arbitrarily chosen as the closest.) These exceptional cases and preference rules are vital to the analysis.

### 2.3.1 Preliminaries for the Analysis

Let $\sigma$ be the input sequence. We denote by $\texttt{opt}(\sigma)$ the optimal offline solution obtained by the following greedy algorithm: sort all points in $\sigma$ from left to right; cover the leftmost point $p$ and all points within unit distance of it by a unit interval started at $p$; and repeat the procedure for the remaining uncovered points. Obviously, the unit intervals obtained by this algorithm are disjoint.

We refer to a cluster as a *crossing cluster* if it intersects two adjacent grid cells, or as a *whole cluster* if it is contained completely in a grid cell.

For any real interval $x$ (e.g., a grid cell or a group of consecutive cells), let $\mu_h(x)$ denote the number of whole clusters contained in $x$, and let $\mu_c(x)$ denote the number of clusters crossing the boundaries of $x$, in the solution produced by our algorithm. The *cost* of $x$, denoted by $\mu(x)$, is then defined as

$$\mu(x) = \mu_h(x) + \frac{1}{2}\mu_c(x).$$

We note that $\mu$ is additive, i.e., for two adjacent intervals $x$ and $y$, $\mu(x \cup y) = \mu(x) + \mu(y)$.

A set of $k$ consecutive grid cells containing $k-1$ intervals from $\texttt{opt}(\sigma)$ is called a *block* of size $k$ (see Fig. 2.1). We define $\rho(k)$ to be the expected competitive ratio of the RandWindow algorithm within a block of size $k$. In other words, $\rho(k)$ upper-bounds the expected value of $\mu(B)/(k-1)$ over all blocks $B$ of size $k$.

In the following, a list of objects (e.g., grid cells or clusters) denoted by $\langle x_i, \ldots, x_j \rangle$ is always implicitly assumed to be ordered from left to right on the line. Moreover, $p_1 \ll p_2$ denotes the fact that point $p_1$ arrives before point $p_2$ in the input sequence.

We now establish some observations concerning the behavior of our randomized algorithm that will be used multiple times in the analysis in this and the next section.

**Observation 2.2**

(i) *The enclosing intervals of the clusters are disjoint.*

(ii) *Any grid cell $c$ can contain at most one whole cluster. Thus, we always have $\mu(c) \leq 1 + \frac{1}{2} + \frac{1}{2} = 2$.*

(iii) *If a grid cell $c$ intersects a crossing cluster $u_1$ and a whole cluster $u_2$, then $u_2$ must be opened after $u_1$ has been opened, and after $u_1$ has become a crossing cluster.*

*Proof.* (i) holds because of preference Rule I. (ii) Let $u_1$ and $u_2$ be two whole clusters contained in the grid cell $c$ and suppose that $u_1$ is opened before $u_2$. Then all points of $u_2$ would be assigned to $u_1$, because line 2 precedes line 7.

For (iii), let $p_1$ be the first point of $u_1$ in $c$ and $p_1'$ be the first point of $u_1$ in a cell adjacent to $c$. Let $p_2$ be the first point of $u_2$. Among these three points, $p_1$ cannot be the last to arrive: otherwise, $p_1$ would be assigned to the whole cluster $u_2$ instead of $u_1$, because of Rule II. Furthermore, $p_1'$ cannot be the last to arrive: otherwise, $p_1$ would be assigned to $u_2$ instead. So, $p_2$ must be the last to arrive. □

**Observation 2.3** *Let $u_1$ be a whole cluster contained in a grid cell $c$, and let $u_2$ and $u_3$ be two clusters crossing the boundaries of $c$. Then*

(i) *$u_1$ and $u_2$ cannot be entirely contained in the same interval from $\texttt{opt}(\sigma)$.*

(ii) *there are no two intervals $I_1$ and $I_2$ in $\texttt{opt}(\sigma)$ such that $u_1 \cup u_2 \cup u_3 \subseteq I_1 \cup I_2$.*

*Proof.* (i) Suppose by way of contradiction that $u_1$ and $u_2$ are entirely contained in an interval $I$ from $\texttt{opt}(\sigma)$. Then by Observation 2.2(iii), $u_1$ is opened after $u_2$ has become a crossing cluster, but then the points of $u_1$ would be assigned to $u_2$ instead: a contradiction.

(ii) Suppose that $u_1 \cup u_2 \cup u_3 \subseteq I_1 \cup I_2$, where $I_1$ and $I_2$ are the two intervals from $\texttt{opt}(\sigma)$ intersecting $c$. We now proceed as in part (i). By Observation 2.2(iii), $u_1$ is opened after $u_2$ and $u_3$ have become crossing clusters, but then the points of $u_1$ would be assigned to $u_2$ or $u_3$ instead: a contradiction. □

**Lemma 2.4** *Let $B = \langle c_1, \ldots, c_k \rangle$ be a block of size $k \geq 2$, and $S$ be the set of all odd-indexed (or even-indexed) cells in $B$. Then there exists a cell $c \in S$ such that $\mu(c) < 2$.*

*Proof.* Let $\langle I_1, \ldots, I_{k-1}\rangle$ be the $k-1$ intervals from $\texttt{opt}(\sigma)$ in $B$, where each interval $I_i$ intersects two cells $c_i$ and $c_{i+1}$ $(1 \leq i \leq k-1)$. Let $O$ represent the set of all odd integers between 1 and $k$. We first prove the lemma for the odd-indexed cells.

Suppose by way of contradiction that for each $i \in O$, $\mu(c_i) = 2$. It means that for each $i \in O$, $c_i$ intersects three clusters $\langle u_i^\ell, u_i, u_i^r\rangle$, where $u_i$ is a whole cluster, and $u_i^\ell$ and $u_i^r$ are two crossing clusters. We prove inductively that for each $i \in O$, $u_i \cap I_i \neq \emptyset$ and $u_i^r \cap I_{i+1} \neq \emptyset$.

Base Case: $u_1 \cap I_1 \neq \emptyset$ and $u_1^r \cap I_2 \neq \emptyset$.

The first part is trivial, because $c_1$ intersects just $I_1$, and hence, $u_1 \subseteq I_1$. The second part is implied by Observation 2.3(i), because $u_1$ and $u_1^r$ cannot be entirely contained in $I_1$.

Inductive Step: $u_i \cap I_i \neq \emptyset \,\wedge\, u_i^r \cap I_{i+1} \neq \emptyset \Rightarrow u_{i+2} \cap I_{i+2} \neq \emptyset \,\wedge\, u_{i+2}^r \cap I_{i+3} \neq \emptyset$.

Suppose by contradiction that $u_{i+2} \cap I_{i+2} = \emptyset$. Therefore, $u_{i+2}$ must be entirely contained in $I_{i+1}$. On the other hand, $u_i^r \cap I_{i+1} \neq \emptyset$ implies that $u_{i+2}^\ell$ is entirely contained in $I_{i+1}$. But this is a contradiction, because $u_{i+2}$ and $u_{i+2}^\ell$ are contained in the same interval, which is impossible by Observation 2.3(i).

Now, suppose that $u_{i+2}^r \cap I_{i+3} = \emptyset$. Since $u_i^r \cap I_{i+1} \neq \emptyset$, and clusters do not overlap, $u_{i+2}^\ell$, $u_{i+2}$, and $u_{i+2}^r$ should be contained in $I_{i+1} \cup I_{i+2}$, which is impossible by Observation 2.3(ii).

Repeating the inductive step zero or more times, we end up at either $i = k$ or $i = k-1$. If $i = k$, then $u_k \cap I_k \neq \emptyset$ which is a contradiction, because there is no $I_k$. If $i = k-1$, then $u_{k-1}^r \cap I_k \neq \emptyset$ which is again a contradiction, because we have no $I_k$.

Both cases lead to contradiction. It means that there exists some $i \in O$ such that $\mu(c_i) < 2$. The proof for the even-indexed cells is similar. The only difference is that we need to prove the base case for $i = 2$, which is easy to do with Observations 2.3(i) and 2.3(ii). $\square$

**Lemma 2.5** *If $B$ is a block of size $k \geq 2$, then $\mu(B) \leq 2k - 1$.*

*Proof.* This is a direct corollary of Lemma 2.4, because there are at least two cells in $B$ (one odd-indexed and one even-indexed) that have cost at most 3/2, and the other cells have cost at most 2. $\square$

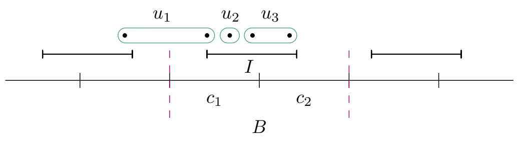

Figure 2.2: Impossibility of Subcase 1.1.

### 2.3.2 Analysis

We are now ready to analyze the expected competitive ratio of our algorithm within a block of size $k \geq 2$. The required case analysis is delicate and is described in detail below. The main case to watch out for is $k = 2$: any bound for $\rho(2)$ strictly smaller than 2 will lead to a competitive ratio strictly smaller than 2 for the final algorithm (as we will see in Section 2.3.3), although bounds for $\rho(3), \rho(4), \ldots$ will affect the final constant.

**Theorem 2.6** $\rho(2) = 7/4$.

*Proof.* Consider a block $B$ of size 2, consisting of cells $\langle c_1, c_2 \rangle$. Let $I$ be the single unit interval in $B$ in $\texttt{opt}(\sigma)$. There are two possibilities:

- Lucky Case: $B$ falls completely in one window $w$. After a cluster $u$ has been opened for the new point (by line 7), all subsequent points in $I$ are put in the same cluster $u$ (by line 2). Note that the condition put in line 4 prevents points from the neighboring windows to join $u$ and make crossing clusters. So, $u$ is the only cluster in $B$, and hence, $\mu(B) = 1$.

- Unlucky Case: $B$ is split between two neighboring windows. We first rule out some subcases:

  - Subcase 1.1: $\mu(c_1) = 2$. Here, $c_1$ intersects three clusters $\langle u_1, u_2, u_3 \rangle$ (from left to right), where $u_1$ and $u_3$ are crossing clusters and $u_2$ is a whole cluster (see Fig. 2.2). By Observation 2.2(iii), $u_2$ is opened after $u_3$ has become a crossing cluster, but then the points of $u_2$ would be assigned to $u_3$ instead (because line 2 precedes line 7 and $u_2 \cup u_3 \subset I$ has length at most 1): a contradiction.
  - Subcase 1.2: $\mu(c_2) = 2$. Similarly impossible.
  - Subcase 1.3: $\mu(c_1) = \mu(c_2) = 3/2$. We have only two scenarios:

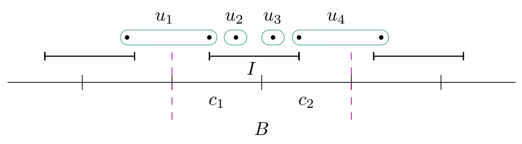

Figure 2.3: Impossibility of Subsubcase 1.3.2.

- ∗ SUBSUBCASE 1.3.1: $B$ intersects three clusters $\langle u_1, u_2, u_3 \rangle$, where $u_2$ is a crossing cluster, and $u_1$ and $u_3$ are whole clusters. By Observation 2.2(iii), $u_1$ is opened after $u_2$ has become a crossing cluster, but then the points of $u_1$ would be assigned to $u_2$ instead (because of line 2 and $u_1 \cup u_2 \subset I$): a contradiction.
- ∗ SUBSUBCASE 1.3.2: $B$ intersects four clusters $\langle u_1, u_2, u_3, u_4 \rangle$, where $u_1$ and $u_4$ are crossing clusters and $u_2$ and $u_3$ are whole clusters (see Fig. 2.3). W.l.o.g., say $u_2$ is opened after $u_3$. By Observation 2.2(iii), $u_2$ is the last to be opened after $u_1, u_3, u_4$, but then $u_2$ would not be opened as points in $u_2$ may be assigned to $u_3$ (because lines 3–5 precede line 7, $u_2 \cup u_3 \subset I$, and $c_2$ intersects more than one cluster): a contradiction.

In all remaining subcases, $\mu(B) = \mu(c_1) + \mu(c_2) \leq \frac{3}{2} + 1 = \frac{5}{2}$.

Since the lucky case occurs with probability exactly 1/2, we conclude that $\rho(2) \leq \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot \frac{5}{2} = \frac{7}{4}$. This bound is tight: to see this just consider the block $B = [1, 3)$ and the sequence of points $\langle 0.5, 1.5, 2.5, 1.8 \rangle$. □

**Theorem 2.7** *$\rho(3) = 9/4$.*

*Proof.* Consider a block $B$ of size 3, consisting of cells $\langle c_1, c_2, c_3 \rangle$. We assume w.l.o.g. that $c_1$ and $c_2$ fall in the same window (the other scenario is symmetric).

- SUBCASE 1.1: $\mu(c_2) = 2$. Impossible by Lemma 2.4.
- SUBCASE 1.2: $\mu(c_1) = \mu(c_3) = 2$. Impossible by Lemma 2.4.
- SUBCASE 1.3: $\mu(c_1) = 2$ and $\mu(c_2) = \mu(c_3) = 3/2$. Here, $B$ intersects six clusters $\langle u_1, \ldots, u_6 \rangle$, where $u_1, u_3, u_6$ are crossing clusters and $u_2, u_4, u_5$ are whole clusters. Let $\langle I_1, I_2 \rangle$ be the two unit intervals in $B$ in $\texttt{opt}(\sigma)$. By Observation 2.3(i), $u_3$ cannot be entirely contained in $I_1$. This implies that $u_4 \cup u_5 \subset I_2$. Now suppose w.l.o.g. that $u_4$ is opened after $u_5$. By

Observation 2.2(iii), $u_4$ is the last to be opened after $u_3, u_5, u_6$. Consider the first point $p$ in $u_4$. Upon arrival of $p$, the window containing $p$ contains at least one cluster, $u_3$, and the neighboring window contains two clusters $u_5$ and $u_6$. Therefore, by the condition in line 4, the algorithm would assign $p$ to $u_5$ instead of $u_4$, which is a contradiction.

- SUBCASE 1.4: $\mu(c_1) = \mu(c_2) = 3/2$ and $\mu(c_3) = 2$. Similarly impossible.

In all remaining subcases, $\mu(B) = \mu(c_1) + \mu(c_2) + \mu(c_3)$ is at most $2 + \frac{3}{2} + 1 = \frac{9}{2}$ or $\frac{3}{2} + \frac{3}{2} + \frac{3}{2} = \frac{9}{2}$. We conclude that $\rho(3) \leq 9/4$. This bound is tight: to see this just consider the block $B = [2, 5)$, and the sequence of points

$$\sigma = \langle 0.2, 6.8, 2.6, 3.4, 3.6, 2.4, 3.8, 4.4, 3.2, 1.4, 4.6, 5.6, 2.5, 4.5\rangle$$

that realizes $\mu(B) = 9/2$ regardless of the windowing selected. □

**Theorem 2.8** *$\rho(k) \leq (2k-1)/(k-1)$ for all $k \geq 4$.*

*Proof.* This is a direct corollary of Lemma 2.5. □

While RANDWINDOW is better than 2-competitive on blocks of size two, its competitive ratio exceeds 2 for larger block sizes. In the next subsection we utilize a combination strategy to overcome this deficiency.

### 2.3.3 The Combined Algorithm

We combine the RANDWINDOW algorithm (Algorithm 4) with the GRID algorithm (Algorithm 2) to obtain a randomized online algorithm with competitive ratio strictly less than 2. Note that only two random bits in total are used at the beginning.

**Algorithm 5 (COMBO)** *With probability* $1/2$*, run* RANDWINDOW*, else run* GRID.

**Theorem 2.9** COMBO *is* $15/8$*-competitive (against oblivious adversaries).*

*Proof.* The GRID algorithm uses exactly $k$ clusters on a block of size $k$. Therefore, the competitive ratio of this algorithm within a block of size $k$ is $k/(k-1)$.

Table 2.1: Upper bounds on the competitive ratio of Grid, RandWindow, and Combo within a block.

| **Block Size** | **2** | **3** | $k \geq \mathbf{4}$ |
|---|---|---|---|
| Grid | 2 | 3/2 | $k/(k-1)$ |
| RandWindow | 7/4 | 9/4 | $(2k-1)/(k-1)$ |
| Combo | 15/8 | 15/8 | $(3k-1)/(2k-2)$ |

Table 2.1 shows the competitive ratio of the RandWindow, Grid, and Combo algorithms, for all possible block sizes.

As we can see, the competitive ratio of Combo within a block is always at most 15/8. By summing over all blocks and exploiting the additivity of our cost function $\mu$, we see that expected total cost of the solution produced by Combo is at most 15/8 times the size of $\texttt{opt}(\sigma)$ for every input sequence $\sigma$. $\square$

## 2.4 Using More Randomization

In the previous section, we showed that it is possible to obtain an online algorithm for one-dimensional unit clustering with expected competitive ratio strictly less than 2 using randomization. In this section, we improve our previous result further and obtain a randomized online algorithm with expected competitive ratio at most $11/6 \approx 1.8333$, improving the previous ratio of $15/8 = 1.875$.

The new algorithm is built upon the RandWindow algorithm presented in the previous section. The key difference here is to make more use of randomization (the previous algorithm requires only 2 random bits). In the previous algorithm, clusters crossing two adjacent windows are "discouraged"; in the new algorithm, crossings of adjacent windows are discouraged to a "lesser" extent, as controlled by randomization. This calls for a subtle change in the algorithm, as well as a lengthier case analysis.

### 2.4.1 The Improved Algorithm

In this section, we modify the RandWindow algorithm form the previous section. The competitive ratio of the algorithm is still not less than 2, but will become less than 2 when combined with the naïve grid strategy as described in Section 2.4.2.

In the previous algorithm (RandWindow), clusters crossing two adjacent windows were not strictly forbidden but were discouraged in some sense. In the new algorithm, the idea, roughly speaking, is to permit more clusters crossing windows. More specifically, call the grid point lying between two adjacent windows a *border*; generate a random bit for every border, where a 1 bit indicates an *open* border and a 0 bit indicates a *closed* border. Clusters crossing closed borders are still discouraged, but not clusters crossing open borders. (As it turns out, setting the probability of border opening/closing to 1/2 is the best choice.)

The new algorithm is given below. In this pseudocode, $b(w, w')$ refers to the border indicator between windows $w$ and $w'$.

---

**Algorithm 6 (RandBorder)** Partition the line into windows each of the form $[2i, 2i + 2)$. With probability 1/2, shift all windows one unit to the right. For each two neighboring windows $w$ and $w'$ set $b(w, w')$ to a number uniformly drawn at random from $\{0, 1\}$. For each new point $p$, find the window $w$ containing $p$, and do the following:

```
1: if $p$ fits in a cluster intersecting $w$ then
2:     put $p$ in the "closest" such cluster
3: else if $p$ fits in a cluster $u$ inside a neighboring window $w'$ then
4:     if $b(w, w') = 1$ then put $p$ in $u$
5:     if $w$ contains at least 1 cluster and $w'$ contains at least 2 clusters then
6:         put $p$ in $u$
7: if $p$ is not put in any cluster then open a new cluster for $p$
```

---

Thus, a cluster is allowed to cross the boundary of two grid cells within a window freely, but it can cross the boundary of two adjacent windows only in two exceptional cases: when the corresponding border indicator is set to 1, or when the condition specified in line 5 arises. We will see the rationale for this condition during the analysis.

Like in the previous algorithm, the "closest" cluster in line 2 is specified by the following two preference rules:

- Rule I. If $p$ lies in a cluster $u$, then $u$ is the closest cluster to $p$.
- Rule II. If $p$ lies in a cell $c$, then any cluster intersecting $c$ is closer to $p$ than any cluster contained in a neighboring cell.

Note that the random bits used for the border indicators can be easily generated on the fly, upon inspecting each border in line 4. It is easy to observe that the number of random bits used in this way is at most the number of intervals in the optimal solution.

### 2.4.2 Analysis

In the RandBorder algorithm, if all border indicators are set to 0, the algorithm simply becomes identical to RandWindow. This enables us to reuse most of the analysis presented in the previous section. In particular, it is easy to verify that all observations and lemmas provided in Section 2.3.1 are still valid for the RandBorder algorithm.

We now establish some new observations concerning the behavior of the RandBorder algorithm.

**Observation 2.10** *Any interval in* $\texttt{opt}(\sigma)$ *that does not cross a closed border can contain at most one whole cluster.*

*Proof.* Let $u_1$ and $u_2$ be two whole clusters contained in the said interval and suppose that $u_1$ is opened before $u_2$. Then all points of $u_2$ would be assigned to $u_1$, because lines 2 and 4 precede line 7. □

**Lemma 2.11** *Let $B$ be a block of size $k \geq 2$. If all borders strictly inside $B$ are open, then $\mu(B) \leq 2(k-1)$.*

*Proof.* This is immediate from the fact that each block of size $k \geq 2$ contains exactly $k-1$ intervals from $\texttt{opt}(\sigma)$, and that each of these $k-1$ intervals has cost at most 2 by Observation 2.10. □

In the rest of this section, we define $\rho(k)$ to be the expected competitive ratio of the RandBorder algorithm within a block of size $k$. In other words, $\rho(k)$ upper-bounds the expected value of $\mu(B)/(k-1)$ over all blocks $B$ of size $k$.

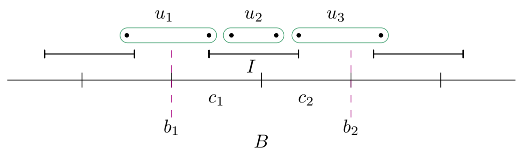

Figure 2.4: Illustration of Subcase 1.3.

