Geometric Optimization for Classification Problems

Pablo Pérez Lantero
2010

Universidad de Sevilla
Escuela Técnica Superior de Ingenieros
Dpto. Matemática Aplicada II

Geometric Optimization for Classification Problems

Memoria de tesis doctoral presentada por
Pablo Pérez Lantero para optar al grado de
Doctor en Matemáticas por la Universidad de Sevilla

Director: José Miguel Díaz-Báñez

Sevilla, Abril de 2010

Universidad de Sevilla

Memoria de tesis doctoral para optar al grado de
Doctor en Matemáticas

Autor: Pablo Pérez Lantero

Título: Geometric Optimization for Classification Problems

Departamento: Matemática Aplicada II

V^{o} B^{o} Director: _____
José Miguel Díaz-Báñez

El autor: _____
Pablo Pérez Lantero

A mis padres,
a Diana.

Acknowledgements / Agradecimientos

First of all, I would like to thank José Miguel Díaz-Báñez for being my advisor. Throughout these years, he has provided me with guidance in research, valuable knowledge, and financial support. I am indebted to him.

I give thanks to Inma Ventura for being an excellent partner. We spent a lot of time discussing problems, writing results, and reviewing papers.

I am grateful to have collaborated with Jorge Urrutia, Sergey Bereg, and Sergio Cabello. They let me share part of their wisdom and experience in Computational Geometry and Algorithms. I also give thanks to everyone else whom I have worked with in open problem workshops.

I would like to show my gratitude with the invitations of: Jorge Urrutia to the UNAM, Sergio Cabello to the University of Ljubljana, Clemens Huemer to the Universitat Politècnica de Catalunya, and Leslie Bajuelos to the University of Aveiro.

I acknowledge the projects MEC MTM2006-03909 and MEC MTM2009-08652 for offering me the problems of this thesis, and, of course, the collaboration project MAEC-AECID of Spain for giving me the scholarship.

Gracias a los miembros del Departamento Matemática Aplicada II de la Universidad de Sevilla por todo su apoyo, especialmente a Santigo, Soledad, Mari Carmen, Juan Antonio, Julio, Emilio, y por supuesto, Miguel. Gracias a la Universidad de La Habana por mi formación.

Gracias a toda mi familia y a todos mis amigos por el apoyo brindado. Estaré eternamente agradecido de mis padres por ser lo que son, excelentes padres, y sobre todo, por soportar la distancia y alentarme desde lejos. Agradezco a mi esposa, con todo el orgullo de poder hacerlo, por toda su entrega, amor, compañía, y el sustento que me ha dado durante mis estudios.

Contents

Abstract

Data Mining is a relevant discipline in Computer Science which its main goal is to explore data and extract information that is potentially useful and previously unknown. By using mathematical tools, such as Operation Research, Statistics, Artificial Intelligence and more recently Computational Geometry, Data Mining solves problems in many areas where there are big databases. Within Computational Geometry, the techniques of Geometric Optimization can be applied to solve many problems in this field. Typically, problems in Data Mining concern data belonging to two classes, say red and blue, and mainly appear in important subareas such as the classification of new data and the recognition of patterns.

This thesis focuses on the study of optimization problems with application in data classification and pattern recognition. In all of them, we are given a two-class data set represented as red and blue points in the plane, and the goal is to find simple geometrical shapes meeting some requirements for classification. The problems are approached from the Computational Geometry point of view, and efficient algorithms that use the inherent geometry of the problems are proposed.

A crucial problem in Data Mining is the so-called *Maximum Box Problem*, where the geometrical shape to be found is a maximum box, that is, an axis-aligned rectangle containing the maximum number of elements of only one class in the given data set. This thesis solves some natural variants of this basic problem: by considering two boxes (one per class), the minimum number of boxes to cover a class, or the maximum box problem for kinetic scenarios. Commonly, classification methods suppose a *good* data distribution, so a clustering procedure can be applied. However, if the classes are *well mixed*, a clustering for selecting prototypes that represent a class is not possible. In that sense, this thesis studies a new parameter to measure, a priori, if a given two-class data set is suitable or not for classification.