**Theorem 2.12** $\rho(2) = 27/16$.

*Proof.* Consider a block $B$ of size 2, consisting of two cells $\langle c_1, c_2\rangle$ (see Fig. 2.4). Let $I$ be the single unit interval in $B$ in $\texttt{opt}(\sigma)$. There are two possibilities.

CASE 1: $B$ falls completely in one window $w$. Let $\langle b_1, b_2\rangle$ be the two border indicators at the boundaries of $w$. Let $p_0$ be the first point to arrive in $I$. W.l.o.g., assume $p_0$ is in $c_2$ (the other case is symmetric). We consider four subcases.

- SUBCASE 1.1: $\langle b_1, b_2\rangle = \langle 0, 0\rangle$. Here, both boundaries of $B$ are closed. Thus, after a cluster $u$ has been opened for $p_0$ (by Line 7), all subsequent points in $I$ are put in the same cluster $u$. Note that the condition in Line 5 prevents points from the neighboring windows from joining $u$ and making crossing clusters. So, $u$ is the only cluster in $B$, and hence, $\mu(B) = 1$.

- SUBCASE 1.2: $\langle b_1, b_2\rangle = \langle 1, 0\rangle$. When $p_0$ arrives, a new cluster $u$ is opened, since $p_0$ is in $c_2$, the right border is closed, and $w$ contains $< 1$ cluster at the time so that the condition in line 5 fails. Again, all subsequent points in $I$ are put in the same cluster, and points from the neighboring windows cannot join $u$ and make crossing clusters. Hence, $\mu(B) = 1$.

- SUBCASE 1.3: $\langle b_1, b_2\rangle = \langle 0, 1\rangle$. We show that $\mu(B) < 2$. Suppose by contradiction that $\mu(B) = 2$. By Observation 2.10, $I$ cannot contain two clusters entirely. Therefore, the only way to get $\mu(B) = 2$ is that $I$ intersects three clusters $\langle u_1, u_2, u_3\rangle$ (from left to right, as always), where $u_1$ and $u_3$ are crossing clusters, and $u_2$ is entirely contained in $I$ (see Fig. 2.4). By a similar argument as in the proof of Observation 2.2(iii), $u_2$ is opened after $u_1$ and $u_3$ have become crossing clusters. Let $p_1$ be the first point of $u_1$ in $w$, and $p_2$ be the first point of $u_1$ in the neighboring window. We have two scenarios:

  - SUBSUBCASE 1.3.1: $p_1 \ll p_2$. In this case, cluster $u_1$ is opened for $p_1$. But $p_2$ cannot be put in $u_1$, because upon arrival of $p_2$, $w$ contains $< 2$ clusters, and thus, the condition in line 5 does not hold.

- Subsubcase 1.3.2: $p_2 \ll p_1$. Here, cluster $u_1$ is opened for $p_2$. But $p_1$ cannot be put in $u_1$, because upon arrival of $p_1$, $w$ contains $< 1$ cluster, and hence, the condition in line 5 does not hold.

Both scenarios lead to contradiction. Therefore, $\mu(B) \leq 3/2$.

- Subcase 1.4: $\langle b_1, b_2 \rangle = \langle 1, 1 \rangle$. Here, Lemma 2.11 implies that $\mu(B) \leq 2$.

Since each of the four subcases occurs with probability 1/4, we conclude that the expected value of $\mu(B)$ in Case 1 is at most $\frac{1}{4}(1 + 1 + \frac{3}{2} + 2) = \frac{11}{8}$.

Case 2: $B$ is split between two neighboring windows. Let $b$ be the single border indicator inside $B$. Let $\mu_0(B)$ and $\mu_1(B)$ represent the value of $\mu(B)$ for the case that $b$ is set to 0 and 1, respectively. It is clear by Lemma 2.11 that $\mu_1(B) \leq 2$. We rule out two possibilities:

- Subcase 2.1: $\mu_0(B) = 3$. Since $I$ cannot contain both a whole cluster and a crossing cluster by Observation 2.3(i), the only possible scenario is that $c_1$ intersects two clusters $\langle u_1, u_2 \rangle$, and $c_2$ intersects two clusters $\langle u_3, u_4 \rangle$, where $u_1$ and $u_4$ are crossing clusters, and $u_2$ and $u_3$ are whole clusters. Let $p_1$ be the first point in $u_2$ and $p_2$ be the first point in $u_3$. Suppose w.l.o.g. that $p_1 \ll p_2$. By Observation 2.2(iii), $p_1$ arrives after $u_1$ has been opened, and $p_2$ arrives after $u_4$ has been opened. But when $p_2$ arrives, the window containing it contains one cluster, $u_4$, and the neighboring window contains two clusters $u_1$ and $u_2$. Therefore, $p_2$ would be assigned to $u_2$ by line 5 instead: a contradiction.

- Subcase 2.2: $\mu_0(B) = 5/2$ and $\mu_1(B) = 2$. Suppose that $\mu_1(B) = 2$. Then $I$ intersects three clusters $\langle u_1, u_2, u_3 \rangle$, where $u_1$ and $u_3$ are crossing clusters, and $u_2$ is completely contained in $I$. Let $t$ be the time at which $u_1$ becomes a crossing cluster, and let $\sigma(t)$ be the subset of input points coming up to time $t$. By a similar argument as in the proof of Observation 2.2(iii), any point in $I \cap c_1$ not contained in $u_1$ arrives after time $t$. Therefore, upon receiving the input sequence $\sigma(t)$, $u_1$ becomes a crossing cluster no matter whether the border between $c_1$ and $c_2$ is open or closed. Using the same argument we conclude that $u_3$ becomes a crossing cluster regardless of the value of $b$. Now consider the case where $b = 0$. Since both $u_1$ and $u_3$ remain crossing clusters, $\mu_0(B)$ must be an integer (1, 2, or 3) and cannot equal 5/2.

Ruling out these two subcases, we have $\mu_0(B) + \mu_1(B) \leq 4$ in all remaining subcases, and therefore, the expected value of $\mu(B)$ in this case is at most 2.

Since each of Cases 1 and 2 occurs with probability 1/2, we conclude that $\rho(2) \leq \frac{1}{2}(\frac{11}{8}) + \frac{1}{2}(2) = \frac{27}{16}$. This bound is tight: to see this just consider the block $B = [2, 4)$, and the sequence of 8 points $\langle 4.5, 3.5, 5.5, 2.5, 1.5, 3.2, 2.7, 0.5\rangle$. If $B$ falls in one window, then the value of $\mu(B)$ in Subcases 1.1 to 1.4 is 1, 1, 3/2 and 2 respectively. If $B$ split between two windows, then the value of $\mu(B)$ is 2, regardless of the value of the border indicator inside $B$. Therefore, $E[\mu(B)] = \frac{1}{2}[\frac{1}{4}(1 + 1 + \frac{3}{2} + 2)] + \frac{1}{2}(2) = \frac{27}{16}$. □

**Theorem 2.13** *$\rho(3) \leq 17/8$.*

*Proof.* Consider a block $B$ of size 3, consisting of cells $\langle c_1, c_2, c_3\rangle$, and let $b$ be the single border indicator strictly inside $B$. We assume w.l.o.g. that $c_1$ and $c_2$ fall in the same window (the other scenario is symmetric). We consider two cases.

- Case 1: $b = 0$. In this case, $\mu(B) \leq 9/2$ by the same argument used in the proof of Theorem 2.7 in Section 2.3.

- Case 2: $b = 1$. Here, Lemma 2.11 implies that $\mu(B) \leq 4$.

Each of Cases 1 and 2 occurs with probability 1/2, therefore $\rho(3) \leq \frac{1}{2}(4 + \frac{9}{2})/2 = 17/8$. □

**Theorem 2.14** *$\rho(4) \leq 53/24$.*

*Proof.* Consider a block $B$ of size 4. We consider two easy cases.

- Case 1: $B$ falls completely in two windows. Let $b$ be the single border indicator strictly inside $B$. Now, if $b = 1$, $\mu(B) \leq 6$ by Lemma 2.11, otherwise, $\mu(B) \leq 7$ by Lemma 2.5. Therefore, the expected cost in this case is at most $\frac{1}{2}(6 + 7) = \frac{13}{2}$.

- Case 2: $B$ is split between three consecutive windows. Let $\langle b_1, b_2\rangle$ be the two border indicators inside $B$. For the subcase where $\langle b_1, b_2\rangle = \langle 1, 1\rangle$ the cost is at most 6 by Lemma 2.11, and for the remaining 3 subcases, the cost of $B$ is at most 7 by Lemma 2.5. Thus, the expected cost in this case is at most $\frac{1}{4}(6) + \frac{3}{4}(7) = \frac{27}{4}$.

Since each of Cases 1 and 2 occurs with probability exactly 1/2, we conclude that $\rho(4) \leq \frac{1}{2}(\frac{13}{2} + \frac{27}{4})/3 = \frac{53}{24}$. □

**Theorem 2.15** $\rho(k) \leq (2k-1)/(k-1)$ *for all* $k \geq 5$.

*Proof.* This is a direct implication of Lemma 2.5. □

## The Combined Algorithm

We are now ready to combine the RandBorder algorithm with the simple Grid algorithm (Algorithm 2) to obtain a randomized online algorithm with competitive ratio strictly less than 2.

**Algorithm 7 (Combined)** *With probability* 8/15 *run* RandBorder*, and with probability* 7/15 *run* Grid*.*

**Theorem 2.16** *The competitive ratio of* Combined *is at most* 11/6 *against oblivious adversaries.*

*Proof.* The competitive ratios of RandBorder and Grid within blocks of size 2 are 27/16 and 2, respectively. Therefore, the expected competitive ratio of the Combined algorithm is $\frac{8}{15}(\frac{27}{16}) + \frac{7}{15}(2) = \frac{11}{6}$ within a block of size 2. For larger block sizes, the expected competitive ratio of Combined is always at most 11/6, as shown in Table 2.2. By summing over all blocks and exploiting the additivity of our cost function $\mu(\cdot)$, we see that the expected total cost of the solution produced by Combined is at most 11/6 times the size of $\texttt{opt}(\sigma)$ for every input sequence $\sigma$. □

Table 2.2: Upper bounds on the competitive ratio of Grid, RandBorder, and Combined within a block.

| **Block Size** | **2** | **3** | **4** | $k \geq \mathbf{5}$ |
|---|---|---|---|---|
| Grid | 2 | 3/2 | 4/3 | $k/(k-1)$ |
| RandBorder | 27/16 | 17/8 | 53/24 | $(2k-1)/(k-1)$ |
| Combined | 11/6 | 11/6 | 9/5 | $(23k-8)/(15k-15)$ |

## 2.5 Beyond One Dimension

In the two-dimensional $L_\infty$-metric case, we want to partition the given point set into subsets, each of $L_\infty$-diameter at most 1 (i.e., each enclosable by a unit square), so as to minimize the number of subsets used. (See Fig. 2.5.)

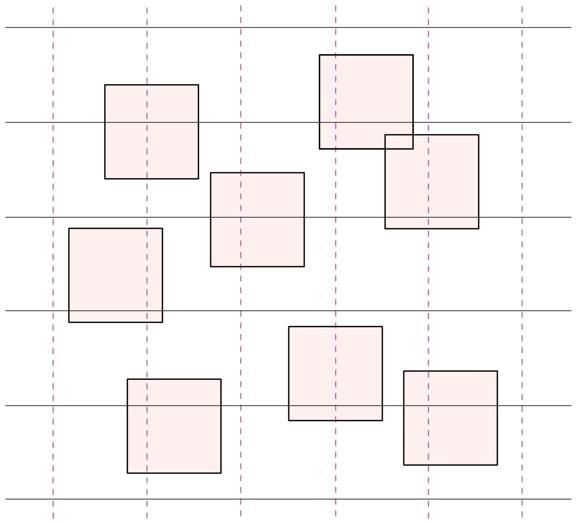

Figure 2.5: Unit clustering in the $L_\infty$ plane.

All the naïve algorithms mentioned in Section 2.2, when extended to two dimensions, provide 4-competitive solutions to the optimal solution. Theorem 2.1 can be generalized to a deterministic lower bound of 4 on the competitive ratio for the unit covering problem. We show how to extend Theorem 2.16 to obtain a competitive ratio strictly less than 4 for unit clustering.

**Theorem 2.17** *There is a $11/3$-competitive algorithm for the online unit clustering problem in the $L_\infty$ plane.*

*Proof.* Our online algorithm is simple: just use Combined to find a unit clustering $C_i$ for the points inside each horizontal strip $i \le y < i+1$. (Computing each $C_i$ is indeed a one-dimensional problem.)

Let $\sigma$ be the input sequence. We denote by $\sigma_i$ the set of points from $\sigma$ that lie in the strip $i \le y < i+1$. Let $Z_i$ be an optimal unit covering for $\sigma_i$. Let $O$ be an optimal unit covering for $\sigma$, and $O_i$ be the set of unit squares in $O$ that intersect the grid line $y = i$. Since all squares in $O_i$ lie in the strip $i-1 \le y < i+1$, we have $|Z_i| \le |O_{i-1}| + |O_i|$. Therefore $\sum_i |Z_i| \le 2|O|$, so $\sum_i |C_i| \le \frac{15}{8} \sum_i |Z_i| \le \frac{15}{4}|O|$. $\square$

The above theorem can be easily extended to dimension $d > 2$, with ratio $(\frac{11}{12}) \cdot 2^d$.

## 2.6 Conclusions

We have shown that determining the best competitive ratio for the online unit clustering problem is nontrivial even in the simplest one-dimensional case. In particular, we have obtained an online algorithm for this one-dimensional problem that achieves a competitive ratio of 11/6 using randomization.

An intriguing possibility that we have not ruled out is whether a nontrivial result can be obtained without randomization at all. After the appearance of the conference versions of this work, Epstein and van Stee [41] succeeded to obtain a deterministic algorithm for the one-dimensional unit clustering problem with a competitive ratio of 7/4. It is still open what is the best competitive ratio for this problem in one dimension. We will provide some complementary lower bounds in the next section.

In a recent work, Epstein, Levin and van Stee [39] have investigated several variations of the unit clustering problem. Examples of these variants include *clustering with temporary requests* in which request points are not permanent, but arrive and leave over time, and *max clustering* in which each point has a weight and the cost of a cluster is the maximum weight of any point assigned to it, with the objective of minimizing the total cost of the clusters. Epstein *et al.* [39] have shown that our simple GRID algorithm yields a competitive ratio of 2 for the above two variants, and that the 2 bound is the best possible.

# Chapter 3

# Online Coloring Co-interval Graphs

In this chapter, we study a problem closely related to online unit clustering, namely, the problem of online coloring co-interval graphs. In this problem, a set of intervals on the real line is presented to the algorithm one at a time, and upon receiving each interval $I$, the algorithm must assign $I$ a color different from the colors of all previously presented intervals not intersecting $I$. The objective is to use as few colors as possible. It is known that the competitive ratio of the simple First-Fit algorithm on the class of co-interval graphs is at most 2. We show that for the class of unit co-interval graphs, where all intervals have equal length, the 2-bound on the competitive ratio of First-Fit is tight. On the other hand, we show that no deterministic online algorithm for coloring unit co-interval graphs can be better than 3/2-competitive. We then study the effect of randomization on our problem, and show a lower bound of 4/3 on the competitive ratio of any randomized algorithm for the unit co-interval coloring problem. We also prove that for the class of general co-interval graphs no randomized algorithm has competitive ratio better than 3/2.

## 3.1 Introduction

A variety of optimization problems in scheduling, partitioning and resource allocation can be modeled as graph coloring problems. The graph coloring problem

An extended abstract of this chapter has been published as: H. Zarrabi-Zadeh. Online coloring co-interval graphs. In *Proc. 12th International CSI Computer Conference*, pages 1328–1332, 2007.

involves assigning colors to the vertices of a graph so that adjacent vertices are assigned different colors, with the objective of minimizing the number of colors used. It is well-known that the problem is NP-hard, even for graphs with a fixed chromatic number $k \geq 3$ [69]. Furthermore, it is intractable to approximate the problem to within a factor of $n^c$ for some constant $c > 0$ unless P = NP [76].

In the *online* graph coloring problem, vertices of the graph are presented one at a time. When a vertex is presented, all the edges from it to vertices presented earlier are given. An online coloring algorithm assigns a color to the current vertex before the next vertex is presented, and once a color is assigned to a vertex, the algorithm is not allowed to change its color at a future time.

The online graph coloring problem has been widely studied. Lovász, Saks, and Trotter [75] gave an online algorithm that achieves a competitive ratio of $O(n/\log^* n)$ on all graphs. Vishwanathan [94] gave a randomized algorithm which, as improved in [55], obtains a competitive ratio of $O(n/\log n)$ against an oblivious adversary. A close lower bound of $\Omega(n/\log^2 n)$ is proved by Halldórsson and Szegedy [56] for both deterministic and randomized algorithms.

Due to the inherent complexity of the online coloring problem on general graphs, many researchers have focused their study on special classes of graphs [33, 40, 68, 71, 78]. For example, Kierstead and Trotter [71] constructed an optimum online algorithm for coloring interval graphs. Their algorithm uses at most $3\omega - 2$ colors, where $\omega$ is the maximum clique size of the graph. Since interval graphs are perfect, their chromatic numbers are equal to their clique numbers (see [50] for the definition of graph theory terms used throughout this chapter). This means that the algorithm of Kierstead and Trotter is 3-competitive for interval graphs. A matching randomized lower bound of 3 is presented in [73].

Another line of research has been on analyzing the performance of the simple First-Fit algorithm, i.e., the algorithm that simply assigns the smallest available color to each new vertex. For example, it is known that for the class of interval graphs the competitive ratio of First-Fit is at least 4.4 [29] and at most 10 [87].

In this chapter, we study the online coloring problem for the class of co-interval graphs. Gyárfás and Lehel [54] have shown that First-Fit uses at most $2\omega - 1$ colors on any co-chordal graph. Since interval graphs are chordal [50], this bound also applies to co-interval graphs. Kierstead and Qin [70] have shown that no online deterministic algorithm can beat this 2-bound on co-interval graphs. We present the first lower bound, up to our best knowledge, for randomized online coloring of co-interval graphs. We show that any randomized algorithm for the problem has

Table 3.1: Lower and upper bounds on the competitive ratio of online algorithms for coloring co-interval graphs. Entries shown in bold are proved in this chapter.

| | Deterministic | | Randomized | |
|---|---|---|---|---|
| | LB | UB | LB | UB |
| Coloring co-interval graphs | 2 [70] | 2 [54] | **3/2** | 2 [54] |
| Coloring unit co-interval graphs | **3/2** | 2 [54] | **4/3** | 11/6 [Chap. 2] |

an expected competitive ratio at least 3/2.

For the class of unit co-interval graphs, where all intervals have equal (unit) length, we show that the competitive ratio of First-Fit is still 2. We then show that no deterministic algorithm for this problem can be better than 3/2-competitive. We also prove a lower bound of 4/3 on the competitive ratio of any randomized algorithm for coloring unit co-interval graphs. Both these lower bounds also apply to the online unit clustering problem due to the close connection between these two problems which will be addressed in Section 3.2. A summary of the lower and upper bounds provided in this chapter and the previous works is presented in Table 3.1.

## 3.2 Coloring Unit Co-interval Graphs

All comparability and co-comparability graphs are perfect, so their chromatic numbers are equal to their clique numbers. Gyárfás and Lehel [54] have shown that for any deterministic online coloring algorithm $\mathcal{A}$ and any positive integer $k$, there is a tree $T$ such that $\mathcal{A}$ uses at least $k$ colors on $T$. Since trees are special cases of comparability graphs with clique number two, it means that in general the number of colors used by First-Fit on comparability graphs cannot be bounded in terms of their clique number.

On the other hand, for the class of co-interval graphs, which are a subclass of comparability graphs, we can obtain better bounds on the competitive ratio of the First-Fit algorithm. The result of [54] shows that First-Fit needs at most $2\omega - 1$ colors on the class of co-chordal graphs, where $\omega$ is the clique number of the graph. Since co-interval graphs are a subclass of co-chordal graphs, this bound also applies to (unit) co-interval graphs.

In the following, we show that the above upper bound on the number of colors used by First-Fit is indeed tight, even on the class of unit co-interval graphs.

**Theorem 3.1** FIRST-FIT *has a competitive ratio of 2 on the class of unit co-interval graphs.*

*Proof.* This is analogous to what is proved in Section 2.2. The adversary provides a sequence of $2k-2$ intervals of the form $[i, i+1]$ $(2 \leq i < 2k)$, followed by $k$ unit intervals of the form $[2i - \frac{1}{2}, 2i + \frac{1}{2}]$ $(1 \leq i \leq k)$. FIRST-FIT uses $2k-1$ colors to color this sequence, while the co-interval graph represented by this set of intervals has chromatic number $k$. □

### 3.2.1 Lower Bounds

As mentioned in the previous section, FIRST-FIT has a competitive ratio of 2 on the class of unit co-interval graphs. An immediate question is whether one can obtain a better deterministic algorithms for this problem. The following theorem shows that no such algorithm can be better than 3/2-competitive.

**Theorem 3.2** *There is a lower bound of* $3/2$ *on the competitive ratio of any deterministic online algorithm for coloring unit co-interval graphs.*

*Proof.* Let $I_i = [i, i+1]$, for $i \in \{1, \ldots, 4\}$. Consider two input sequences $\sigma_1 = \langle I_2, I_3 \rangle$ and $\sigma_2 = \langle I_2, I_3, I_1, I_4 \rangle$. The adversary chooses one of the two sequences as input. Let $\mathcal{A}$ be any deterministic algorithm for the problem. No matter which sequence is chosen by the adversary, $\mathcal{A}$ receives $I_2$ and $I_3$ as the first two intervals. If $\mathcal{A}$ decides to color $I_2$ and $I_3$ with two different colors, then $\mathcal{A}$ is 2-competitive on $\sigma_1$. If $\mathcal{A}$ decides to color $I_2$ and $I_3$ with the same color, then it needs two more colors to color $I_1$ and $I_4$. It means that $\mathcal{A}$ needs three colors on $\sigma_2$, while the chromatic number of $\sigma_2$ is 2. Thus, $\mathcal{A}$ is at least 3/2-competitive on these two input sequences. □

A randomized lower bound for online coloring unit co-interval graphs is provided in the following theorem.

**Theorem 3.3** *Any randomized algorithm for online coloring unit co-interval graphs is at least* $4/3$*-competitive.*

*Proof.* Let $\mathcal{A}$ be an arbitrary randomized algorithm for the problem, and let $\rho_{\mathcal{A}}(\sigma)$ be the expected competitive ratio of $\mathcal{A}$ on an input sequence $\sigma$. As in the previous theorem, define $I_i = [i, i+1]$ for $i \in \{1, \ldots, 4\}$, and consider two input sequences $\sigma_1 = \langle I_2, I_3 \rangle$ and $\sigma_2 = \langle I_2, I_3, I_1, I_4 \rangle$. No matter which of these two sequences is chosen by the adversary, $\mathcal{A}$ receives $I_2$ and $I_3$ as the first two intervals. Let $E$ be the event that $\mathcal{A}$ assigns two different colors to $I_2$ and $I_3$. If $E$ occurs, then $\mathcal{A}$ uses 2 colors on each of the input sequences $\sigma_1$ and $\sigma_2$. If $E$ doesn't occur, then $\mathcal{A}$ uses one color on $\sigma_1$ and 3 colors on $\sigma_2$. Let $p = \Pr[E]$. Then it is clear that

$$\rho_{\mathcal{A}}(\sigma_1) = 2p + (1-p) = p + 1,$$

and

$$\rho_{\mathcal{A}}(\sigma_2) = \frac{1}{2}(2p + 3(1-p)) = (3-p)/2 \ .$$

If $p > 1/3$ then $\rho_{\mathcal{A}}(\sigma_1) = p + 1 > 4/3$, and hence, $\mathcal{A}$ is not 4/3-competitive on $\sigma_1$. Thus we can assume that $p \leq 1/3$. But then $\mathcal{A}$ cannot be better than 4/3-competitive, because $\rho_{\mathcal{A}}(\sigma_2) = (3-p)/2 \geq 4/3$. □

### 3.2.2 Connection to Unit Clustering

The problem of online coloring unit co-interval graphs has a close connection to the online unit clustering problem in one dimension, as stated in the following observation.

**Observation 3.4** *The unit clustering problem in one dimension is equivalent to the problem of coloring unit co-interval graphs.*

*Proof.* It is clear from elementary graph theory that coloring co-interval graphs is equivalent to clique partitioning interval graphs, i.e., partitioning a set of intervals into minimum number of subsets such that intervals in each subset have a common intersection point. In other words, given a set $\mathcal{I}$ of unit intervals, we want to find a minimum cardinality set of points $P$ such that each interval in $\mathcal{I}$ is "pierced" by at least one of the points in $P$.

Now, we define the following "point-interval" duality: For a given point $p$, we define $p^*$ to be the unit interval centered at $p$. For a unit interval $I$, we define $I^*$ to be its center. It is easy to observe that

$$p \in I \quad \Leftrightarrow \quad I^* \in p^* \ .$$

Given an instance of the unit interval piercing problem, we can map each unit interval to its dual point, and hence, come up with the following equivalent "dual" problem: given a set $P$ of points on the line, find a minimum cardinality set $\mathcal{I}$ of unit intervals such that each point in $P$ is contained in at least one interval of $\mathcal{I}$. This is equivalent to the online unit clustering problem in one dimension. $\square$

As a direct corollary of Observation 3.4, the lower bounds proved in Theorems 3.2 and 3.3 can be applied to the online unit clustering problem as well.

**Corollary 3.5** *There is a lower bound of* $3/2$ *(respectively,* $4/3$*) on the competitive ratio of any deterministic (respectively, randomized) algorithm for the online unit clustering problem in one dimension.*

## 3.3 Coloring Co-interval Graphs

In this section, we consider the class of general co-interval graphs. Obviously, the 4/3 lower bound proved in the previous section also applies to the general co-interval graphs. In this section, we obtain a stronger result by proving that no randomized algorithm for coloring co-interval graphs can be better than 3/2-competitive.

**Theorem 3.6** *There is a lower bound of* $3/2$ *on the competitive ratio of any randomized algorithm for online coloring co-interval graphs.*

*Proof.* By Yao's minimax principle [96], the expected competitive ratio of the optimal deterministic algorithm for an arbitrarily chosen input distribution is a lower bound on the expected competitive ratio of every randomized algorithm. Thus, to show the lower bound, we only need to provide a probability distribution on a set of input sequences such that the expected competitive ratio of any deterministic online algorithm on that distribution is at least 3/2.

Let $k \geq 1$ be a fixed integer. For $1 \leq i \leq k$, we define three types of intervals $a_i$, $b_i$, and $c_i$ on the real line as follows:

$$\begin{aligned} a_i &= [3i-3,\ 3i-2], \\ b_i &= [3i+1,\ 3i+2], \\ c_i &= [3i+2,\ +\infty]. \end{aligned}$$

Consider $k$ blocks of intervals $B_1, \ldots, B_k$, where each block is a sequence of two or three intervals defined as follows:

$$B_i = \begin{cases} \langle b_1, c_1 \rangle & i = 1, \\ \langle a_i, b_i, c_i \rangle & 2 \leq i < k, \\ \langle a_k, b_k \rangle & i = k. \end{cases}$$

We construct a set $\mathcal{I}$ of $k$ input sequences $\sigma_1$ to $\sigma_k$, where each $\sigma_i$ is obtained by concatenating the first $i$ blocks $B_1$ to $B_i$, in order from left to right. It is easy to observe that the co-interval graph represented by each $\sigma_i$ has chromatic number equal to $i$.

Now, we define a probability distribution $\mathcal{P}$ over the input set $\mathcal{I}$. Let $p_i$ be the probability that $\sigma_i$ is chosen as input. We define

$$p_i = \begin{cases} \frac{2^{k-1}}{2^k-1} \cdot \frac{i}{2^i} & 1 \leq i \leq k-1, \\ \frac{k}{2^k-1} & i = k. \end{cases}$$

Note that our probability distribution is properly defined, as $\sum_{i=1}^{k} p_i = 1$.

First-Fit uses exactly $2i-1$ colors to color each $\sigma_i$ (it opens a new color on every $b_j$ $(1 \leq j \leq i)$ and a new color on every $a_j$ $(1 < j \leq i)$). Thus, the expected competitive ratio of First-Fit on the input distribution $\mathcal{P}$ is

$$\rho_{\mathrm{FF}} = \sum_{i=1}^{k} p_i \left( \frac{2i-1}{i} \right) = \frac{3}{2} - \frac{1}{2(2^k-1)} .$$

Our aim is to show that the expected competitive ratio of any other deterministic online algorithm is at least $\rho_{\mathrm{FF}}$ on the input distribution $\mathcal{P}$.

Let $\mathcal{A}$ be an arbitrary deterministic online algorithm. Since all sequences in $\mathcal{I}$ are prefixes of $\sigma_k$, $\mathcal{A}$ makes the same decision on any specific interval in all input sequences. Let $d_i$ be the decision made by $\mathcal{A}$ upon receiving the interval $c_i$ $(1 \leq i < k)$. We set $d_i = 0$, if $\mathcal{A}$ colors $c_i$ with the same color assigned to $b_i$, and set $d_i = 1$, otherwise. Note that $\mathcal{A}$ can always color $c_i$ with the same color assigned to its preceding $b_i$. This is because $b_i$ has no intersection with the intervals of types $a$ and $b$ presented before it, and has intersection with all intervals of type $c$ presented thus far.

We call the sequence of decisions $\langle d_1, \ldots, d_{k-1} \rangle$ the *characteristic sequence* of $\mathcal{A}$. The claim is that knowing the characteristic sequence of an algorithm $\mathcal{A}$, we can determine the minimum number of colors that $\mathcal{A}$ needs on each input sequence, regardless of decisions it makes on intervals of types $a$ and $b$.

Suppose that a sequence $\delta = \langle d_1, \ldots, d_{k-1} \rangle$ of $k-1$ bits is given, where each $d_i \in \{0, 1\}$. We define a deterministic online algorithm $\mathrm{FF}(\delta)$ as follows. Upon receiving an interval of type $c$, say $c_i$, the algorithm checks the corresponding bit $d_i$ in the bit sequence $\delta$ and opens a new color for the interval or colors it using the same color assigned to its preceding $b_i$, depending on whether $d_i$ is 1 or 0, respectively. On the other hand, when $\mathrm{FF}(\delta)$ receives an interval of type $a$ or $b$, it just behaves like the First-Fit algorithm, i.e., assigns the first available color to the given interval, or opens a new color for the interval provided no previous color is available.

**Claim 1** *Among all deterministic online algorithms with characteristic sequence $\delta$, $\mathrm{FF}(\delta)$ uses the minimum number of colors on every input sequence in $\mathcal{I}$.*

**Claim 2** *For any arbitrary sequence $\delta$ of $k-1$ bits, the expected competitive ratio of $\mathrm{FF}(\delta)$ on the input distribution $\mathcal{P}$ is equal to $\rho_{\mathrm{FF}}$.*

Claim 1 is easy to prove. Here, we provide a proof for Claim 2. Let $m$ be the number of bits which are equal to 1 in the given sequence $\delta$. We prove the claim by induction on $m$. The base case $m = 0$ is trivial, because in this case all entries in $\delta$ are 0, and hence $\mathrm{FF}(\delta)$ is the same as the First-Fit algorithm.

Now, suppose that the claim is true for all sequences with less than $m$ bits equal to 1, and consider a sequence $\delta$ in which $m$ bits are 1. Let $j$ $(1 \le j \le k-1)$ be the position of the last 1-bit in $\delta$. Changing the $j$-th bit in $\delta$ from 1 to 0, we obtain a new sequence which we call $\bar{\delta}$. Since the number of 1-bits in $\bar{\delta}$ is $m-1$, by the induction hypothesis

$$\rho_{\mathrm{FF}(\bar{\delta})} = \rho_{\mathrm{FF}},$$

where $\rho_{\mathrm{FF}(\bar{\delta})}$ is the expected competitive ratio of $\mathrm{FF}(\bar{\delta})$. Since the first $j-1$ bits of $\delta$ and $\bar{\delta}$ are the same, the competitive ratio of $\mathrm{FF}(\delta)$ is equal to that of $\mathrm{FF}(\bar{\delta})$ on all input sequences smaller than $\sigma_j$ in $\mathcal{I}$. For $\sigma_j$, $\mathrm{FF}(\delta)$ uses one color more than $\mathrm{FF}(\bar{\delta})$, because $\mathrm{FF}(\delta)$ assigns two different colors to two intervals $b_j$ and $c_j$, while $\mathrm{FF}(\bar{\delta})$ colors these two intervals with the same color. On the other hand, $\mathrm{FF}(\delta)$ uses one color less than $\mathrm{FF}(\bar{\delta})$ on any input sequence $\sigma_i$ for all $j < i \le k$. This is because $\mathrm{FF}(\bar{\delta})$ needs to open two new colors for $a_{j+1}$ and $b_{j+1}$, while $\mathrm{FF}(\delta)$ does not open any new color for these two intervals: it colors $a_{j+1}$ with the same color assigned to $b_j$ and colors $b_{j+1}$ with the same color assigned to $c_j$. Thus,

$$\rho_{\mathrm{FF}(\delta)} \;=\; \rho_{\mathrm{FF}(\bar{\delta})} + p_j\left(\frac{1}{j}\right) - \sum_{i=j+1}^{k} p_i\left(\frac{1}{i}\right).$$

But, we know that

$$\begin{aligned}\sum_{i=j+1}^{k} p_i\left(\frac{1}{i}\right) &= \sum_{i=j+1}^{k-1} \frac{2^{k-1}}{2^k-1}\cdot\frac{1}{2^i} + \frac{1}{2^k-1}\\ &= \frac{2^{k-1}}{2^k-1}\left(\frac{1}{2^j}-\frac{1}{2^{k-1}}\right)+\frac{1}{2^k-1}\\ &= \frac{2^{k-1}}{2^k-1}\left(\frac{1}{2^j}\right) = p_j\left(\frac{1}{j}\right).\end{aligned}$$

Therefore

$$\rho_{\mathrm{FF}(\delta)} = \rho_{\mathrm{FF}(\bar{\delta})} = \rho_{\mathrm{FF}},$$

and the proof of Claim 2 is complete.

Claims 1 and 2 together show that the expected competitive ratio of any deterministic online algorithm on the input distribution $\mathcal{P}$ is at least $\rho_{\mathrm{FF}}$. If $k$ is chosen arbitrarily large, $\rho_{\mathrm{FF}}$ tends to $3/2$ and the theorem statement follows. □

## 3.4 Conclusions

In this chapter, we proved several lower bounds on the competitive ratio of deterministic and randomized algorithms for online coloring co-interval graphs. Our work raises many open questions concerning the gap between the upper and lower bounds presented in Table 3.1. Very recently, Epstein and van Stee have succeeded to improve the deterministic and randomized lower bounds for coloring unit co-interval graphs to 8/5 and 3/2, respectively [41].

For the class of general co-interval graphs, it is known that no deterministic online coloring algorithm can be better than 2-competitive [70], but we do not see any simple argument that achieves a similar randomized lower bound. An interesting question that remains open is whether one can obtain an online coloring algorithm for general co-interval graphs with a competitive ratio strictly less than 2 using randomization.

# Chapter 4

# Streaming Algorithms for Coresets

In this chapter, we switch to the data stream model of computation and present a new streaming algorithm for maintaining an $\varepsilon$-kernel of a point set in $\mathbb{R}^d$ using $O((1/\varepsilon^{(d-1)/2})\log(1/\varepsilon))$ space. The space used by our algorithm is optimal up to a small logarithmic factor. Our algorithm immediately improves the space complexity of the best previous streaming algorithms for a number of fundamental geometric optimization problems in fixed dimensions, including width, minimum enclosing cylinder, minimum-width enclosing annulus, minimum-width enclosing cylindrical shell, etc.

## 4.1 Introduction

The coreset framework has recently emerged as a powerful tool for approximating various measures of a geometric data set. Agarwal *et al.* [4] developed a generic method for computing coresets for various optimization problems by introducing the notion of $\varepsilon$-kernel. Roughly speaking, a subset $Q \subseteq P$ is called an *$\varepsilon$-kernel* of $P$ if for every slab $S$ containing $Q$, the $(1+\varepsilon)$-expansion of $S$ contains $P$. The technique of Agarwal *et al.* yields approximation algorithms for a wide range of shape-fitting problems, as noted in Section 1.4.

An extended abstract of this chapter has been published as: H. Zarrabi-Zadeh. An almost space-optimal streaming algorithm for coresets in fixed dimensions. In *Proc. 16th European Symposium on Algorithms*, LNCS 5193, pages 817-829, 2008.

Table 4.1: Space complexity of various streaming algorithms for maintaining $\varepsilon$-kernels in $\mathbb{R}^d$.

| ALGORITHM | SPACE BOUND | REF |
|---|---|---|
| Agarwal, Har-Peled, and Varadarajan '04 | $O((1/\varepsilon^{\frac{d-1}{2}}) \log^d n)$ | [4] |
| Chan '06 | $O((1/\varepsilon^{d-(3/2)}) \log^d(1/\varepsilon))$ | [21] |
| Agarwal and Yu '07 | $O(1/\varepsilon^{d-(3/2)})$ | [12] |
| This work | $O((1/\varepsilon^{\frac{d-1}{2}}) \log(1/\varepsilon))$ | Here |

In this chapter, we are interested in space-efficient streaming algorithms for maintaining $\varepsilon$-kernels in $\mathbb{R}^d$. Here, the input is given to the algorithm as a stream over time, and the algorithm has to process the input elements as they arrive in only one pass, using a limited amount of working storage. Using the general dynamization technique of Bentley and Saxe [14], Agarwal *et al.* [4] gave a streaming algorithm for maintaining an $\varepsilon$-kernel of a stream of points in $\mathbb{R}^d$ using $O((1/\varepsilon^{(d-1)/2}) \log^d n)$ space and $O(1/\varepsilon^{d-1})$ time per update, where $n$ is the number of points in the stream. Chan [21] succeeded to remove the dependency of the space bound to $n$ and provide the first constant-space streaming algorithm that uses only $O([(1/\varepsilon) \log(1/\varepsilon)]^{d-1})$ space and needs $O(1)$ amortized time for processing each new point. He also showed how the space bound can be improved to $O((1/\varepsilon^{d-(3/2)}) \log^d(1/\varepsilon))$ at the expense of increasing the update time to $O(1/\sqrt{\varepsilon})$. Later on, Agarwal and Yu [12] removed the extra logarithmic factors and slightly improved the space complexity to $O(1/\varepsilon^{d-(3/2)})$, with $O(\log(1/\varepsilon))$ update time per input point. Agarwal and Yu's algorithm is indeed space-optimal in two dimensions, but is still far from optimal in dimensions higher than two.

**Our Results.** Chan [21] left this question open whether the space bound for the problem of maintaining $\varepsilon$-kernels in $\mathbb{R}^d$ can be brought down to near $O(1/\varepsilon^{(d-1)/2})$. In this chapter, we answer Chan's question in the affirmative by providing a streaming algorithm that uses a near optimal space. More precisely, our algorithm maintains an $\varepsilon$-kernel in $\mathbb{R}^d$ using only $O((1/\varepsilon^{(d-1)/2}) \log(1/\varepsilon))$ space and $(1/\varepsilon)^{O(d)}$ update time per insertion. The space bound of our algorithm is optimal up to a logarithmic factor, as one can easily verify that any $\varepsilon$-kernel for sufficiently many points uniformly distributed on the surface of a $d$-dimensional hypersphere has size $\Omega(1/\varepsilon^{(d-1)/2})$ [5].

Our algorithm differs from its predecessors [12, 21] in that the high level structure of our algorithm is not dimensionality-reduction (which ultimately exploits efficient techniques in two dimensions), but rather a high-dimensional partition of the input stream into $d$-dimensional "fat substreams" for which $\varepsilon$-kernels can be maintained efficiently by a kind of bucketing scheme. Our algorithm can be viewed as a generalization of the algorithm proposed by Chan [21] in two dimensions, which also employs a version of the compression technique used in [12]. See Table 4.1 for a comparison between the space complexity of our algorithm and the previous ones.

Our algorithm immediately improves the space complexity of the best previous streaming algorithms for a wide range of geometric problems in fixed dimensions, including width, minimum-volume bounding box, minimum-radius enclosing cylinder, minimum-width enclosing cylindrical shell, etc. The improvement obtained by our algorithm is substantial when the problem's dimension is large. However, the algorithm improves several previous results in lower dimensions as well. For example, for the two-dimensional minimum-width enclosing annulus problem, combined with the lifting technique of Agarwal *et al.* [4], our algorithm requires only $O((1/\varepsilon)\log(1/\varepsilon))$ space, while the best previous space bound for this problem was $O(1/\varepsilon^{3/2})$. As a byproduct of our algorithm, we also show how to maintain an $\varepsilon$-kernel of a stream of points in $\mathbb{R}^d$ in an optimal time of $O(1)$ using $O((1/\varepsilon^{d-(3/2)})\log(1/\varepsilon))$ space, which improves the best previously known time-optimal algorithm by Chan [21] that requires $O([(1/\varepsilon)\log(1/\varepsilon)]^{d-1})$ space.

## 4.2 Preliminaries

We first introduce the notation used in this chapter. For a point set $P \subseteq \mathbb{R}^d$ and a direction $u \in \mathbb{S}^{d-1}$, the *directional width* of $P$ along $u$ is defined by $w(P,u) = \max_{p,q\in P} \langle p-q, u\rangle$, where $\langle\cdot,\cdot\rangle$ denotes the inner product function. A subset $Q \subseteq P$ is called an *$\varepsilon$-kernel* of $P$, if for all $u \in \mathbb{S}^{d-1}$,

$$w(Q,u) \geq (1-\varepsilon)w(P,u).$$

We will use the following result from Chan throughout this chapter:

**Theorem 4.1 (Chan [21])** *Given a set $P$ of $n$ points in $\mathbb{R}^d$, an $\varepsilon$-kernel of $P$ of size $O(1/\varepsilon^{(d-1)/2})$ can be computed in $O(n + 1/\varepsilon^{d-(3/2)})$ time for $d \geq 2$, or in $O((n + 1/\varepsilon^{d-2})\log(1/\varepsilon))$ time for $d \geq 3$.*

Let $S = \{p_1, \ldots, p_k\}$ be a set of $k$ points in $\mathbb{R}^d$ ($k \leq d+1$). We denote by $\mathcal{F}(S)$ the *flat* spanned by $S$, i.e., $\mathcal{F}(S) = \left\{\sum_{i=1}^{k} a_i p_i \mid a_1, \ldots, a_k \in \mathbb{R} \text{ and } \sum_{i=1}^{k} a_i = 1\right\}$. Given a point $p \in \mathbb{R}^d$ and a flat $\mathcal{F} \in \mathbb{R}^d$, the *orthogonal projection* of $p$ onto $\mathcal{F}$ is defined as $\text{proj}(\mathcal{F}, p) = \arg\min_{p' \in \mathcal{F}} \|pp'\|$. The Euclidean distance between $p$ and its projection onto $\mathcal{F}$ is denoted by $d_{\mathcal{F}}(p)$. Throughout this chapter, we simply use $d_S(p)$ instead of $d_{\mathcal{F}(S)}(p)$ to refer to the distance of $p$ to a flat defined by the point set $S$. If $S = \emptyset$, then $d_S(p) = 0$ by definition.