Resumen

La Minería de Datos es una importante disciplina dentro de la Ciencia de la Computación cuyo objetivo es la exploración de datos para extraer información potencialmente útil y previamente desconocida. Usando técnicas matemáticas como la Investigación Operativa, la Inteligencia Artificial, y más recientemente la Geometría Computacional, la Minería de Datos está resolviendo problemas en muchas áreas donde hay grandes bases de datos. En el marco de la Geometría Computacional, las técnicas de Optimización Geométrica pueden ser aplicadas en la resolución de muchos problemas en este campo. Comúnmente, los problemas en Minería de Datos están vinculados a datos de dos clases, rojos y azules, y aparecen en importantes subáreas como la clasificación de nuevos datos y el reconocimiento de patrones.

En esta tesis se estudian problemas de optimización con aplicaciones en clasificación de datos y reconocimiento de ratrones. En ellos tenemos datos de dos clases representados por puntos rojos y azules en el plano, y consisten en buscar objetos geométicos simples cumpliendo propiedades de optimización. Los problemas son enfocados desde la óptica de la Geometría Computacional, y se presentan algoritmos que usan su inherente geometría.

Un problema importante en la Minería de Datos es el de la *Caja Máxima*, donde el objeto a buscar es un rectángulo con lados paralelos a los ejes coordenados que contiene el máximo número de puntos de una sola clase. En esta tesis se resuelven algunas variantes de este problema considerando: dos cajas (una para cada clase), el mínimo número de cajas para cubrir una clase, o el mantenimiento de la máxima caja para puntos en movimiento. Comúnmente los metódos de clasificación suponen una *buena* distribución de los datos, de tal suerte que puede ser aplicado un procedimiento de agrupación. Sin embargo, si las clases están *bien mezcladas*, no es posible entonces obtener una agrupación para seleccionar luego prototipos que representen una clase. En este sentido, esta tesis estudia un nuevo parámetro para medir a priori si datos de dos clases son buenos o no para hacer clasificación.

Chapter 1

Introduction

The framework of this thesis is the application of the techniques from Computational Geometry, specifically from Geometric Optimization, in order to efficiently solve, from the mathematical and algorithmic point of view, problems that arise in application areas such as Data Mining and Classification. Most of these problems have a geometric modeling which will be used in the design of the algorithms.

This chapter presents a brief introduction and motivation for the optimization problems in Classification, followed by a short description of the main contributions developed in this thesis and an overview of the organization into chapters. It serves as an extended abstract for the following chapters.

1.1 Computational Geometry

Computational Geometry focuses on the design and analysis of efficient algorithms for problems whose mathematical representation is purely geometrical. Computing geometric properties of a set of geometric objects is a classic problem in Computational Geometry. For example, given a set of points in the plane compute their convex hull or the nearest pair of points, given a polygon to obtain one of its triangulations, etc. A deeper goal of Computational Geometry is the study of the computational complexity that is inherent to some geometric problems under different models of computation. For instance, to know if for a specific problem there exists an efficient computational solution. It is well known, for example, that given a set of points in the plane, there is no efficient algorithm capable of finding a tour that visits all points and minimizes the total traveled distance. The handbook of Preparata and Shamos [91] is considered as an introductory book

to Computational Geometry. A more recent handbook is the one by M. de Berg et al. [40]. We also cite the handbooks of Goodman and O'Rourke [58] and Sack and Urrutia [92], which are very useful to know what has been studied about any topic. Nowadays, Computational Geometry has a wide area of research with many specialized journals and important worldwide congress.