Let $X = \langle x_0, x_1, \ldots, x_k \rangle$ be a sequence of $k+1$ points in $\mathbb{R}^d$ ($k \leq d$). We say that a point $p \in \mathbb{R}^d$ is *$X$-respecting*, if for every $0 \leq i < k$,

$$d_{X_i}(p) \leq 2 \cdot d_{X_i}(x_{i+1}),$$

where $X_i = \{x_0, x_1, \ldots, x_i\}$. If $|X| \leq 1$, then every point is $X$-respecting by definition. The sequence $X$ is called *self-respecting*, if for every $1 \leq j \leq k$, $x_j$ is $\langle x_0, \ldots, x_{j-1} \rangle$-respecting. Moreover, a point set $P$ is called $X$-respecting, if for every point $p \in P$, $p$ is $X$-respecting.

## 4.3 Algorithm for Fat Substreams

In this section, we present a simple efficient algorithm for maintaining $\varepsilon$-kernels of fat substreams to be used as a subroutine of our main algorithm in Section 4.4. Let $\mathbb{B}$ be a hyperbox in $\mathbb{R}^d$. A point set $P \subseteq \mathbb{R}^d$ is called *fat with respect to* $\mathbb{B}$, if there exist a positive constant $\alpha \leq 1$ and two points $v$ and $v'$ so that $v + \alpha\mathbb{B} \subseteq \text{conv}(P) \subseteq v' + \mathbb{B}$. If $P$ is fat with respect to some hyperbox $\mathbb{B}$, then an $\varepsilon$-kernel of $P$ of size $O(1/\varepsilon^{(d-1)/2})$ can be computed efficiently using a simple grid-rounding method proposed in [21, 97] based on Dudley's construction [34]. Dudley's method actually works for the case where $\mathbb{B}$ is a hypercube. However, we can easily transform $\mathbb{B}$ to a hypercube by an affine transform $\tau$, and then compute an $\varepsilon$-kernel $Q$ of the set $\tau(P)$. The set $\tau^{-1}(Q)$ is then an $\varepsilon$-kernel of $P$, as proved in [4]. In the following, we show how this idea can be used for $X$-respecting substreams.

Let $X = \langle x_0, x_1, \ldots, x_d \rangle$ be a sequence of $d+1$ points in $\mathbb{R}^d$. For each $1 \leq i \leq d$, we denote by $\hat{x}_i$ the projection of $x_i$ onto $\mathcal{F}(X_{i-1})$, where $X_i = \{x_0, x_1, \ldots, x_i\}$. Let $w_i = \|\hat{x}_i x_i\|$ and $u_i = (1/w_i)\overrightarrow{\hat{x}_i x_i}$. We denote by $\mathcal{B}_X$ the $d$-dimensional box centered at $x_0$, whose $i$-th side has length $4w_i$ in direction $u_i$ ($1 \leq i \leq d$). The following lemma (which is analogous to what is proved in [13] for three dimensions) provides a connection between $X$-respecting and fat sets.

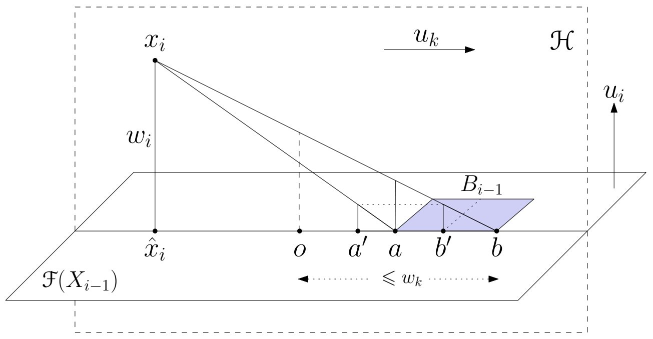

Figure 4.1: On Lemma 4.2.

**Lemma 4.2** *Let $X$ be a self-respecting sequence of $d+1$ points in $\mathbb{R}^d$. Given a point set $P \in \mathbb{R}^d$, if $P$ is $X$-respecting, then $P \cup X$ is fat with respect to $\mathcal{B}_X$.*

*Proof.* Obviously, $\mathcal{B}_X$ contains all the points of $P \cup X$. In the following, we show that $\text{conv}(X)$ (and therefore, $\text{conv}(P \cup X)$) contains a translated copy of $(1/4)^d \mathcal{B}_X$. Suppose by induction that $\text{conv}(X_{i-1})$ contains an $(i-1)$-dimensional box $B_{i-1}$, whose $j$-th side has length $w_j/4^{i-1}$ in direction $u_j$ $(1 \leq j < i)$. Now, consider an $i$-dimensional pyramid $\mathcal{P}_i$ obtained by connecting $x_i$ to all facets of $B_{i-1}$. Clearly, $\mathcal{P}_i \subseteq \text{conv}(X_i)$. We only need to show that $\mathcal{P}_i$ contains an $i$-dimensional box $B_i$, whose $j$-th side has length at least $w_j/4^i$ in direction $u_j$ $(1 \leq j \leq i)$.

Fix a $k$ $(1 \leq k < i)$, and consider the plane $\mathcal{H}$ through $x_i\hat{x}_i$ parallel to $u_k$ (see Figure 4.1). The projection of $B_{i-1}$ onto $\mathcal{H}$ is a line segment $ab$ of length $w_k/4^{i-1}$. Let $o$ be the orthogonal projection of $x_0$ onto $\mathcal{H}$. Then we have $\|oa\|, \|ob\| \leq w_k$ (because $B_{i-1} \subset \text{conv}(X_{i-1})$), and $\|o\hat{x}_i\| \leq \|x_0x_i\| \leq 2w_k$ (because $X$ is self-respecting).

Let $a'$ (respectively, $b'$) be a point on $\overleftrightarrow{ab}$ whose vertical distance (in direction $u_i$) from $x_ia$ (respectively, from $x_ib$) is equal to $w_i/4^i$. We have $\|aa'\|, \|bb'\| \leq 3w_k/4^i$, due to similarity of triangles, and because $\|a\hat{x}_i\|, \|b\hat{x}_i\| \leq 3w_k$. Let $s_k = \overline{ab} - (\overline{aa'} \cup \overline{bb'})$. If one of the two angles $\angle abx_i$ and $\angle bax_i$ is obtuse, then one of the segments $aa'$ and $bb'$ falls completely outside $ab$, and therefore $\|s_k\| \geq w_k/4^i$. If both $\angle abx_i$ and $\angle bax_i$ are at most $\pi/2$, then $\|aa'\| + \|bb'\| = \|ab\|/4 = w_k/4^i$, and therefore, $\|s_k\| = 3w_k/4^i \geq w_k/4^i$. Now, we cut that portion of $B_{i-1}$ whose projection onto $\mathcal{H}$ lies inside $s_k$, and repeat this procedure for every $1 \leq k < i$. The remaining box, $B'_{i-1}$, has length at least $w_k/4^i$ in each direction $u_k$. If we expand $B'_{i-1}$ by $w_i/4^i$ units in direction $u_i$, we obtain the desired $i$-dimensional box $B_i$ which completely remains inside $\mathcal{P}_i$. □

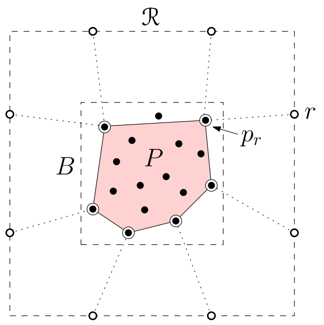

Figure 4.2: Constructing an $\varepsilon$-kernel of a fat point set.

Note that the fatness parameter $\alpha$ in Lemma 4.2 is exponential in $d$, but is a constant in any fixed dimension.

**Algorithm for Fat Sets.** Let $P$ be a point set in $\mathbb{R}^d$ which is fat with respect to some hyperbox $B$. By a translation, we may assume that $B$ is centered at origin. Moreover, we may assume that $B = [-1, 1]^d$ by an affine transform. According to Chan [21] and Yu *et al.* [97], one can easily compute an $\varepsilon$-kernel of $P$ using a simple grid method as follows: Let $\mathcal{R}$ be the set of points of a $\sqrt{\varepsilon}$-grid over the boundary of the cube $[-2, 2]^d$, and let $p_r$ denote the nearest neighbor of a point $r \in \mathcal{R}$ in the set $P$. Then the set $Q = \{p_r \mid r \in \mathcal{R}\}$ is an $\varepsilon$-kernel of $P$ (see Fig. 4.2). Obviously, $|Q| \le |\mathcal{R}| = O(1/\varepsilon^{(d-1)/2})$. It just remains to show how we can efficiently maintain $Q$ when new points are inserted into $P$, while $P$ remains fat with respect to $B$.

Let Kernel$(P) = \{p_r \in P \mid r \in \mathcal{R}\}$. The function Insert-Box described below inserts a point $p$ into the fat stream $P$ (enclosed by $B$) and returns an $\varepsilon$-kernel of $P$. The algorithm maintains two subsets $Q_0$ and $Q_1$ at each time, which are initially empty.

```
$B$.Insert-Box($p$):
1:  $Q_1 \leftarrow Q_1 \cup \{p\}$
2:  **if** $|Q_1| > 1/\varepsilon^{(d-1)/2}$ **then**
3:      $Q_0 \leftarrow$ Kernel$(Q_0 \cup Q_1)$
4:      $Q_1 \leftarrow \emptyset$
5:  return $Q_0 \cup Q_1$
```

The algorithm divides the stream $P$ into substreams of size $\lfloor 1/\varepsilon^{(d-1)/2} \rfloor$. Whenever a substream is completely received, it is merged to the kernel maintained for the previous substreams in order to obtain a single kernel for the whole stream received so far. The correctness of the algorithm immediately follows from the following two facts: (i) $\textsc{Kernel}(P \cup Q) \subseteq \textsc{Kernel}(P) \cup \textsc{Kernel}(Q)$, and (ii) $\textsc{Kernel}(\textsc{Kernel}(P)) = \textsc{Kernel}(P)$. The kernel in line 3 can be computed using Theorem 4.1 in $O(n + 1/\varepsilon^{d-(3/2)})$ or $O((n + 1/\varepsilon^{d-2}) \log(1/\varepsilon))$ time, where $n = |Q_0 \cup Q_1| = \Theta(1/\varepsilon^{(d-1)/2})$. Therefore, the amortized update time charged to each input point is $\max\{O(1), O((1/\varepsilon^{(d-3)/2}) \log(1/\varepsilon))\}$. We conclude:

**Theorem 4.3** *Given a stream of points $P$ in $\mathbb{R}^d$ which is fat with respect to a fixed hyperbox, an $\varepsilon$-kernel of $P$ can be maintained using $O(1/\varepsilon^{(d-1)/2})$ space and $\max\{O(1), O((1/\varepsilon^{(d-3)/2}) \log(1/\varepsilon))\}$ amortized time per input point.*

**Remark.** In two dimensions, Agarwal and Yu [12] used a balanced binary search tree to maintain an $\varepsilon$-kernel of a fat stream in $O(\log(1/\varepsilon))$ time. Theorem 4.3 immediately improves their method by providing an algorithm that requires only $O(1)$ amortized time. Meanwhile, our method is more direct and does not require any extra data structure.

We will show in the next chapter that a generalized version of the bucketing scheme utilized here can lead to improved algorithms for several other geometric problems.

**Corollary 4.4** *Let $X$ be a self-respecting sequence of $d+1$ points in $\mathbb{R}^d$. Given an $X$-respecting stream $P$ in $\mathbb{R}^d$, an $\varepsilon$-kernel of $P \cup X$ can be maintained using $O(1/\varepsilon^{(d-1)/2})$ space and $\max\{O(1), O((1/\varepsilon^{(d-3)/2}) \log(1/\varepsilon))\}$ amortized update time.*

*Proof.* By Lemma 4.2, $P \cup X$ is fat with respect to $\mathcal{B}_X$. Therefore, we can use the $\textsc{Insert-Box}$ function on $\mathcal{B}_X$ with the only exception that $Q_0$ is initially set to $X$. $\square$

## 4.4 The Main Algorithm

In this section, we describe the main algorithm for maintaining an $\varepsilon$-kernel of a data stream $P \subseteq \mathbb{R}^d$. The $\textsc{Insert}$ function presented below inserts a point $p$ into an $X$-respecting stream $P$ and returns an $\varepsilon$-kernel $Q$ of it. Each new point $p$ is inserted into the stream by calling $P\,$.INSERT$(p, \langle\rangle)$, where $\langle\rangle$ denotes the empty sequence. In the following, $b = \lceil \log(1/\varepsilon) \rceil$, and $\phi_i$ (used in line 15) is a function to be defined precisely at the end of this section.

```
P .INSERT(p, X):
 1: if $p$ is the first point of $P$ then
 2:     $i \leftarrow 1,\ v_1 \leftarrow p$
 3: if $|X| = d + 1$ then
 4:     return $Q = \mathcal{B}_X$.INSERT-BOX$(p)$
 5: if $d_X(p) \le 2 \cdot d_X(v_i)$ then
 6:     $Q_i \leftarrow P_i$ .INSERT$(p, X + \langle v_i \rangle)$
 7: else
 8:     if $|Q_i| > 1/\varepsilon^{(d-1)/2}$ then
 9:         $Q_i \leftarrow \varepsilon$-KERNEL$(Q_i)$
10:     $P_i$ .FREE()
11:     $i \leftarrow i + 1,\ v_i \leftarrow p$
12:     $Q_i \leftarrow P_i$ .INSERT$(p, X + \langle v_i \rangle)$
13:     if $i - b > 0$ then
14:         for each $q \in Q_{i-b}$ do
15:             $q' \leftarrow \phi_1 \circ \cdots \circ \phi_i(q)$
16:             $Q' \leftarrow P'$.INSERT$(q', \langle\rangle)$
17:         $Q_0 \leftarrow \{q \mid q' \in Q'\}$
18:         $P_{i-b}$ .FREE(), $Q_{i-b} \leftarrow \emptyset$
19: return $Q = \cup_{j=0}^{i} Q_j$
```

The algorithm divides the input $X$-respecting stream $P$ into substreams $P_1$ to $P_i$. Each substream $P_i$ is $(X + \langle v_i \rangle)$-respecting, where $v_1$ is the first point of $P$, and each subsequent $v_i$ is chosen by the algorithm as the first point for which $d_X(v_i) > 2 \cdot d_X(v_{i-1})$.

If the new input point $p$ is $(X + \langle v_i \rangle)$-respecting, we add $p$ to the current substream $P_i$ by recursively calling the INSERT function in line 6. When the recursion in the size of $X$ reaches $|X| = d + 1$ (line 3), the stream $P$ is fat with respect to

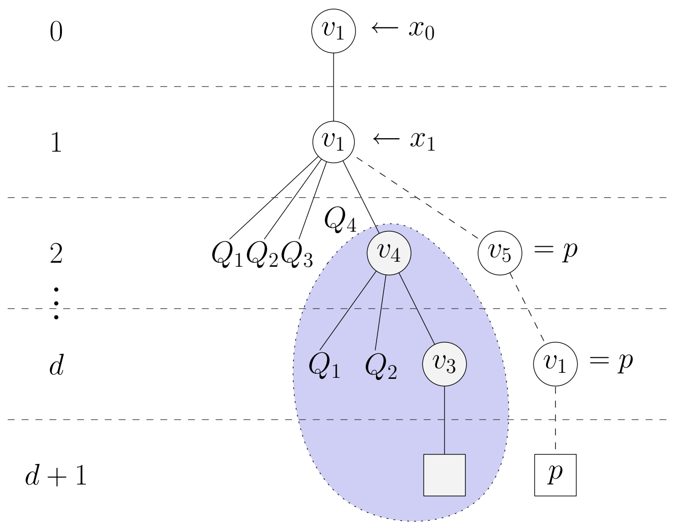

Figure 4.3: An example of the execution of the algorithm. The new inserted point, $p$, is not $\langle x_0, x_1, v_4 \rangle$-respecting. As a result, the coreset maintained for the substream started at $v_4$ is compressed and stored in $Q_4$, the subtree rooted at $v_4$ is discarded, and a new substream is created with $p$ as its first point.

$\mathcal{B}_X$, and therefore, we can use the Insert-Box function described in Section 4.3 (Corollary 4.4) to compute an $\varepsilon$-kernel of $P$ of size $O(1/\varepsilon^{(d-1)/2})$.

If the new point $p$ is not $(X + \langle v_i \rangle)$-respecting, then we need to open a new substream $P_{i+1}$ for $p$. But before that, we check the size of the $\varepsilon$-kernel $Q_i$ currently maintained for $P_i$, and if $|Q_i| > 1/\varepsilon^{(d-1)/2}$, we reduce it to $O(1/\varepsilon^{(d-1)/2})$ (in line 9) using Theorem 4.1. After this compression step, all kernels previously maintained for $P_i$ and its substreams are discarded by calling function Free in line 10. Lines 11–12 increment $i$ and create a new substream $P_i$ initialized with $\{p\}$. (See Figure 4.3.)

Lines 13–18 ensure that only $b$ coresets $Q_{i-b+1}, \ldots, Q_i$ are active; earlier ones are mapped into a $(d-1)$-dimensional substream $P'$ whose coreset is maintained in $Q'$. The mapping $\phi_1 \circ \cdots \circ \phi_i$ used in line 15 is defined as follows: let $\hat{v}_i = \mathrm{proj}(\mathcal{F}(X), v_i)$, and let $u_i = \overrightarrow{\hat{v}_i v_i}$. We denote by $\mathcal{H}_0$ an arbitrary hyperplane through $X$, and for $1 \leq i \leq d$, denote by $\mathcal{H}_i$ the hyperplane through $X$ perpendicular to $u_i$. The function $\phi_i$ denotes the projection to $\mathcal{H}_{i-1}$ parallel to the direction $u_i$ (see Figure 4.4). We keep the mapping function $\phi_1 \circ \cdots \circ \phi_i$ in a single matrix and update it only once whenever $i$ is increased.

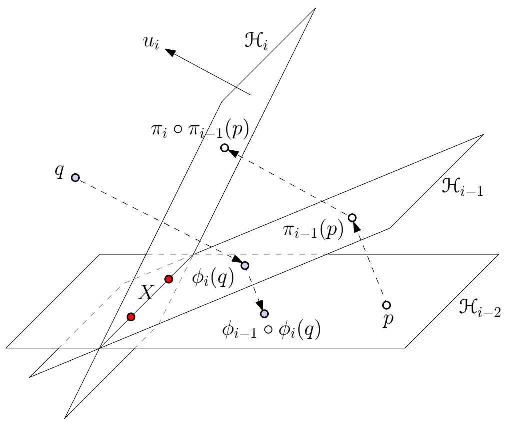

Figure 4.4: Mapping functions $\pi_i$ and $\phi_i$.

### 4.4.1 Analysis

In this section we prove the correctness of our algorithm, and analyze its space and time complexity. The notation used here closely follows the one used in [21]. Let $\pi_i$ denote the orthogonal projection onto $\mathcal{H}_i$. Note that $\pi_i$ is a weak inverse of $\phi_i$ in the sense that $\pi_i \circ \cdots \circ \pi_1 \circ \phi_1 \circ \cdots \circ \phi_i = \pi_i$ (see Figure 4.4). In the following, $f$ denotes the final value of $i$, and $\xi = \pi_f \circ \cdots \circ \pi_1$. We first prove two technical lemmas.

**Lemma 4.5** *For every $i \leq f$ and every direction $u \in \mathbb{S}^{d-1}$, $\langle v_i - \pi_i(v_i), u \rangle \leq 4^d w(P \cup X, u)$.*

*Proof.* Let $\mathcal{X}$ be a sequence of $d+1$ points obtained as follows: starting from $\mathcal{X} = X$, we repeatedly add to $\mathcal{X}$ a point from $P$ which is farthest from $\mathcal{F}(\mathcal{X})$, until $\mathcal{X}$ has $d+1$ points. Obviously, $P$ is $\mathcal{X}$-respecting and $P \cup \mathcal{X} = P \cup X$. Thus, by Lemma 4.2, $\mathrm{conv}(P \cup X)$ is sandwiched between $\mathcal{B}_{\mathcal{X}}$ and a translated copy of $(1/4)^d \mathcal{B}_{\mathcal{X}}$. Both $v_i$ and $\pi_i(v_i)$ lie inside $\mathcal{B}_{\mathcal{X}}$. Therefore, for each direction $u \in \mathbb{S}^{d-1}$,

$$\langle v_i - \pi_i(v_i), u \rangle \leq w(\mathcal{B}_{\mathcal{X}}, u) = 4^d w((1/4)^d \mathcal{B}_{\mathcal{X}}, u) \leq 4^d w(P \cup X, u).$$

□

**Lemma 4.6** *Let $q \in Q_{j-b}$ for some $b < j \leq f$, and let $q' = \phi_1 \circ \cdots \circ \phi_j(q)$. Then for every direction $u \in \mathbb{S}^{d-1}$, $\langle q - \xi(q'), u \rangle \leq 4^{d+1} \varepsilon w(P \cup X, u)$.*

*Proof.* By the doubling property we have $d_X(q) \le 2d_X(v_{j-b}) \le 2^{1-b}d_X(v_j)$. Moreover, $\langle q - \pi_i(q), u\rangle / d(q, \mathcal{H}_i) = \langle v_i - \pi_i(v_i), u\rangle / d(v_i, \mathcal{H}_i)$. Since $d_X(v_i) = d(v_i, \mathcal{H}_i)$ and $d_X(q) \ge d(q, \mathcal{H}_i)$, we have

$$\langle q - \pi_i(q), u\rangle \le \frac{d_X(q)}{d_X(v_i)} \langle v_i - \pi_i(v_i), u\rangle \le \frac{d_X(q)}{d_X(v_i)} \cdot 4^d w(P \cup X, u), \tag{4.1}$$

where the last inequality holds by Lemma 4.5. Now, define $q_j = q$, and $q_t = \pi_{t-1} \circ \cdots \circ \pi_j(q)$ for all $t > j$. It is clear that $\pi_i(q_i) = q_{i+1}$. Therefore,

$$\sum_{i=j}^{f} \langle q_i - \pi_i(q_i), u\rangle = \sum_{i=j}^{f} \langle q_i - q_{i+1}, u\rangle \ge \langle q_j - q_{f+1}, u\rangle . \tag{4.2}$$

Furthermore,

$$\sum_{i=j}^{f} \frac{d_X(q_i)}{d_X(v_i)} \le \sum_{i=j}^{f} \frac{1}{2^{(j-i)}} \cdot \frac{d_X(q_i)}{d_X(v_j)} \le 2 \cdot \frac{d_X(q)}{d_X(v_j)} \le 2(2^{1-b}) \le 4\varepsilon.$$

Therefore, if we replace $q$ by $q_i$ in (4.1) and sum up the inequality over $i$ from $j$ to $f$, we get

$$\sum_{i=j}^{f} \langle q_i - \pi_i(q_i), u\rangle \ \le \ \sum_{i=j}^{f} \frac{d_X(q_i)}{d_X(v_i)} \cdot 4^d w(P \cup X, u) \ \le \ 4^{d+1}\varepsilon w(P \cup X, u). \tag{4.3}$$

The lemma statement follows by (4.2) and (4.3) and the fact that $q_{f+1} = \pi_f \circ \cdots \circ \pi_j(q) = \xi(q')$, due to the weak-inverse relationship between $\pi_i$'s and $\phi_i$'s. □

**Theorem 4.7** *Given a stream of points $P$ in $\mathbb{R}^d$, an $\varepsilon$-kernel of $P$ can be maintained using $O((1/\varepsilon^{(d-1)/2})\log(1/\varepsilon))$ space and $\max\{O(1), O((1/\varepsilon^{(d-3)/2})\log(1/\varepsilon))\}$ update time.*

*Proof.* We show that for every $X$-respecting stream $P$, the set $Q$ returned by our algorithm is an $\varepsilon$-kernel of $P \cup X$. If $|X| = d+1$, then the set $Q$ computed in line 4 is an $\varepsilon$-kernel of $P \cup X$ by Corollary 4.4. Otherwise, $|X| \le d$. Consider an arbitrary point $p \in P$. The algorithm inserts $p$ into a substream $P_i$ ($1 \le i \le f$) upon its arrival. If $i = f$, then the set $Q_i$ is an $\varepsilon$-kernel of $P_i \cup X$ by induction. If $f - b < i < f$, then $Q_i$ has passed the compression step in lines 8–9, but it is still active, i.e., is not merged into $Q_0$. Therefore, $Q_i$ is a $(2\varepsilon)$-kernel of $P_i \cup X$ due to the fact that an $\varepsilon$-kernel of an $\varepsilon'$-kernel of a set is an $(\varepsilon + \varepsilon')$-kernel of that set. The only remaining case is when $i \le f - b$. In this case, $Q_i$ is merged into $Q_0$ in lines 13–18. Since $Q_i$ is a $(2\varepsilon)$-kernel of $P_i \cup X$ before merging, there exists a point

$q \in Q_i$ such that for every direction $u \in \mathbb{S}^{d-1}$, $\langle q, u\rangle \geq \langle p, u\rangle - 2\varepsilon w(P_i \cup X, u)$. The mapped point of $q$, $q'$, is inserted into $P'$ in line 16. Since $Q'$ is an $\varepsilon$-kernel of $P'$, $\xi(Q')$ is an $\varepsilon$-kernel of $\xi(P')$. Moreover, there exists a point $r \in Q_0$ with $r' \in Q'$, such that $\langle r', u\rangle \geq \langle q', u\rangle - \varepsilon w(P', u)$, and hence

$$\langle \xi(r'), u\rangle \geq \langle \xi(q'), u\rangle - \varepsilon w(\xi(P'), u). \tag{4.4}$$

Let $\rho = 4^{d+1}$. By Lemma 4.6, for every $q \in Q_1 \cup \cdots \cup Q_{f-b}$ and its corresponding $q' \in P'$,

$$\langle q, u\rangle - \rho\varepsilon w(P \cup X, u) \leq \langle \xi(q'), u\rangle \leq \langle q, u\rangle + \rho\varepsilon w(P \cup X, u),$$

which implies that $w(\xi(P'), u) \leq w(Q_1 \cup \cdots \cup Q_{f-b}, u) + 2\rho\varepsilon w(P \cup X, u) \leq (1 + 2\rho\varepsilon)w(P \cup X, u)$. Furthermore, by Lemma 4.6 we have $\langle \xi(r'), u\rangle \leq \langle r, u\rangle + \rho\varepsilon w(P \cup X, u)$ and $\langle \xi(q'), u\rangle \geq \langle q, u\rangle - \rho\varepsilon w(P \cup X, u)$. Replacing in (4.4), we get

$$\langle r, u\rangle + \rho\varepsilon w(P \cup X, u) \geq \langle q, u\rangle - \rho\varepsilon w(P \cup X, u) - \varepsilon[(1 + 2\rho\varepsilon)w(P \cup X, u)],$$

and hence, $\langle r, u\rangle \geq \langle q, u\rangle - O(\varepsilon)w(P \cup X, u)$. Since $\langle q, u\rangle \geq \langle p, u\rangle - 2\varepsilon w(P \cup X, u)$, we have $\langle r, u\rangle \geq \langle p, u\rangle - O(\varepsilon)w(P \cup X, u)$. Therefore, in any of the above cases, there exists a point in $\cup_{i=0}^{f} Q_i$ whose projected length along direction $u$ differs from that of $p$ by at most $O(\varepsilon)w(P \cup X, u)$, and hence, $Q = \cup_{i=0}^{f} Q_i$ is an $O(\varepsilon)$-kernel of $P \cup X$. (Note that in our proof, the algorithm returns an $O(\varepsilon)$-kernel rather than an $\varepsilon$-kernel; but this is not a problem as the depth of the recursion tree of our algorithm is $d + 1$, and therefore, we can adjust $\varepsilon$ at the beginning by a constant, depending only on $d$.)

**Space Complexity.** Let $S(d, k)$ denote the space used by the algorithm to compute an $\varepsilon$-kernel of the $d$-dimensional $X$-respecting stream $P$, where $|X| = k$. Then, $|Q_0| = |Q'| = S(d-1, 0)$, $|Q_1|, \ldots, |Q_{f-b}| = 0$ by the merging step, $|Q_{f-b+1}|, \ldots, |Q_{f-1}| = O(1/\varepsilon^{(d-1)/2})$ by the compression step, and $|Q_f| = S(d, k+1)$. Therefore, $S(d, k)$ is upper-bounded by the following recurrence:

$$S(d,k) = \begin{cases} S(d, k+1) + S(d-1, 0) + \log(1/\varepsilon)O(1/\varepsilon^{(d-1)/2}) & 0 \leq k \leq d, \\ O(1/\varepsilon^{(d-1)/2}) & k = d+1, \\ O(1) & d = 1, \end{cases}$$

which solves to $S(d, k) = O((1/\varepsilon^{(d-1)/2})\log(1/\varepsilon))$, for every $0 \leq k \leq d$.

**Update Time.** The compression step in line 9 can be done using Theorem 4.1 in $O(|Q_i|+1/\varepsilon^{d-(3/2)})$ or $O((|Q_i|+1/\varepsilon^{d-2})\log(1/\varepsilon))$ time. Since $|Q_i| = \Theta(1/\varepsilon^{(d-1)/2})$), we can charge an amortized time of $\max\{O(1), O((1/\varepsilon^{(d-3)/2})\log(1/\varepsilon))\}$ to each point of $Q_i$. In lines 14–16, each point is inserted into $P'$ at most once. Therefore, the cost of insertion into the $(d-1)$-dimensional stream $P'$ can be charged to each point upon its insertion to a $P_i$, which at most doubles its total insertion cost. The main cost incurred by each point is therefore the time needed to insert the point into a fat subset, which is $\max\{O(1), O((1/\varepsilon^{(d-3)/2})\log(1/\varepsilon))\}$ by Corollary 4.4. □

### 4.4.2 Reducing Update Time

While the main focus in designing streaming algorithms is to optimize the working storage, the time needed to process each element is also of particular interest, especially in applications where a huge amount of data arrives in a short period of time. For the problem of maintaining $\varepsilon$-kernels in $\mathbb{R}^d$, Chan [21] proposed a streaming algorithm that processes each input point in $O(1)$ time using a data structure of size $O([(1/\varepsilon)\log(1/\varepsilon)]^{d-1})$. Here, we show how to improve the space complexity of Chan's algorithm for all fixed dimensions, while the optimal update time, $O(1)$, is preserved.

We provide a general framework to trade-off between time and space complexity in our algorithm as follows: Let $\lambda(\varepsilon) = \Omega(1/\varepsilon^{(d-1)/2})$ be a function of $\varepsilon$. We replace $1/\varepsilon^{(d-1)/2}$ by $\lambda(\varepsilon)$ in line 2 of the Insert-box function and in line 9 of the main Insert function. It is easy to verify that the amortized update time of the new algorithm is $O([1/\varepsilon^{d-(3/2)}]/\lambda(\varepsilon))$ for $d \geq 2$, and $O(\log(1/\varepsilon)+[(1/\varepsilon^{d-2})\log(1/\varepsilon)]/\lambda(\varepsilon))$ for $d \geq 3$. Furthermore, the space complexity of the algorithm is upper-bounded by the recurrence $S(d,k) = S(d,k+1) + S(d-1,0) + \log(1/\varepsilon)O(\lambda(\varepsilon))$, with the base cases $S(d,d+1) = O(\lambda(\varepsilon))$ and $S(1,k) = O(1)$. The recurrence solves to $S(d,k) = O(\lambda(\varepsilon)\log(1/\varepsilon))$. Setting $\lambda(\varepsilon) = 1/\varepsilon^{d-(3/2)}$, we immediately get the following result:

**Theorem 4.8** *Given a stream of points $P$ in $\mathbb{R}^d$, an $\varepsilon$-kernel of $P$ can be maintained using $O((1/\varepsilon^{d-(3/2)})\log(1/\varepsilon))$ space and $O(1)$ amortized update time.*

Note that the above theorem improves the best previous time-optimal streaming algorithm of Chan without using the common dimension-reduction approach used in [21] and [12]. Moreover, by setting $\lambda(\varepsilon) = 1/\varepsilon^{d-2}$, we obtain a streaming algorithm with space complexity $O((1/\varepsilon^{d-2})\log(1/\varepsilon))$ and amortized update time $O(\log(1/\varepsilon))$, which for all $d \geq 3$, improves over the algorithm of Agarwal and Yu [12] without increasing their update time.

## 4.5 Applications

In this section, we briefly review some of the implications of our result. Consider a measure $\mu$ so that for any point set $P \subseteq \mathbb{R}^d$, an $\varepsilon$-kernel of $P$ is an $O(\varepsilon)$-coreset for $P$ with respect to $\mu$. Examples of such measures include diameter, width, radius of the smallest enclosing ball, and volume of the smallest bounding box. Theorem 4.7 provides space-efficient streaming algorithms to maintain $\varepsilon$-approximations to all these measures using near $O(1/\varepsilon^{(d-1)/2})$ space. Note that $O(1/\varepsilon^{(d-1)/2})$-space streaming algorithms were previously known only for diameter [7], while the best space bound for other measures was $O(1/\varepsilon^{d-(3/2)})$ [12]. Using the general technique described in [4], our result implies improved streaming algorithms for various other shape-fitting problems like minimum-width spherical shell/annulus and minimum-width cylindrical shell. Improved results for kinetic versions of the above problems (where input is a stream of moving points) are implied as well. In these kinetic versions, the trajectory of each moving point is assumed to be a fixed algebraic function determined upon arriving the point, and the objective is to maintain an approximation at any time for the points received up to that time.

Our streaming algorithm can be also used in noisy environments, using the "robust kernel" paradigm proposed in [62, 6]. Roughly speaking, a subset $Q \subseteq P$ is called a $(k, \varepsilon)$-kernel of $P$, if $Q$ $\varepsilon$-approximates the directional width of $P$, for any direction, when $k$ outliers can be ignored in that direction. According to [6], a $(k, \varepsilon)$-kernel of a point set $P$ can be obtained as follows. Let $P_1 = P$, and $Q_1$ be an $\varepsilon$-kernel of $P_1$. For $2 \leq i \leq 2k + 1$, let $P_i = P_{i-1} \backslash Q_{i-1}$, and $Q_i$ be an $\varepsilon$-kernel of $P_i$. Then the set $Q = \cup_{i=1}^{2k+1} Q_i$ is a $(k, \varepsilon)$-kernel of $P$ [6]. As noted in [12], this idea can be easily imported to the data stream model as well: we run $2k + 1$ instances $\mathcal{A}_1, \ldots, \mathcal{A}_{2k+1}$ of our $\varepsilon$-kernel algorithm in parallel, where each algorithm $\mathcal{A}_i$ maintains an $\varepsilon$-kernel $Q_i$ of its input stream $P_i$ ($1 \leq i \leq 2k + 1$). Each new point of the input stream $P$ is first inserted into $P_1$. If $p$ is not added by $\mathcal{A}_1$ into $Q_1$, we insert it into $P_2$ and proceed recursively. Moreover, any point removed from $Q_i$ ($1 \leq i \leq 2k$) at each step is inserted into the next substream $P_{i+1}$. It is easy to verify that at any time, $P_i = P_{i-1} \backslash Q_{i-1}$, and $Q_i$ is an $\varepsilon$-kernel of $P_i$. Thus, Theorem 4.7 immediately implies the following result:

**Corollary 4.9** *Given a stream $P$ of points in $\mathbb{R}^d$ and a parameter $k \geq 0$, a $(k, \varepsilon)$-kernel of $P$ can be maintained using $O((k/\varepsilon^{(d-1)/2}) \log(1/\varepsilon))$ space.*

The kernel size obtained by Corollary 4.9 substantially improves over the previous known upper bound $O(k/\varepsilon^{d-(3/2)})$ [12], and is very close to the lower bound which is proved to be $O(k/\varepsilon^{(d-1)/2})$ in the worst case [62].

## 4.6 Conclusions

In this chapter, we presented a streaming algorithm for maintaining an $\varepsilon$-kernel of a stream of points in $\mathbb{R}^d$ using $O((1/\varepsilon^{(d-1)/2}) \log(1/\varepsilon))$ space. The space complexity of our algorithm is optimal up to a $\log(1/\varepsilon)$ factor. In the special case of two dimensions, Agarwal and Yu [12] proposed a rather involved technique to remove this extra log factor at the expense of increasing update time from $O(1)$ to $O(\log(1/\varepsilon))$. It remains open whether this small log factor can be removed in any fixed dimension.

# Chapter 5

# Core-Preserving Algorithms

In this chapter, we define a class of algorithms for constructing coresets of (geometric) data sets, and show that algorithms in this class can be dynamized efficiently in the insertion-only (data stream) model. As a result, we show that for a set of points in fixed dimensions, additive and multiplicative $\varepsilon$-coresets for the $k$-center problem can be maintained in $O(1)$ and $O(k)$ time respectively, using a data structure whose size is independent of the size of the input. We also provide a faster streaming algorithm for maintaining $\varepsilon$-coresets for fat extent-related problems such as diameter and minimum enclosing ball.

## 5.1 Introduction

In the previous chapter, we have shown that an $\varepsilon$-kernel of a point set in fixed dimensions can be maintained efficiently using near optimal space. The main idea behind our algorithm was to decompose the given stream into $d$-dimensional fat substreams, where each substream is fat with respect to a fixed hyperbox. In this chapter, we show that a simpler partitioning framework similar to what is used in Section 4.3 can be employed to obtain still better results for a number of problems in the data stream model.

As noted in Section 1.4, several streaming algorithms have been developed over the past few years for various geometric problems using the notion of coresets [21,

An extended abstract of this chapter has been published as: H. Zarrabi-Zadeh. Core-preserving algorithms. In *Proc. 20th Canadian Conference on Computational Geometry*, pages 159–162, 2008.

12, 47, 60]. For all these problems, the coresets defined satisfy the following two properties:

a) If $Q$ is an $\varepsilon$-coreset of $P$ and $Q'$ is an $\varepsilon$-coreset of $P'$, then $Q \cup Q'$ is an $\varepsilon$-coreset of $P \cup P'$;

b) If $Q$ is an $\varepsilon$-coreset of $S$ and $S$ is a $\delta$-coreset of $P$, then $Q$ is an $(\varepsilon + \delta)$-coreset of $P$.

Using the above two properties and based on the general dynamization technique of Bentley and Saxe [14], Agarwal *et al.* [4] obtained the following result in the data stream model: If there is an (offline) algorithm that constructs an $\varepsilon$-coreset of size $f(\varepsilon)$ for an optimization problem, then the problem can be solved in the data stream model using $O(f(\varepsilon/\log^2 n)\log n)$ overall space, where $n$ is the number of elements received so far in the stream.

In this chapter, we show that for a special class of algorithms which we call core-preserving, the space complexity of the streaming algorithm can be reduced to $f(\varepsilon)$, using a simple bucketing scheme. The importance of this result is that the dependency on the space complexity to the input size $n$ is removed. (Such a result was previously known only for the $\varepsilon$-coresets with respect to the extent measure [21, 12].) This independence on the input size is very important as the input size in data streams is generally huge.

Our framework leads to improved algorithms for a number of problems in the data stream model, some of which are listed below. In the following, the input is assumed to be a stream of points in $\mathbb{R}^d$, where $d$ is a constant.

- **(Additive) coreset for $k$-center**: We show that an additive $\varepsilon$-coreset for the $k$-center problem (as defined in Section 5.2) can be maintained in $O(k/\varepsilon^d)$ space and $O(1)$ amortized update time, improving the previous algorithm attributed to Har-peled [57] which requires $O(\text{poly}(k, 1/\varepsilon, \log n))$ space and similar time. This is indeed the first streaming algorithm maintaining an $\varepsilon$-coreset for this problem using a total space independent of $n$.

- **Multiplicative coreset for $k$-center**: For the $k$-center problem, we show that a multiplicative $\varepsilon$-coreset (as defined in Section 5.2) can be maintained in $O(k!/\varepsilon^{kd})$ space and $O(k)$ amortized update time. This is again the first streaming algorithm for this problem whose space is independent of the input size. This result immediately extends to a variant of the $k$-clustering problem in which the objective is to minimize the sum of the clusters radii [27, 49].

- **Coreset for fat measures**: For "fat" measures such as diameter and radius of the minimum enclosing ball, one can easily maintain an $\varepsilon$-coreset by just keeping the extreme points along $O(1/\varepsilon^{(d-1)/2})$ directions. The time and space complexity of this naïve algorithm is $O(1/\varepsilon^{(d-1)/2})$. In two-dimensions, using the recent algorithm of Agarwal and Yu [12], one can improve the update time from $O(\sqrt{1/\varepsilon})$ to $O(\log(1/\varepsilon))$. We show that the update time in 2D can be further reduced to $O(1)$ using our framework. Moreover, the update time in three dimensions is reduced from $O(1/\varepsilon)$ to $O(\log(1/\varepsilon))$ using our algorithm. A slight improvement in higher dimensions is implied as well.

## 5.2 Preliminaries

Let $P$ be a set of points in $\mathbb{R}^d$. A *$k$-clustering* of $P$ is a set $\mathcal{B}$ of $k$ balls that completely cover $P$. We denote by $\text{rad}(b)$ the radius of a ball $b$, and define $\text{rad}(\mathcal{B}) = \max_{b\in\mathcal{B}} \text{rad}(b)$. A *$\delta$-expansion* of $\mathcal{B}$ is obtained by increasing the radius of each ball of $\mathcal{B}$ by an additive factor of $\delta$.

**Definition 5.1** A set $Q \subseteq P$ is called an *additive $\varepsilon$-coreset* of $P$ for the $k$-center problem, if for every $k$-clustering $\mathcal{B}$ of $Q$, $P$ is covered by an $(\varepsilon \cdot \text{rad}(\mathcal{B}))$-expansion of $\mathcal{B}$.

We denote by $(1+\varepsilon)\mathcal{B}$ a clustering obtained from $\mathcal{B}$ by expanding each ball $b \in \mathcal{B}$ by a factor of $\varepsilon \cdot \text{rad}(b)$.

**Definition 5.2** A set $Q \subseteq P$ is called a *multiplicative $\varepsilon$-coreset* of $P$ for the $k$-center problem, if for every $k$-clustering $\mathcal{B}$ of $Q$, $P$ is covered by $(1+\varepsilon)\mathcal{B}$.

Note that by definition, a multiplicative $\varepsilon$-coreset for $k$-center is also an additive $\varepsilon$-coreset. In the special case of the minimum enclosing ball (the 1-center) problem, both coreset definitions coincide.

Given two points $p, q \in \mathbb{R}^d$, we say that $p$ is *smaller* than $q$, if $p$ lies before $q$ in the lexicographical order of their coordinates. Throughout this chapter, we denote by $\lfloor x \rfloor_2$ the largest (integer) power of 2 which is less than or equal to $x$.