1.2 Geometric Optimization

Geometric optimization deals with problems of computing geometric objects which are optimal subject to certain criteria and geometric constraints. Typical examples include facility location, low-dimensional clustering, network design, optimal path planning, shape matching, proximity, and statistical measure problems. In such cases one expects that faster and simpler algorithms can be developed by exploiting the geometric nature of the problem. Much work has been done on geometric optimization problems during the last twenty-five years. Many elegant and sophisticated techniques have been proposed and successfully applied to a wide range of geometric optimization problems. See the paper of Agarwal and Sharir [3] for a comprehensive survey of these techniques.

1.3 Data Mining and Classification

Data Mining is a relevant discipline in Computer Science whose main goal is to explore data and extract information that is potentially useful and previously unknown. By using mathematical tools, such as Operation Research, Statistics, Artificial Intelligence, and more recently the Computational Geometry, Data Mining solves problems in many areas where there are big databases, which occasionally have erroneous data or do not contain a well defined structure due to out-of-range data. Data Mining has many applications, for example: in marketing for the segmentation of clients, in banks to evaluate the risk on credits given to clients and to detect credit cards fraud, in internet for the mining of web pages and to filter unwanted emails, and further examples.

One of the basic objectives in Data Mining is *the classification of new cases*, that is, to identify members between two different classes of data. For example, for the diagnosis of cancer, one should be able to discriminate if new laboratory results of a patient correspond to a benign tumor or to a malign tumor. Within the classification methods used in Data Mining,

two types are distinguished, the *unsupervised* and the *supervised* [28, 66]. In the unsupervised classification, the problem consists in partitioning the data into groups or clusters which are called "classes". In this way, a new data is classified according to the nearest cluster by considering a *function of similitude*. In the supervised classification, one is given the big database in which the class or type of each element is previously known. The aim is to create a method or rule that permits, given a new data, to decide which class it belongs to. The diagnosis of cancer is an example of supervised classification. We are given a big historical database of cases, each classified in benign or malign, and the classification rule says the type of a new tumor.

1.4 Overview, related work and motivation

In this section we give a brief review of the related literature and provide motivation for each of the problems studied in this thesis.

Suppose we have a database of analysis of a specific type of tumor, to be more precise, of a particular human body cells. Each data is a numerical vector of the same dimension, whose components represent the analysis results, for instance, the uniformity of cells shape, the variability of their size, etc. In turn, each vector belongs to one class of two possible: benign or malign. Now, the problem is to build a supervised classification rule, that is, to study the database and obtain a method which, when the results of a new analysis are given and represented in the corresponding numerical vector, determine with high precision in which class the new analysis is. A natural method to perform this classification is the selection of prototypes that represent data of different classes. The classification approach for a new data is to find the most adjusted prototype to it by using a certain function of distance.

A standard classification technique is the analysis of clusters in the training data [46], which are the database under study. Then, the prototypes are defined from the clusters. The database, since it consists of numerical vectors of a fixed dimension d, can be seen from the geometrical point of view as a set of points in the $\mathbb{R}^d$ space. That is why some kind of clusters can be obtained by using simple geometric objects such as lines, disks, balls, boxes, strips, etc. For example, Aronov and Har-Peled [10] and Eckstein et al. [48] considered disks and axis-aligned boxes, respectively. Henceforth, the training data will be referred to as points. The points of one class will be colored *red* and the points of the other class will be colored *blue*.

A condition on the distribution of the point set that is often handled with

care is its separability. The separation of the points is a natural way to obtain clusters. The simplest concept of separability is the *linear separability* that takes place by using a "line", that is, if there exists a hyperplane that leaves the red points to one of its sides and the blue points to the other one. Assuming that the set of points is linearly separable, a technique frequently applied is the *Support Vector Machine* (SVM) [66], that is a robust and tested methodology for problems of inference and prototype searching. SVM resides in finding a hyperplane that separates the red points from the blue points, and maximizes the minimum distance of the points to it (Figure 1.1). In this sense, the set of red points will form a cluster as will the set of blue points. Then, the classification method is very simple: if the new data is in the half-plane (half-space) where the red points are, then it is classified red, otherwise it is classified blue.