## 5.3 Core-Preserving Algorithms

In this section, we formally define the concept of core-preserving algorithms, and show how it can be used to efficiently maintain coresets in data streams.

**Definition 5.3** *Let $\mathcal{A}$ be an (offline) algorithm that for every input set $P$, computes an $\varepsilon$-coreset $\mathcal{A}(P)$ of $P$. We call $\mathcal{A}$* core-preserving*, if for every two sets $R$ and $S$, $\mathcal{A}(R \cup \mathcal{A}(S))$ is an $\varepsilon$-coreset of $R \cup S$.*

For $R = \emptyset$, the above condition implies that $\mathcal{A}(\mathcal{A}(S))$ is an $\varepsilon$-coreset of $S$. It means that repeated calls to a core-preserving algorithm on a set $S$ always returns an $\varepsilon$-coreset of $S$. This is why the algorithm is called "core-preserving".

**Theorem 5.1** *Let $\mathcal{A}$ be a core-preserving algorithm that for any set $S$, computes an $\varepsilon$-kernel of $S$ of size $O(\mathcal{S}_{\mathcal{A}}(\varepsilon))$ in time $O(\alpha|S| + \mathcal{T}_{\mathcal{A}}(\varepsilon))$. Then for every stream $P$, we can maintain an $\varepsilon$-coreset of $P$ of size $O(\mathcal{S}_{\mathcal{A}}(\varepsilon))$ using $O(\mathcal{S}_{\mathcal{A}}(\varepsilon))$ total space and $O(\alpha + \mathcal{T}_{\mathcal{A}}(\varepsilon)/\mathcal{S}_{\mathcal{A}}(\varepsilon))$ amortized time per update.*

*Proof.* The function Insert described below inserts a data item $p$ into the stream $P$ and returns an $\varepsilon$-kernel of $P$. Initially, $Q$ and $R$ are empty sets.

```
Insert($p$):
1:  $R \leftarrow R \cup \{p\}$
2:  if $|R| > \mathcal{S}_{\mathcal{A}}(\varepsilon)$ then
3:      $Q \leftarrow \mathcal{A}(R \cup Q)$
4:      $R \leftarrow \emptyset$
5:  return $R \cup Q$
```

The algorithm divides the input stream $P$ into buckets of size $\lfloor \mathcal{S}_{\mathcal{A}}(\varepsilon) \rfloor$. At any time, only the last bucket is active which is maintained in the set $R$. Let $S = P \backslash R$. The algorithm maintains an $\varepsilon$-coreset of $S$ in $Q$. Upon arrival of a new item $p$, it is first added to the active bucket $R$, and if $R$ is full, algorithm $\mathcal{A}$ is invoked to compute an $\varepsilon$-coreset of $R \cup Q$. The correctness of the algorithm immediately follows from the facts that $\mathcal{A}$ is core-preserving and $Q$ is an $\varepsilon$-coreset of $S$; thus, $\mathcal{A}(R \cup Q)$ is an $\varepsilon$-coreset of $R \cup S = P$.

The total space used by the algorithm is bounded by $|Q|+|R| = O(\mathcal{S}_{\mathcal{A}}(\varepsilon))$. Algorithm $\mathcal{A}$ is invoked once per $\lceil \mathcal{S}_{\mathcal{A}}(\varepsilon) \rceil$ inserts. Since each call to $\mathcal{A}$ requires $O(\alpha|S| + \mathcal{T}_{\mathcal{A}}(\varepsilon))$ time, the amortized update time per input is $O(\alpha + \mathcal{T}_{\mathcal{A}}(\varepsilon)/\mathcal{S}_{\mathcal{A}}(\varepsilon))$. $\square$

Theorem 5.1 yields two improvements over the general Bentley-Saxe method used in [4]: First of all, the total space required is reduced from $O(\mathcal{S}_{\mathcal{A}}(\varepsilon/\log^2 n)\log n)$ to $O(\mathcal{S}_{\mathcal{A}}(\varepsilon))$, which is independent of $n$. Secondly, the running time in the worst case is reduced from $O([\alpha \mathcal{S}_{\mathcal{A}}(\varepsilon/\log^2 n) + \mathcal{T}_{\mathcal{A}}(\varepsilon/\log^2 n)]\log n)$ to only $O(\alpha|P| + \mathcal{T}_{\mathcal{A}}(\varepsilon))$, again independent of $n$.

## 5.4 Coresets for the $k$-Center Problem

In this section, we provide efficient streaming algorithms for maintaining $\varepsilon$-coresets for the $k$-center problem in both additive and multiplicative forms.

### 5.4.1 Additive Coreset

The following lemma is the main ingredient of our streaming algorithm for maintaining an additive $\varepsilon$-coreset for the $k$-center problem in fixed dimensions.

**Lemma 5.2** *There is a core-preserving algorithm that for any given point set $P \subseteq \mathbb{R}^d$, computes an additive $\varepsilon$-coreset for the $k$-center problem of size $O(k/\varepsilon^d)$ in time $O(|P| + k/\varepsilon^d)$.*

*Proof.* Let $r^*(P)$ be the radius of an optimal $k$-clustering of $P$, and let $\tilde{r}(P)$ be a 2-approximation of $r^*(P)$, i.e., $r^*(P) \le \tilde{r}(P) \le 2r^*(P)$ (note that $\tilde{r}(P)$ can be computed using Gonzalez's 2-approximation algorithm for $k$-center [51]).

We first define some notation: Let $\mathcal{G}_\alpha$ be a uniform grid of side length $\alpha$, and $X_\alpha(P)$ be the set of all $p \in P$, such that $p$ is the smallest point in a non-empty grid cell of $\mathcal{G}_\alpha$. Let $\delta(P) = \lfloor \varepsilon \tilde{r}(P)/(4d^{1/2}) \rfloor_2$. Our core-preserving algorithm is as follows: given a point set $P$, we first compute $\delta = \delta(P)$, and return $X_\delta(P)$ as the output. It is easy to observe that any $k$-clustering of $X_\delta(P)$, when expanded by a factor of $\varepsilon r^*(P)$, covers all the grid cells containing at least one point from $P$, and therefore, $X_\delta(P)$ is an $\varepsilon$-coreset of $P$ [8, 58].

Let $R$ and $S$ be two arbitrary point sets in $\mathbb{R}^d$, and let $Q$ be an $\varepsilon$-coreset of $S$ computed by our algorithm. To show that our algorithm is core-preserving, we

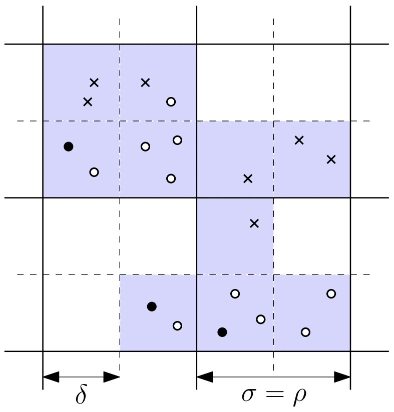

Figure 5.1: Additive coreset for $k$-center. The points of $Q$, $S \backslash Q$, and $R$ are shown in black, white, and crossed, respectively.

need to prove that for any input of the form $P = R \cup Q$, the algorithm returns an $\varepsilon$-coreset of $R \cup S$.

Let $\delta = \delta(P)$, $\sigma = \delta(S)$, and $\rho = \max\{\delta(P), \delta(S)\}$. Obviously, $X_\rho(R \cup S)$ is an $\varepsilon$-coreset of $R \cup S$, because both $P$ and $S$ are subsets of $R \cup S$, and hence, $\max\{\tilde{r}(P), \tilde{r}(S)\} \leq 2r^*(R \cup S)$. We claim that $X_\rho(R \cup S) \subseteq X_\delta(R \cup Q)$. Since $\rho/\delta$ (resp., $\rho/\sigma$) is a non-negative power of 2, every grid cell of $\mathcal{G}_\delta$ (resp., $\mathcal{G}_\sigma$) is completely contained in a grid cell of $\mathcal{G}_\rho$ (see Figure 5.1). Let $p$ be the smallest point of $R \cup S$ in a grid cell $c$ of $\mathcal{G}_\rho$. Two cases arise:

- $p \in R$: in this case, $p$ is the smallest point of a cell $c' \in \mathcal{G}_\delta$ (otherwise, there is a point $p'$ smaller than $p$ in $c'$, which is smaller than $p$ in $c$ as well, a contradiction). Therefore, $p \in X_\delta(R \cup Q)$.

- $p \in S$: here, $p$ is simultaneously the smallest point of a cell $c' \in \mathcal{G}_\sigma$ and a cell $c'' \in \mathcal{G}_\delta$ (otherwise, if there is a smaller point $p'$ in either $c'$ or $c''$, it would be picked instead of $p$ as the smallest point of $c$, a contradiction). Since $p$ is the smallest point in $c'$, we have $p \in Q$, and since $p$ is the smallest point of $c''$, we conclude that $p \in X_\delta(R \cup Q)$.

Therefore, any $p \in X_\rho(R \cup S)$ is contained in $X_\delta(R \cup Q) = X_\delta(P)$, which completes the proof.

For the space complexity, note that every ball of an optimal $k$-clustering of $P$ intersects $O(1/\varepsilon^d)$ grid cells of $\mathcal{G}_\delta$. Therefore, the size of the resulting $\varepsilon$-coreset is $O(k/\varepsilon^d)$. We can use a linear-time implementation of Gonzalez's algorithm [51, 57] to compute a 2-approximation of $r^*(P)$, and therefore, the total running time required is $O(|P| + k/\varepsilon^d)$. $\square$

Plugging Lemma 5.2 into the general framework provided in Theorem 5.1, we immediately get the following result.

**Theorem 5.3** *Given a stream of points $P$ in $\mathbb{R}^d$, an additive $\varepsilon$-coreset for the $k$-center problem of size $O(k/\varepsilon^d)$ can be maintained using $O(k/\varepsilon^d)$ total space and $O(1)$ amortized time per update.*

The above results also hold for any $L_p$ metric: it just suffices to replace $d^{1/2}$ by $d^{1/p}$ in the definition of $\delta(P)$.

### 5.4.2 Multiplicative Coreset

Here, we provide an efficient streaming algorithm to maintain a multiplicative $\varepsilon$-coreset for the $k$-center problem in fixed dimensions.

**Lemma 5.4** *There is a core-preserving algorithm that for any given point set $P \subseteq \mathbb{R}^d$, computes a multiplicative $\varepsilon$-coreset for the $k$-center problem of size $O(k!/\varepsilon^{kd})$ in time $O(k|P| + k!/\varepsilon^{kd})$.*

*Proof.* It is known that constructing a multiplicative $\varepsilon$-coreset for the $k$-center problem reduces to the problem of finding additive $\varepsilon$-coresets for $k$-center [9, 58]. Let $r^*(P)$, $\tilde{r}(P)$, $\delta(P)$, $\mathcal{G}_\delta$, and $X_\delta(P)$ be as defined in the Section 5.4.1. The algorithm for computing multiplicative $\varepsilon$-coreset is as follows: We first compute $\delta = \delta(P)$, build a uniform grid $\mathcal{G}_\delta$, and extract an additive $\varepsilon$-coreset $X_\delta(P)$ just like in Section 5.4.1. Let $\mathcal{C}$ be the set of non-empty grid cells in $\mathcal{G}_\delta$. For every cell $c \in \mathcal{C}$, we recursively compute a multiplicative $\varepsilon$-coreset $Q(c)$ of $P \cap c$ with respect to $(k-1)$-center. The set $\mathcal{Q}(P) = X_\delta(P) \cup (\cup_{c \in \mathcal{C}} Q(c))$ is a multiplicative $\varepsilon$-coreset of $P$ with the following simple argument.

Let $\mathcal{B}$ be a $k$-clustering of $\mathcal{Q}(P)$, and let $b$ be the largest ball in $\mathcal{B}$. Consider a non-empty cell $c$ in $\mathcal{C}$. If $b$ intersects $c$, then an $\varepsilon$-expansion of $b$ by a factor of $\varepsilon \cdot \mathrm{rad}(b) = \varepsilon r^*(P)$ completely covers $c$. If $b \cap c = \emptyset$, then at most $k-1$ balls of $\mathcal{B}$ intersect $c$, and since $Q(c)$ is a multiplicative $\varepsilon$-coreset of $P \cap c$ with respect to $(k-1)$-center, expanding each of the $k-1$ balls by a factor of $\varepsilon$ will cover the whole points in $c$ by induction.

Now, we show that our algorithm is core-preserving. Consider two arbitrary point sets $R$ and $S$ in $\mathbb{R}^d$, and let $Q$ be a multiplicative $\varepsilon$-coreset of $S$ computed by our algorithm. We prove that for any input of the form $P = R \cup Q$, the algorithm returns a multiplicative $\varepsilon$-coreset of $R \cup S$.

Our algorithm computes the coreset in $k$ layers. In the topmost layer (layer $k$), an additive $\varepsilon$-coreset for $k$-center is computed, and in any other layer $j$ ($1 \leq j < k$), the algorithm computes $O(\Pi_{i=j+1}^{k} i/\varepsilon^d)$ additive $\varepsilon$-coresets with respect to $j$-center, each of size $O(j/\varepsilon^d)$. Let $\mathcal{Q}_j(P)$ denote the union of all additive $\varepsilon$-coresets computed in the $j$th layer. Obviously, $\mathcal{Q}(P) = \cup_{j=1}^{k} \mathcal{Q}_j(P)$.

Let $r_j^*(P)$, $\tilde{r}_j(P)$, $\delta_j(P)$ be the $j$-center analogous of the notations defined before (if $P = \emptyset$, we define $\delta_j(P)$ to be 0). Let $\delta = \delta_k(P)$, $\sigma = \delta_k(S)$, and $\rho = \max\{\delta, \sigma\}$. As shown in Lemma 5.2, $X_\rho(R \cup S)$ is an additive $\varepsilon$-coreset of $R \cup S$, and $X_\rho(R \cup S) \subseteq X_\delta(R \cup Q) = \mathcal{Q}_k(P)$. Suppose w.l.o.g. that $\delta \geq \sigma$ (the other case is analogous). Suppose that there are $n$ non-empty cells $c_1, \ldots, c_n$ in $\mathcal{G}_\delta$. Fix one of these cells, say $c_i$. There are $m = (\rho/\sigma)^d$ cells of side length $\sigma$ in $c_i$ which we denote by $c_{i1}, \ldots, c_{im}$. Define $\delta_i = \delta_{k-1}(P \cap c_i)$, and $\sigma_{ij} = \delta_{k-1}(S \cap c_{ij})$ for $1 \leq j \leq m$ (see Figure 5.2).

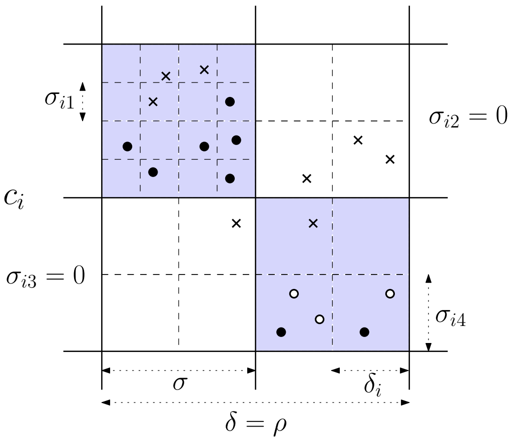

Figure 5.2: Multiplicative coreset for $k$-center. The points of $Q$, $S \backslash Q$, and $R$ are shown in black, white, and crossed, respectively.

Let $\rho_i = \max\{\delta_i, \sigma_{i1}, \ldots, \sigma_{im}\}$. It is easy to verify that $X_{\rho_i}((R \cup S) \cap c_i)$ is an additive $\varepsilon$-coreset of $(R \cup S) \cap c_i$. Moreover, we can show that $X_{\rho_i}((R \cup S) \cap c_i) \subseteq X_{\delta_i}((R \cup Q) \cap c_i) = X_{\delta_i}(P \cap c_i)$, by an argument similar to what is used in Lemma 5.2. Let $\mathcal{Q}'_{k-1}(R \cup S) = \cup_{i=1}^{n} X_{\rho_i}((R \cup S) \cap c_i)$. Since $\mathcal{Q}_{k-1}(P) = \cup_{i=1}^{n} X_{\delta_i}(P \cap c_i)$, we have $\mathcal{Q}'_{k-1}(R \cup S) \subseteq \mathcal{Q}_{k-1}(P)$. We define $\mathcal{Q}'_j(R \cup S)$ in a similar manner for all $1 \leq j \leq k$, and set $\mathcal{Q}'(R \cup S) = \cup_{j=1}^{k} \mathcal{Q}'_j(R \cup S)$. Obviously, $\mathcal{Q}'(R \cup S)$ is a multiplicative $\varepsilon$-coreset of $R \cup S$. Moreover, $\mathcal{Q}'(R \cup S) \subseteq \mathcal{Q}(P)$, which completes the proof.

Clearly, the resulting coreset has size $O(k!/\varepsilon^{kd})$. In each level $j$, we need to compute a 2-approximation of the $j$-center clustering for the points lying in each non-empty cell separately. Since the total number of points in each layer is $|P|$, and the approximation algorithm used is linear, the total time needed for this step in each layer is $O(|P|)$. Therefore, the total running time is $O(k|P| + k!/\varepsilon^{kd})$. $\square$

Note that in the above construction, $\mathcal{Q}_i(P)$ is a subset of $\mathcal{Q}_{i-1}(P)$ for all $1 < i \leq k$. Therefore, we can just extract the additive $\varepsilon$-coresets in layer 1, and discard any other additive coresets computed in the intermediate layers.

**Theorem 5.5** *Given a stream of points $P$ in $\mathbb{R}^d$, a multiplicative $\varepsilon$-coreset for the $k$-center problem can be maintained using $O(k!/\varepsilon^{kd})$ total space and $O(k)$ amortized time per update.*

*Proof.* This is a direct corollary of Lemma 5.4 and Theorem 5.1. $\square$

**Remark.** Some researchers have recently considered a variant of the $k$-clustering problem in which the objective is to minimize the sum of the clusters radii [27, 49]. Theorem 5.5 immediately yields an efficient streaming algorithm to maintain an $\varepsilon$-coreset with respect to this clustering problem as well.

## 5.5 Coreset for Fat Extent-Related Problems

Given a point set $P \subseteq \mathbb{R}^d$, let $\mathbb{B}(P)$ denote the minimum axis-parallel hyperbox enclosing $P$. We denote by $\ell(P)$ the length of the longest side of $\mathbb{B}(P)$. A subset $Q \subseteq P$ is called an *additive $\varepsilon$-kernel of $P$*, if for all $u \in \mathbb{S}^{d-1}$,

$$w(Q, u) \geq w(P, u) - \varepsilon\ell(P),$$

where $w(P, u) = \max_{p,q \in P} \langle p - q, u\rangle$.

A function $\mu(\cdot)$ defined over subsets of $\mathbb{R}^d$ is called a *fat measure*, if there exists a constant $\alpha > 0$ such that for any additive $\varepsilon$-kernel $Q$ of $P$, $\alpha\mu(P) \leq \mu(Q) \leq \mu(P)$. Examples of fat measures are diameter, radius of the minimum enclosing ball, and width of the smallest enclosing hypercube. Obviously, if $Q$ is an additive $\varepsilon$-kernel of $P$ and $\mu$ is a fat measure, then $Q$ is an $(\varepsilon/\alpha)$-coreset of $P$ with respect to $\mu$.

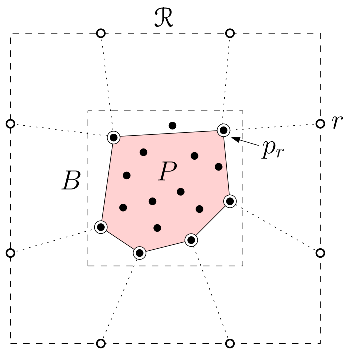

Figure 5.3: Construction of an additive $\varepsilon$-kernel.

Given a point set $P \subseteq \mathbb{R}^d$, an additive $\varepsilon$-kernel of $P$ can be computed efficiently using an adaptation of the simple grid-rounding method proposed in [21, 97] based on Dudley's construction [34]. The algorithm is described in the following lemma.

**Lemma 5.6** *There is a core-preserving algorithm that for every point set $P \subseteq \mathbb{R}^d$, computes an additive $\varepsilon$-kernel of $P$ of size $O(1/\varepsilon^{(d-1)/2})$ in $O(|P| + 1/\varepsilon^{d-(3/2)})$ time for $d \geq 2$, or in $O((|P| + 1/\varepsilon^{d-2}) \log(1/\varepsilon))$ time for $d \geq 3$.*

*Proof.* We assume w.l.o.g. that conv$(P)$ contains the origin. Let $\mathcal{B}(P)$ be the smallest hypercube centered at the origin containing $P$. If $\ell'(P)$ denotes the side length of $\mathcal{B}(P)$, then we have $\ell(P) \leq \ell'(P) \leq 2\ell(P)$.

Let $B = \mathcal{B}(P)$. By a simple scaling, we may assume that $B = [-1,1]^d$. Let $\mathcal{R}$ be the set of points of a $\sqrt{\varepsilon}$-grid over the boundary of the cube $[-2,2]^d$, and let $p_r$ denote the nearest neighbor of a point $r \in \mathcal{R}$ in the set $P$ (see Figure 5.3). Let $\mathcal{Q} = \{p_r \mid r \in \mathcal{R}\}$. Obviously, $|\mathcal{Q}| \leq |\mathcal{R}| = O(1/\varepsilon^{(d-1)/2})$. Moreover, $\mathcal{Q}$ is an additive $\varepsilon$-kernel of $P$ with the argument provided below. The running time follows immediately from the fast implementation of Chan using the discrete nearest neighbor queries [21].

Consider two arbitrary point sets $R$ and $S$ in $\mathbb{R}^d$, and let $Q$ be an additive $\varepsilon$-kernel of $S$ computed by our algorithm. In order for our algorithm to be core-preserving, we need to show that for any input of the form $P = R \cup Q$, the algorithm returns an additive $\varepsilon$-kernel of $R \cup S$.

We adapt the proof from [21]. Fix a unit vector $u \in \mathbb{S}^{d-1}$ and a point $p \in R \cup S$. There is a point $r \in \mathcal{R}$ such that $\angle(r - p, u) \leq \arccos(1 - \varepsilon/8)$ (See [21], Observation 2.3). If $p_r \in S$, then by our construction there is a point $q \in Q$ such that $\|r - q\| \leq (1 + c\varepsilon)\|r - p_r\|$. If $p_r \in R$, we simply set $q = p_r$. Therefore,

$$\begin{aligned}
& \|r - q\| \leq (1 + c\varepsilon)\|r - p\| \\
\Rightarrow\ & (1 - \varepsilon/8)\,\langle r - q, u\rangle \leq (1 + c\varepsilon)\,\langle r - p, u\rangle \\
\Rightarrow\ & \langle r - q, u\rangle - 3\sqrt{d}\varepsilon/8 \leq \langle r - p, u\rangle + 3c\sqrt{d}\varepsilon \\
& \quad \text{(since } \|r - p\| \text{ and } \|r - q\| \text{ are at most } 3\sqrt{d}) \\
\Rightarrow\ & \langle p, u\rangle \leq \langle q, u\rangle + 3\sqrt{d}(c + 1/8).
\end{aligned}$$

It means that the projections of $p$ and $q$ in direction $u$ differ by at most $O(\varepsilon)$. Since $\ell(P) \geq 1/2$, we conclude that $\langle p - q, u\rangle = O(\varepsilon)\ell(P)$ in every direction $u$, which completes the proof. □

Combining Lemma 5.6 with Theorem 5.1, we get the following result:

**Theorem 5.7** *Given a stream of points $P$ in $\mathbb{R}^d$ and a fat measure $\mu$, an $\varepsilon$-coreset of $P$ with respect to $\mu$ can be maintained using $O(1/\varepsilon^{(d-1)/2})$ total space and $\max\{O(1), O((1/\varepsilon^{(d-3)/2})\log(1/\varepsilon))\}$ amortized time per update.*

Theorem 5.7 surprisingly shows that the natural approach taken by previous streaming algorithms [21, 12] to maintain coresets in different directions is not necessarily the best for fat measures. For example, the best previous streaming algorithm for diamater and minimum enclosing ball in the plane due to Agarwal and Yu [12] requires $O(\sqrt{1/\varepsilon})$ space and $O(\log(1/\varepsilon))$ update time, while Theorem 5.7 implies an algorithm with the same space complexity and only $O(1)$ update time. Theorem 5.7 also improves our previous result in Theorem 4.7 by removing the extra $\log(1/\varepsilon)$ factor from the space complexity of the algorithm (i.e., acheiving $O(1/\varepsilon^{(d-1)/2})$ space) without increasing the update time for fat extent-related problems.

## 5.6 Conclusions

In this chapter, we have introduced the notion of core-preserving algorithms, and presented a general framework based on this notion to efficiently maintain $\varepsilon$-coresets for a number of fundamental geometric problems, including $k$-center (in both additive and multiplicative forms), diameter, and minimum enclosing ball.

We believe that our framework will easily find applications in other geometric (and non-geometric) problems. It can be also used to simplify some of the previously known results. For example, Agarwal and Yu [12] have shown that an $\varepsilon$-kernel of a stream of points within a fixed two-dimensional slab (which they called an *epoch*) can be maintained using $O(\sqrt{1/\varepsilon})$ space and $O(\log(1/\varepsilon))$ update time. They have employed an involved compression technique along with a balanced binary search tree to achieve this result. However, observing that the simple grid-rounding method described in Lemma 5.6 is core-preserving within a fixed slab, one can immediately get an even better result using $O(\sqrt{1/\varepsilon})$ space and an improved $O(1)$ update time, without using any extra data structure.

# Chapter 6

# Minimum Enclosing Ball in High Dimensions

As shown in the previous two chapters, a $(1+\varepsilon)$-approximation to the minimum enclosing ball can be maintained efficiently in fixed dimensions. However, the algorithms presented in those sectoins are not suitable in high-dimensional data streams, due to exponential dependency of their space complexity to the dimension. In this chapter, we analyze an extremely simple approximation algorithm for computing the minimum enclosing ball (or the 1-center) of a set of points in high dimensions. We prove that this algorithm computes a 3/2-factor approximation in any dimension using a minimum amount of space.

## 6.1 Introduction

The minimum enclosing ball problem, which is equivalent to Euclidean 1-center, is the problem of covering a set of $n$ points in $d$ dimensions using the smallest ball possible. This is one of the most fundamental and well-known problems in computational geometry, having many applications.

The 2-dimensional version of the problem was originally posed in 1857 by Sylvester [92] who proposed the first algorithm for computing the smallest circle enclosing a finite set of points in the plane [93]. Megiddo [81] showed that in any

An extended abstract of this chapter has been published as: H. Zarrabi-Zadeh and T. M. Chan. A simple streaming algorithm for minimum enclosing balls. In *Proc. 18th Canadian Conference on Computational Geometry*, pages 139–142, 2006.

fixed dimension, the Euclidean 1-center problem can be solved in linear time using linear programming techniques. Dyer [35] extended this result to the weighted 1-center problem. Sharir and Welzl [90, 95] showed that in any dimension $d$ the minimum enclosing ball problem is an LP-type problem with combinatorial dimension $d+1$, and can thus be solved exactly in $O(d^{O(d)}n)$ deterministic time [28], or $O(d^2 n + 2^{O(\sqrt{d \log d})})$ expected time [30, 79].

In this chapter, we focus on the high-dimensional version of the problem, where $d$ may be arbitrarily large. We are interested in faster approximation algorithms that avoid the exponential (or superpolynomial) dependency on the dimension. Furthermore, we are interested in algorithms that operate under the data stream model, where only one pass over the input is allowed and the algorithm has only a limited amount of working storage. We assume here that one unit of space can hold one coordinate of a point.

In high dimensions, several approximation algorithms have been proposed based on the notion of coresets. Bădoiu *et al.* [18] showed that for any point set in $d$ dimensions there exists a coreset of size $O(1/\varepsilon^2)$ (whose size is independent of the dimension) such that the minimum enclosing ball of the coreset is a $(1+\varepsilon)$-approximation to the minimum enclosing ball of the original point set. Bădoiu and Clarkson [17] (and independently, Kumar *et al.* [72]) improved the size of the coreset to $O(1/\varepsilon)$, leading to a $(1+\varepsilon)$-approximation algorithm for the minimum enclosing ball problem in $O(nd/\varepsilon + (1/\varepsilon)^5)$ time.

Although Bădoiu and Clarkson's algorithm requires only $O(1/\varepsilon)$ working space, when viewed in the streaming model, it requires more than one pass—specifically, $\lceil 2/\varepsilon \rceil$ passes. In fixed dimensions, there is a simple (one-pass) streaming algorithm that computes a $(1+\varepsilon)$-approximation to the minimum enclosing ball using $O(1/\varepsilon^{(d-1)/2})$ space (in $O(1/\varepsilon^{(d-1)/2})$ time). Namely, the algorithm just keeps track of the extreme points along $O(1/\varepsilon^{(d-1)/2})$ directions.

In high dimensions, there is a trivial 2-approximation streaming algorithm for the simpler diameter problem, using $O(d)$ space. For this problem, Indyk [66] has improved the approximation factor to any constant $c > \sqrt{2}$, using $O(dn^{1/(c^2-1)} \log n)$ space with high probability. His technique does not seem to yield a result for minimum enclosing ball though; in any case, a "constant" space bound independent of $n$ would be much more desirable than sublinear space. For the minimum enclosing cylinder problem in high dimensions, Chan [21] has given a streaming algorithm with any fixed approximation factor $c > 5$ using $O(d)$ space.

For minimum enclosing ball in high-dimensional data streams, the only known result we are aware of is the trivial 2-approximation algorithm that picks an arbitrary input point $c$ as the center of the ball and sets the radius of the ball to be the distance of the farthest point from $c$. In this chapter, we show how to improve this constant factor. We analyze a simple one-pass streaming algorithm for minimum enclosing ball and prove that it achieves approximation factor 3/2. Our algorithm uses $O(d)$ time per point and just $O(d)$ space. In fact, it uses the "minimum" amount of space possible—at any time, it maintains a single ball, and nothing else. Our algorithm is arguably the simplest algorithm that uses the minimum amount of space.

## 6.2 A Simple Streaming Algorithm

Here is our algorithm in its entirety, which returns a ball $B$ enclosing $P$:

```
1:  $B \leftarrow \emptyset$
2:  for each point $p$ in the input stream $P$ do
3:      if $p$ is outside $B$ then
4:          $B \leftarrow$ the smallest ball enclosing $B$ and $p$
```

Despite the utter simplicity of this algorithm, the analysis of its approximation factor seems nonobvious and, to our knowledge, has not been studied before.

To aid in the analysis (and implementation), we mention exactly how the coordinates of the ball $B$ can be calculated in line 4. Let $p_0$ be the first input point, $p_i$ be the input point causing the $i$-th update to $B$, and let $B_i$ be the value of $B$ after its $i$-th update (see Figure 6.1). We denote the center point and the radius of $B_i$ by $c_i$ and $r_i$, respectively.

Initially, we set $r_0 = 0$ and $c_0 = p_0$. Letting $\delta_i = \frac{1}{2}(\|p_i - c_{i-1}\| - r_{i-1})$ (i.e., half the distance of $p_i$ to $B_{i-1}$), we have

$$r_i = r_{i-1} + \delta_i \quad \text{and} \quad \|c_i - c_{i-1}\| = \delta_i.$$

As $c_i$ lies on the line segment from $c_{i-1}$ to $p_i$, the second equation implies that $c_i = c_{i-1} + \frac{\delta_i}{\|p_i - c_{i-1}\|}(p_i - c_{i-1})$.

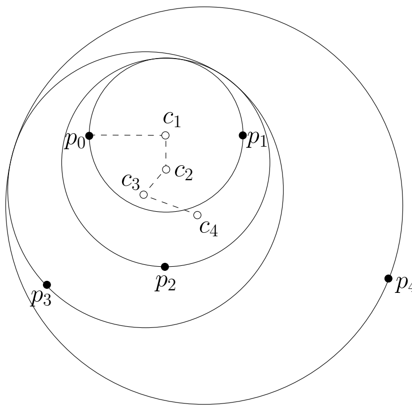

Figure 6.1: An example of the execution of the MEB algorithm.

*Remark.* The above equations imply the following interesting property concerning the trajectory of the center points $c_0, \ldots, c_i$ (the path shown in dashed lines in Figure 6.1): the total length of this trajectory is exactly equal to the last radius $r_i$.

### 6.2.1 Analysis

In this section, we prove an upper bound of 3/2 on the approximation factor of our algorithm. The proof is not long but is tricky and involves a nice invariant. We first recall one known geometric fact.

**Lemma 6.1** *If two chords intersect inside a circle, then the product of the segments of one chord equals the product of the segments of the other chord.*

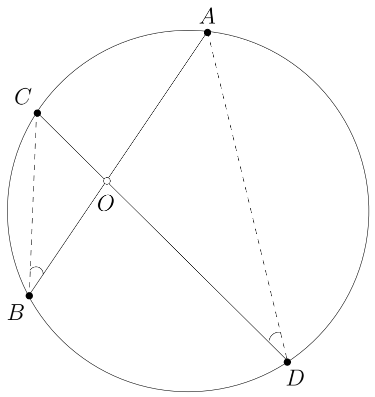

Figure 6.2: On Lemma 6.1.

*Proof.* Let $O$ be the intersection point of the two chords $AB$ and $CD$ (see Figure 6.2). The two angles $\angle ADC$ and $\angle ABC$ are equal, since both are inscribed angles of the same arc $AC$. Moreover, $\angle AOD = \angle BOC$. Therefore, the two triangles $\triangle AOD$ and $\triangle BOC$ are similar. Therefore $\frac{|AO|}{|CO|} = \frac{|DO|}{|BO|}$, and hence $|AO||BO| = |CO||DO|$. $\square$

**Theorem 6.2** *Given a set $P$ of $n$ points in $d$ dimensions, the algorithm in Section 6.2 computes a 3/2-approximation to the minimum enclosing ball of $P$ in $O(dn)$ time and $O(d)$ space.*

*Proof.* Let $B^*$ be the minimum enclosing ball of $P$, of radius $r^*$. Let $p_i$, $c_i$, and $r_i$ be as defined in Section 6.2. The $p_i$'s lie inside $B^*$, and so are the $c_i$'s (as one can see easily by induction, due to the convexity of $B^*$). For each $i > 0$, consider the chord of $B^*$ which passes through $c_{i-1}$ and $p_i$ (which passes through $c_i$ as well). The point $c_i$ splits this chord into two segments, one containing $c_{i-1}$ and the other containing $p_i$. Let $a_i$ be the length of the former segment and $b_i$ be the length of the latter segment (see Figure 6.3). The key to the whole proof lies in finding the right invariant, which turns out to be the following:

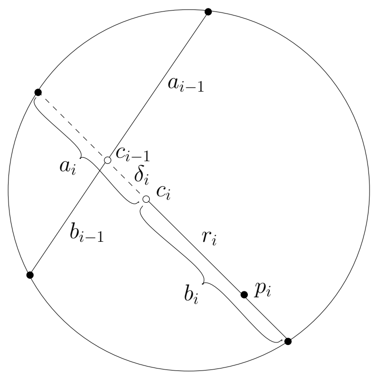

Figure 6.3: From step $i-1$ to step $i$.

**Claim** $r_i^2 < 3a_ib_i$ *for all $i > 0$.*

*Proof.* We prove by induction on $i$. The base case follows immediately because $a_1, b_1 \geq r_1$ (since $c_1$ is the midpoint of $p_0$ and $p_1$, which are both in $B^*$).

Now, suppose that $r_{i-1}^2 < 3a_{i-1}b_{i-1}$. By applying Lemma 6.1 (to the intersection of $B^*$ with the plane through $c_{i-1}$, $p_{i-1}$, and $p_i$), we have

$$a_{i-1}b_{i-1} = (a_i - \delta_i)(b_i + \delta_i). \tag{6.1}$$

A chain of algebraic manipulations then yields the claim:

$$
\begin{aligned}
3a_ib_i &= 3\left(\frac{a_{i-1}b_{i-1}}{b_i+\delta_i}+\delta_i\right)b_i \qquad \text{(by (6.1))}\\
&> \left(\frac{r_{i-1}^2}{b_i+\delta_i}+3\delta_i\right)b_i \qquad \text{(by hypothesis)}\\
&\geq \left(\frac{r_{i-1}^2}{r_i+\delta_i}+3\delta_i\right)r_i\\
&\text{(as } b_i \geq r_i \text{ and } x/(x+\delta_i) \text{ is increasing for } x \geq 0)\\
&= \left(\frac{(r_i-\delta_i)^2}{r_i+\delta_i}+3\delta_i\right)r_i\\
&= \left(\frac{r_i^2+\delta_ir_i+4\delta_i^2}{r_i+\delta_i}\right)r_i\\
&> r_i^2.
\end{aligned}
$$

□

The proof of the theorem is now straightforward. W.l.o.g., assume that $r_i \geq r^*$. Clearly, $a_i + b_i \leq 2r^*$. So,

$$r_i^2 < 3a_ib_i \leq 3(2r^* - b_i)b_i \leq 3(2r^* - r_i)r_i,$$

as $b_i \geq r_i$ and $(2r^* - x)x$ is decreasing for $x \geq r^*$. Thus $r_i < 3(2r^* - r_i)$, which means that $r_i < \frac{3}{2}r^*$ for all $i > 0$. □

### 6.2.2 Lower Bound

In this section we show that the analysis from Section 6.2.1 is essentially tight by providing a lower bound example for which our algorithm produces an enclosing ball with a radius close to 3/2 times the radius of the minimum enclosing ball.

Our example composed of $n$ points equally spaced on the boundary of a unit circle. Obviously, the radius of the minimum enclosing ball for this point set is 1. As $n$ goes to infinity, the trajectory path produced by the algorithm approaches a curve (like what is shown in Figure 6.4 for $n = 6$ and $n = 12$), whose length can be described using some differential integral equation. This curve seems interesting in its own right, though we are unable to find any previous work or information about it. The resulting function is quite complicated and solving it seems to be very hard. We have numerically calculated the length of this curve (as a discrete

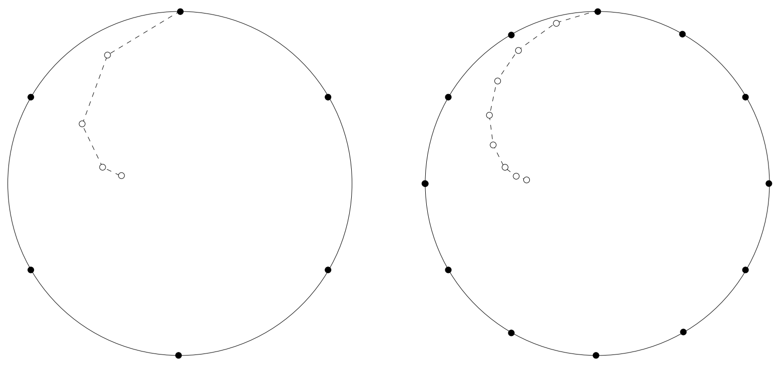

Figure 6.4: A lower bound example and an interesting curve.

| point set size | trajectory length |
|---|---|
| 10 | 1.39426 |
| 100 | 1.48955 |
| 1000 | 1.49895 |
| 10000 | 1.49989 |
| 100000 | 1.49998 |

Table 6.1: The lower bound example for the MEB algorithm.

path) for different values of $n$. The numerical results, presented in Table 6.1, shows that for large values of $n$, the length of the path becomes very close to 1.5.

We can also prove the following lower bound on the approximation factor of any algorithm that uses the same amount of space as our algorithm does.

**Theorem 6.3** *There is a lower bound of* $(1+\sqrt{2})/2 \approx 1.207$ *on the approximation factor of any deterministic algorithm for the minimum enclosing ball problem that at any time stores only one enclosing ball and nothing more.*

*Proof.* Let $\mathcal{A}$ be such an approximation algorithm. The adversary gives a sequence of points in the plane. Let $p_1 = (0,1)$ and $p_2 = (0,-1)$ be the first two points provided by the adversary, and let $B$ be the ball produced by $\mathcal{A}$ to enclose $\{p_1, p_2\}$. It is clear that $B$ contains at least one of the points $q_1 = (-1,0)$ and $q_2 = (1,0)$. Suppose w.l.o.g. that $B$ contains $q_1$. The adversary then gives the point $p_3 = (1+\sqrt{2}, 0)$. Now, $\mathcal{A}$ knows that all points given so far are enclosed by $B$, but for all points enclosed by $B$, it cannot remember which ones (in particular, $q_1 \in B$) have been parts of the input. Therefore, the updated ball after receiving $p_3$ must also enclose $q_1$. So, the radius of the ball must be at least $\frac{1}{2}||p_3 - q_1|| = \frac{1}{2}(2+\sqrt{2})$. However, the point set $\{p_1, p_2, p_3\}$ can be optimally enclosed by a ball of radius $\sqrt{2}$. Hence, the approximation factor of $\mathcal{A}$ is at least $\frac{1}{2\sqrt{2}}(2+\sqrt{2}) = \frac{1}{2}(1+\sqrt{2})$. □

## 6.3 Conclusions

In this chapter, we have provided a very simple (and thus highly practical) one-pass algorithm that computes a 3/2-approximation to the minimum enclosing ball of a set of points in any dimension. Our algorithm is simple enough to easily replace the well-known naïve 2-approximation algorithm, which is the only previously known one-pass algorithm working in high dimensions. Our algorithm can be regarded as a solution to a restricted *online* version of the minimum enclosing ball problem; Section 6.2.1 can be viewed as a *competitive analysis* of an online algorithm.

Many open problems remain. For example, is 3/2 the best possible approximation factor among one-pass algorithms that use the "minimum" amount of space (i.e., can Theorem 6.3 be strengthened)? What is the best approximation factor among one-pass algorithms that use $O(d)$ space, or more generally, $d^{O(1)}$ space? Can we get still better factors for minimum enclosing ball using sublinear ($o(n)$) space, perhaps by adapting Indyk's technique for the diameter problem [66]?

The same questions can also be asked for algorithms with two passes, three passes, and so on. For example, we can consider the following two-pass strategy that invokes our algorithm twice. Let $c_f$ and $r_f$ be the center and the radius of the enclosing ball after terminating the first pass. We then use the same algorithm for the second pass, except that we start with a ball centered at $c_f$ with radius $\frac{1}{3}r_f$. Experimental results suggest that the approximation factor obtained by this two-pass algorithm is at most 1.37, at least for the bad example from Section 6.2.2.

# Chapter 7

# Concluding Remarks

In this thesis, we have proposed new algorithms in the online and data stream models of computation for several fundamental optimization problems in computational geometry ranging from clustering to shape fitting.