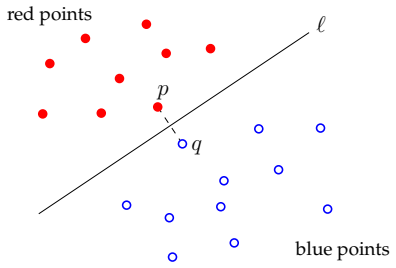


Figure 1.1: The line ℓ is the SVM. The minimum distance from the points to ℓ is maximized. So the nearest red point p to ℓ and the nearest blue point q to ℓ are equidistant from ℓ.

Another technique that can be applied to linearly separable point sets is the *Perceptron* [16]. The appearance of this model was motivated by biological concepts and is defined as follows: *Let $x \in \mathbb{R}^d$ be the point that is to be classified. A perceptron, with weights $w \in \mathbb{R}^d$ and margin $\gamma \in \mathbb{R}$, is the following classification rule: $xw - \gamma > 0 \Rightarrow x$ is red, and $xw - \gamma < 0 \Rightarrow x$ is blue; where xw is the scalar product between x and w.* Geometrically speaking, the perceptron with weights w and margin γ is the hyperplane $xw - \gamma = 0$ that separates the red points from the blue points. Notice that the SVM is a particular case of a perceptron.

When the points are not separable by a straight line, other concepts of separation can be considered. For example, in [94] the authors studied the separation in two dimensions by the use of: a wedge, a strip, a set of wedges, a set of strips, a set of lines, etc. (Figure 1.2). Moreover, separation with planes and extensions of their methods in two dimensions are studied in three dimensions. Another study of separation in $\mathbb{R}^3$ can be found in [1],

and separation by using a circle in [20, 78]. Additionally, results concerning separation with convex polygons and simple polygons in the plane can be found in [14, 49, 53, 88].

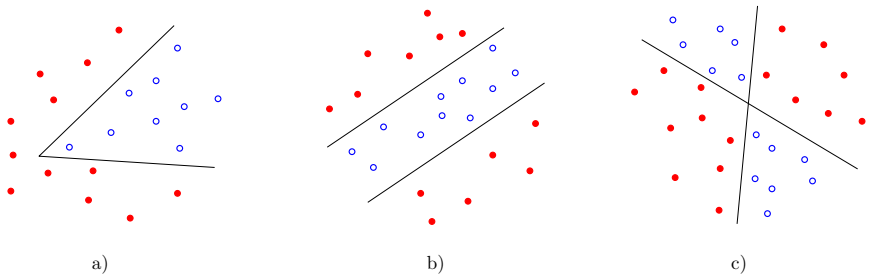


Figure 1.2: Separation with: a) a wedge, b) a strip, y c) two lines.

In some cases, as in medical data analysis [63], classification methods can produce biased classifications due to defective data or data with values out of reasonable ranges. In the terminology of Data Mining, those erroneous data are called *outliers*. In other cases, we may obtain data hard to classify because of strong similarities between subsets of different classes. Culling the minimum number of such points (outlier data) could be a suitable criterion in order to lose the smallest amount of information possible and the for classification method to be effective. For example, when the points are not linearly separable and we insist on using a line (hyperplane) as the pattern of classification, we could remove the minimum number of points so that the resulting point set is separable by a line. In this direction, we arrive at the so-called *Weak Separation Problem* (Figure 1.3 a)). See [29, 50] for results in two dimensions. Another useful criterion is to minimize a measure of the classification errors. Related results can be found in [90], in which the authors consider four measures of separation with hyperplanes and spheres in $\mathbb{R}^d$, and give exact and approximated algorithms. The measures used are: the combinatorial measure (i.e. the number of misclassified points), the $\mathcal{L}_\infty$ measure (i.e. the maximum distance of the misclassified points to the separator object), the $\mathcal{L}_1$ measure (i.e. the sum of the distances of the misclassified points to the separator object), and the $\mathcal{L}_2$ measure (i.e. the sum of the squares of the distances of the misclassified points to the separator object). Furthermore, modifications to the definition of the Perceptron are studied in [16] and can be applied in case the linear separability is not possible. Another example in this scenario is to find two half-planes, one for each class, which their intersection is a strip of minimum width [80] (Figure 1.3 b)).