In the online setting, we have shown that determining the best competitive ratio for the basic unit clustering problem is nontrivial even in the simplest one-dimensional case. We used randomization to beat the naïve upper bound of 2 on the one-dimensional problem and complement our results with some lower bounds. The obvious open problem is to close the gap between the current upper and lower bounds shown in Table 3.1 (some improvements have been recently announced in [41]). Another major problem for future research is to develop better algorithms for online unit clustering in two and higher dimensions. Apparently, our one-dimensional results can be extended to higher dimensions using a "dimension-reduction" approach. However, we wonder if ideas that are more "geometric" may lead to still better results than Theorem 2.17. Our work certainly raises countless questions concerning the best competitive ratio in higher-dimensional cases, for other metrics besides $L_\infty$, and for other geometric measures of cluster sizes besides radius or diameter.

In the data stream model, we have obtained better streaming algorithms for a number of fundamental geometric optimization problems in fixed dimensions, including width, smallest enclosing box, minimum enclosing cylinder, minimum-width enclosing annulus, etc. The main idea behind all these improvements is a new improved algorithm for maintaining an $\varepsilon$-kernel of a point set in fixed dimensions using a data structure whose size is almost optimal. More precisely, our algorithm uses $O(1/\varepsilon^{(d-1)/2}\log(1/\varepsilon))$ space, while the optimal space required is known to be $\Omega(1/\varepsilon^{(d-1)/2})$. We leave this question open whether this small log factor can be removed to achieve the optimal space in any fixed dimension (this question has been answered in affirmative by Agarwal and Yu [12] in two-dimensions only). Another possibility is to reduce the update time while keeping the space (near) optimal. It is also interesting to see if our approach can be extended to other related models like sliding window [23, 44] and dynamic streaming model [67, 47, 46].

We have also defined the class of core-preserving algorithms, and have shown that algorithms in this class can be efficiently dynamized in the data stream model. Our framework leads to improved streaming algorithms for a number of geometric problems in fixed dimensions such as diameter, minimum enclosing ball, and k-center (in both additive and multiplicative forms). In particular, we showed that $\varepsilon$-coresets for diameter and minimum enclosing ball in the plane can be maintained in $O(\sqrt{1/\varepsilon})$ space and $O(1)$ update time. An intriguing question is whether such a result can be obtained for the width problem as well. The current best record for this problem in 2-d is the algorithm of Agarwal and Yu [12] that requires optimal $O(\sqrt{1/\varepsilon})$ space and $O(\log(1/\varepsilon))$ update time. It is also interesting to explore more application of our framework to other geometric (and non-geometric) problems.

In high-dimensional data streams, we have proposed a simple streaming algorithm that computes a 3/2-approximation to the minimum enclosing ball in any dimension using only $O(d)$ space. There are a quite few streaming results for other geometric problems in high dimensions, including a $(\sqrt{2}+\varepsilon)$-approximation algorithm for diameter [66], a $(2+\varepsilon)$-approximation for $k$-center [80], and a $(5+\varepsilon)$-approximation for minimum enclosing cylinder [21]. An interesting thread for future research is to explore the existence of (better) streaming algorithms for other basic geometric problems such as width, minimum bounding box, etc.

Many interesting problems arise when we relax the one-pass restriction slightly, and allow algorithm to have a few passes over the input. For the minimum enclosing ball problem, for instance, our work shows that Bădoiu and Clarkson's $\lceil 2/\varepsilon \rceil$-pass algorithm [17] is not necessarily the best possible result. Similar questions are open even for the diameter problem. For example, there is a simple two-pass diameter algorithm [36] that gives $\sqrt{3}$ factor with $O(d)$ space. We do not know if this is the best two-pass algorithm with $O(d)$ space, or what is the best three-pass algorithm.

# Bibliography

[1] U. Adamy and T. Erlebach. Online coloring of intervals with bandwidth. In *Proceedings of the 1st Workshop on Approximation and Online Algorithms*, volume 2909 of *Lecture Notes in Computer Science*, pages 1–12, 2003. 13

[2] P. K. Agarwal, B. Aronov, and M. Sharir. Line transversals of balls and smallest enclosing cylinders in three dimensions. *Discrete and Computational Geometry*, 21(3):373–388, 1999. 5

[3] P. K. Agarwal and S. Har-Peled. Maintaining approximate extent measures of moving points. In *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms*, pages 148–157, 2001. 7

[4] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Approximating extent measures of points. *Journal of the ACM*, 51(4):606–635, 2004. 5, 7, 8, 41, 42, 43, 44, 54, 58, 61

[5] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Geometric approximation via coresets. *Combinatorial and Computational Geometry* (J. E. Goodman, J. Pach, and E. Welzl, eds.), Math. Sci. Research Inst. Pub., Cambridge, 2005. 6, 42

[6] P. K. Agarwal, S. Har-Peled, and H. Yu. Robust shape fitting via peeling and grating coresets. *Discrete and Computational Geometry*, 39(1-3):38–58, 2008. 7, 54

[7] P. K. Agarwal, J. Matoušek, and S. Suri. Farthest neighbors, maximum spanning trees and related problems in higher dimensions. *Computational Geometry: Theory and Applications*, 1(4):189–201, 1992. 5, 54

[8] P. K. Agarwal and C. M. Procopiuc. Exact and approximation algorithms for clustering. *Algorithmica*, 33(2):201–226, 2002. 61

[9] P. K. Agarwal, C. M. Procopiuc, and K. R. Varadarajan. Approximation algorithms for a $k$-line center. *Algorithmica*, 42(3-4):221–230, 2005. 6, 63

[10] P. K. Agarwal and M. Sharir. Efficient randomized algorithms for some geometric optimization problems. *Discrete and Computational Geometry*, 16(4):317–337, 1996. 5

[11] P. K. Agarwal, M. van Kreveld, and S. Suri. Label placement by maximum independent set in rectangles. *Computational Geometry: Theory and Applications*, 11:209–218, 1998. 13

[12] P. K. Agarwal and H. Yu. A space-optimal data-stream algorithm for coresets in the plane. In *Proceedings of the 23rd ACM Symposium on Computational Geometry*, pages 1–10, 2007. 7, 8, 42, 43, 47, 53, 54, 55, 58, 59, 67, 68, 78

[13] G. Barequet and S. Har-Peled. Efficiently approximating the minimum-volume bounding box of a point set in three dimensions. *Journal of Algorithms*, 38(1):91–109, 2001. 44

[14] J. L. Bentley and J. B. Saxe. Decomposable searching problems I: Static-to-dynamic transformations. *Journal of Algorithms*, 1:301–358, 1980. 42, 58

[15] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998. 2

[16] P. Brucker. On the complexity of clustering problems. In R. Henn, B. Korte, and W. Oletti, editors, *Optimizing and Operations Research.* Springer-Verlag, Berlin, West Germany, 1977. 3

[17] M. Bădoiu and K. L. Clarkson. Smaller core-sets for balls. In *Proceedings of the 14th ACM-SIAM Symposium on Discrete Algorithms*, pages 801–802, 2003. 7, 70, 78

[18] M. Bădoiu, S. Har-Peled, and P. Indyk. Approximate clustering via core-sets. In *Proceedings of the 34th ACM Symposium on Theory of Computing*, pages 250–257, 2002. 7, 70

[19] T. M. Chan. Approximating the diameter, width, smallest enclosing cylinder, and minimum-width annulus. *International Journal of Computational Geometry and Applications*, 12:67–85, 2002. 5

[20] T. M. Chan. Polynomial-time approximation schemes for packing and piercing fat objects. *Journal of Algorithms*, 46:178–189, 2003. 4, 13

[21] T. M. Chan. Faster core-set constructions and data stream algorithms in fixed dimensions. *Computational Geometry: Theory and Applications*, 35(1–2):20–35, 2006. 5, 7, 42, 43, 44, 46, 50, 53, 58, 66, 67, 70, 78

[22] T. M. Chan. Dynamic coresets. In *Proceedings of the 24th ACM Symposium on Computational Geometry*, pages 1–9, 2008. 7

[23] T. M. Chan and B. S. Sadjad. Geometric optimization problems over sliding windows. *International Journal of Computational Geometry and Applications*, 16(2-3):145–158, 2006. 3, 78

[24] T. M. Chan and H. Zarrabi-Zadeh. A randomized algorithm for online unit clustering. In *Proceedings of the 4th Workshop on Approximation and Online Algorithms*, volume 4368 of *Lecture Notes in Computer Science*, pages 121–131, 2006. To appear in *Theory of Computing Systems* (special issue of invited papers from WAOA 2006). 9, 11, 15

[25] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. *SIAM Journal on Computing*, 33(6):1417–1440, 2004. 3, 4, 12

[26] M. Charikar, L. O'Callaghan, and R. Panigrahy. Better streaming algorithms for clustering problems. In *Proceedings of the 35th ACM Symposium on Theory of Computing*, pages 30–39, 2003. 12

[27] M. Charikar and R. Panigrahy. Clustering to minimize the sum of cluster diameters. *Journal of Computer and Systems Sciences*, 68:417–441, 2004. 58, 65

[28] B. Chazelle and J. Matoušek. On linear-time deterministic algorithms for optimization problems in fixed dimension. *Journal of Algorithms*, 21(3):579–597, 1996. 70

[29] M. Chrobak and M. Ślusarek. On some packing problems relating to Dynamical Storage Allocation. *RAIRO Theoretical Informatics and Applications*, 22:487–499, 1988. 32

[30] K. L. Clarkson. Las Vegas algorithms for linear and integer programming. *Journal of the ACM*, 42(2):488–499, 1995. 70

[31] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete and Computational Geometry*, 4(1):387–421, 1989. 5

[32] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2nd edition, 2001. 13

[33] R. G. Downey and C. McCartin. Online problems, pathwidth, and persistence. In *Proceedings of the 1st International Workshop on Parameterized and Exact Computation*, pages 13–24, 2004. 32

[34] R. M. Dudley. Metric entropy of some classes of sets with differentiable boundaries. *Journal of Approximation Theory*, 10:227–236, 1974. 44, 66

[35] M. E. Dyer. On a multidimensional search technique and its application to the Euclidean one-centre problem. *SIAM Journal on Computing*, 15:725–738, 1986. 70

[36] O. Eg̃ecioḡlu and B. Kalantari. Approximating the diameter of a set of points in the Euclidean space. *Information Processing Letters*, 32:205–211, 1989. 78

[37] M. Edwards and K. Varadarajan. No coreset, no cry: Ii. In *Proceedings of the 25th Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 107–115, 2005. 7

[38] A. Efrat, M. J. Katz, F. Nielsen, and M. Sharir. Dynamic data structures for fat objects and their applications. *Computational Geometry: Theory and Applications*, 15:215–227, 2000. 4

[39] L. Epstein, A. Levin, and R. van Stee. Online unit clustering: variations on a theme. *Theoretical Computer Science*, to appear. 30

[40] L. Epstein and M. Levy. Online interval coloring and variants. In *Proceedings of the 32nd International Colloquium on Automata, Languages, and Programming*, pages 602–613, 2005. 13, 32

[41] L. Epstein and R. van Stee. On the online unit clustering problem. In *Proceedings of the 5th Workshop on Approximation and Online Algorithms*, volume 4927 of *Lecture Notes in Computer Science*, pages 193–206, 2007. 30, 39, 77

[42] T. Erlebach, K. Jansen, and E. Seidel. Polynomial-time approximation schemes for geometric intersection graphs. *SIAM Journal on Computing*, 34:1302–1323, 2005. 13

[43] T. Feder and D. H. Greene. Optimal algorithms for approximate clustering. In *Proceedings of the 20th ACM Symposium on Theory of Computing*, pages 434–444, 1988. 3, 4, 12

[44] J. Feigenbaum, S. Kannan, and J. Zhang. Computing diameter in the streaming and sliding-window models. *Algorithmica*, 41(1):25–41, 2004. 3, 78

[45] R. J. Fowler, M. S. Paterson, and S. L. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Information Processing Letters*, 12(3):133–137, 1981. 3, 4, 12

[46] G. Frahling, P. Indyk, and C. Sohler. Sampling in dynamic data streams and applications. *International Journal of Computational Geometry and Applications*, 18(1-2):3–28, 2008. 3, 78

[47] G. Frahling and C. Sohler. Coresets in dynamic geometric data streams. In *Proceedings of the 37th ACM Symposium on Theory of Computing*, pages 209–217, 2005. 3, 7, 58, 78

[48] M. Franceschetti, M. Cook, and J. Bruck. A geometric theorem for network design. *IEEE Transactions on Computers*, 53(4):483–489, 2004. 4

[49] M. Gibson, G. Kanade, E. Krohn, I. A. Pirwani, and K. Varadarajan. On clustering to minimize the sum of radii. In *Proceedings of the 19th ACM-SIAM Symposium on Discrete Algorithms*, pages 819–825, 2008. 58, 65

[50] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs.* Elsevier, 2nd edition, 2004. 32

[51] T. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985. 3, 12, 61, 63

[52] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *Proceedings of the 41st IEEE Symposium on Foundations of Computer Science*, pages 359–366, 2000. 12

[53] U. Gupta, D. T. Lee, and Y. T. Leung. Efficient algorithms for interval graphs and circular-arc graphs. *Networks*, 12(2):45967, 1982. 4

[54] A. Gyárfás and J. Lehel. On-line and First-Fit colorings of graphs. *Journal of Graph Theory*, 12:217–227, 1988. 32, 33

[55] M. M. Halldórsson. Improved performance guarantee for randomized on-line graph coloring. Technical Report 91-35, DIMACS, 1994. 32

[56] M. M. Halldórsson and M. Szegedy. Lower bounds for on-line graph coloring. *Theoretical Computer Science*, 130:163–174, 1994. 32

[57] S. Har-Peled. Clustering motion. *Discrete and Computational Geometry*, 31(4):545–565, 2004. 3, 4, 7, 58, 63

[58] S. Har-Peled. No coreset, no cry. In *Proceedings of the 24th Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 324–335, 2004. 4, 7, 61, 63

[59] S. Har-Peled and A. Kushal. Smaller coresets for $k$-means and $k$-median clustering. *Discrete and Computational Geometry*, 37(1):3–19, 2004. 7

[60] S. Har-Peled and S. Mazumdar. On coresets for $k$-means and $k$-median clustering. In *Proceedings of the 36th ACM Symposium on Theory of Computing*, pages 291–300, 2004. 7, 58

[61] S. Har-Peled and K. Varadarajan. Projective clustering in high dimensions using core-sets. In *Proceedings of the 18th ACM Symposium on Computational Geometry*, pages 312–318, 2002. 7

[62] S. Har-Peled and Y. Wang. Shape fitting with outliers. *SIAM Journal on Computing*, 33(2):269–285, 2004. 7, 54, 55

[63] D. S. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *Journal of the ACM*, 32:130–136, 1985. 4, 12

[64] D. S. Hochbaum and D. Shmoys. A best possible heuristic for the $k$-center problem. *Mathematics of Operational Research*, 10:180–184, 1985. 3

[65] D. S. Hochbaum and D. Shmoys. A unified approach to approximation algorithms for bottleneck problems. *Journal of the ACM*, 33:533–550, 1986. 3

[66] P. Indyk. Better algorithms for high-dimensional proximity problems via asymmetric embeddings. In *Proceedings of the 14th ACM-SIAM Symposium on Discrete Algorithms*, pages 539–545, 2003. 70, 76, 78

[67] P. Indyk. Algorithms for dynamic geometric problems over data streams. In *Proceedings of the 36th ACM Symposium on Theory of Computing*, pages 373–380, 2004. 3, 78

[68] S. Irani. Coloring inductive graphs on-line. *Algorithmica*, 11:53–72, 1994. 32

[69] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972. 32

[70] H. A. Kierstead and J. Qin. Coloring interval graphs with First-Fit. *SIAM Journal on Discrete Mathematics*, 8:47–57, 1995. 32, 33, 39

[71] H. A. Kierstead and W. T. Trotter. An extremal problem in recursive combinatorics. *Congressus Numerantium*, 33:143–153, 1981. 13, 32

[72] P. Kumar, J. S. B. Mitchell, and E. A. Yildirim. Approximate minimum enclosing balls in high dimensions using core-sets. *Journal of Experimental Algorithmics*, 8:1–29, 2003. 70

[73] S. Leonardi and A. Vitaletti. Randomized lower bounds for online path coloring. In *Proceedings of the 2nd International Workshop on Randomization and Computation*, volume 1518 of *Lecture Notes in Computer Science*, pages 232–247, 1998. 32

[74] R. J. Lipton and A. Tomkins. Online interval scheduling. In *Proceedings of the 5th ACM-SIAM Symposium on Discrete Algorithms*, pages 302–311, 1994. 13

[75] L. Lovász, M. Saks, and W. T. Trotter. An on-line graph coloring algorithm with sublinear performance ratio. *Discrete Mathematics*, 75:319–325, 1989. 32

[76] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM*, 41:960–981, 1994. 32

[77] M. V. Marathe, H. Breu, H. B. Hunt III, S. S. Ravi, and D. J. Rosenkrantz. Simple heuristics for unit disk graphs. *Networks*, 25:59–68, 1995. 13

[78] M. V. Marathe, H. B. Hunt III, and S. S. Ravi. Efficient approximation algorithms for domatic partition and on-line coloring of circular arc graphs. *Discrete Applied Mathematics*, 64:135–149, 1996. 32

[79] J. Matoušek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. *Algorithmica*, 16(4-5):498–516, 1996. 70

[80] R. M. McCutchen and S. Khuller. Streaming algorithms for k-center clustering with outliers and with anonymity. In *Proceedings of the 11th International Workshop on Approximation Algorithms*, 2008, to appear. 4, 78

[81] N. Megiddo. Linear-time algorithms for linear programming in $R^3$ and related problems. *SIAM Journal on Computing*, 12(4):759–776, 1983. 69

[82] N. Megiddo and K. J. Supowit. On the complexity of some common geometric location problems. *SIAM Journal on Computing*, 13(1):182–196, 1984. 3, 12

[83] N. Megiddo, A. Tamir, E. Zemel, and R. Chandrasekaran. An $O(n(\log n)^2)$ algorithm for the $k$-th nearest pair in a tree with applications to location problems. *SIAM Journal on Computing*, 10:328–337, 1981. 3

[84] S. M. Muthukrishnan. Data streams: Algorithms and applications. `http://athos.rutgers.edu/~muthu/stream-1-1.ps`, 2003. 2

[85] F. Nielsen. Fast stabbing of boxes in high dimensions. In *Proceedings of the 8th Canadian Conference on Computational Geometry*, pages 87–92, 1996. 4

[86] F. Nielsen. Fast stabbing of boxes in high dimensions. *Theoretical Computer Science*, 246:532, 2000. 4

[87] S. V. Pemmaraju, R. Raman, and K. Varadarajan. Buffer minimization using max-coloring. In *Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms*, pages 562–571, 2004. 32

[88] E. Ramos. Deterministic algorithms for 3-d dimaeter and some 2-d lower envelopes. In *Proceedings of the 16th ACM Symposium on Computational Geometry*, pages 290–299, 2000. 5

[89] C. A. Rogers. A note on coverings. *Mathematika*, 4:1–6, 1957. 4

[90] M. Sharir and E. Welzl. A combinatorial bound for linear programming and related problems. In *Proceedings of the 9th Symposium on Theoretical Aspects

*of Computer Science*, volume 577 of *Lecture Notes in Computer Science*, pages 569–579. Springer-Verlag, 1992. 70

[91] J. Snoeyink. Maximum independent set for intervals by divide-prune-and-conquer. In *Proceedings of the 17th Canadian Conference on Computational Geometry*, pages 264–265, 2005. 4

[92] J. J. Sylvester. A question on the geometry of situation. *Quarterly Journal of Mathematics*, 1:79, 1857. 69

[93] J. J. Sylvester. On Poncelet's approximate valuation of surd forms. *Philos. Magazine*, XX:203–222, 1860. 69

[94] S. Vishwanathan. Randomized on-line graph coloring. In *Proceedings of the 31st IEEE Symposium on Foundations of Computer Science*, pages 121–130, 1990. 32

[95] E. Welzl. Smallest enclosing disks (balls and ellipsoids). In *New Results and New Trends in Computer Science*, volume 555 of *Lecture Notes in Computer Science*, pages 359–370, 1991. 70

[96] A. C. C. Yao. Probabilistic computations: towards a unified measure of complexity. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science*, pages 222–227, 1977. 14, 36

[97] H. Yu, P. K. Agarwal, R. Poreddy, and K. R. Varadarajan. Practical methods for shape fitting and kinetic data structures using core sets. In *Proceedings of the 20th ACM Symposium on Computational Geometry*, pages 263–272, 2004. 44, 46, 66

[98] H. Zarrabi-Zadeh. Online coloring co-interval graphs. In *Proceedings of the 12th International CSI Computer Conference*, pages 1328–1332, 2007. 9

[99] H. Zarrabi-Zadeh. An almost space-optimal streaming algorithm for coresets in fixed dimensions. In *Proceedings of the 16th European Symposium on Algorithms*, volume 5193 of *Lecture Notes in Computer Science*, pages 817–829, 2008. Invited to *Algorithmica* (special issue of selected papers from ESA 2008). 9

[100] H. Zarrabi-Zadeh. Core-preserving algorithms. In *Proceedings of the 20th Canadian Conference on Computational Geometry*, pages 159–162, 2008. 9

[101] H. Zarrabi-Zadeh and T. M. Chan. A simple streaming algorithm for minimum enclosing balls. In *Proceedings of the 18th Canadian Conference on Computational Geometry*, pages 139–142, 2006. 9

[102] H. Zarrabi-Zadeh and T. M. Chan. An improved algorithm for online unit clustering. In *Proceedings of the 13th International Computing and Combinatorics Conference*, volume 4598 of *Lecture Notes in Computer Science*, pages 383–393, 2007. To appear in *Algorithmica* (special issue of invited papers from COCOON 2007). 9, 11