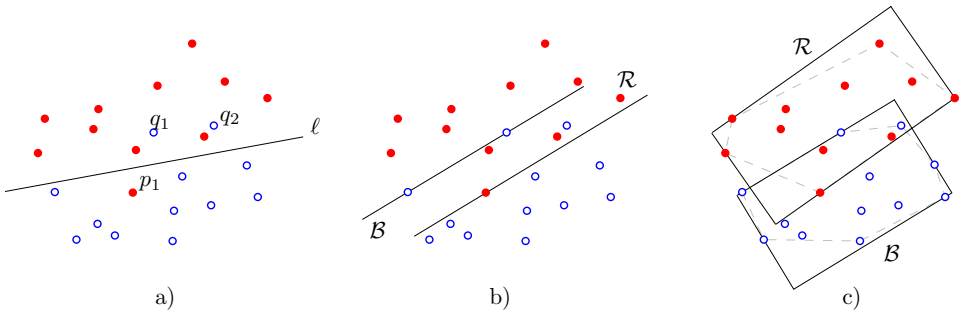


Figure 1.3: a) The weak separation problem. If the red point p_1, and the blue points q_1 and q_2 are removed, then the linear separability is achieved by the line ℓ. b) The half-planes $\mathcal{R}$ and $\mathcal{B}$ contain the red points and the blue points, respectively, and the strip $\mathcal{R} \cap \mathcal{B}$ has minimum width. c) The rectangle $\mathcal{R}$ (resp. $\mathcal{B}$) contains all the red (resp. blue) points and best fits their convex hull.

Classification relies on axis-aligned hyper-rectangles (boxes or rectangles for short) as a main pattern to search clusters and hence prototypes. For numerical data, rectangles generalize multidimensional points and are a good approximation to a description of a subset of them [4, 79]. For instance, in [48, 83] the problem of finding a box containing the maximum number of blue points without containing a red point is studied; and in [44], the one of computing a box of maximum bichromatic discrepancy, that is, the box in which the absolute difference between red and blue points inside it is maximized. The concept of *pattern* is used in Logical Analysis of Data (LAD) [22], and it is generalized by the concept of box [37, 48]. In LAD, it is claimed that very good patterns for classification of data are maximal boxes [64]. This approach is strengthened in [48, 83] with boxes containing the maximum number of data of only one class. Another example is the use of rectangles not necessarily axis-aligned, named *Method of the Hiper-parallelepiped* [80]. The idea of this method is to build a rectangle per class containing all its points and *fitting* their convex hull in the best possible way (Figure 1.3 c)). Most of the optimization problems considered in this thesis use rectangles as classification patterns.

On the other hand, this thesis deals with the concept of *symmetry* in the way red and blue points are treated. We can say that most of the classification techniques are *asymmetric*, that is, the goal is to classify one class of points while the other is discriminated. The aforementioned papers, in which boxes are used, are examples of asymmetric classification. The work of Aronov and Har-Peled [10] asks for a d-dimensional sphere that contains the maximum number of blue points and no red point. This concept can be generalized

to the concept of *symmetric classification* for which a witness set for both classes is found at the same time. This concept has been applied recently by Cabello et al. [26]. They studied how to place two unit disks (or squares) of disjoint interiors, one red and the other blue, so that the number of red points covered by the red disk, plus the number of blue points covered by the blue one, is maximized. Chapter 2 of this thesis deals with a problem of symmetric classification with two boxes.

Another example of asymmetric classification is the covering of one class by using the minimum number of shapes of a given type, but avoiding the elements of the other class (Figure 1.4). In this way, only one class is classified, say the blue class represented by the point set B, and discriminating the other, the red class R. This problem is known in literature as the *Class Cover Problem*. For example, in [27, 41, 84], the problem of obtaining a smallest set of disks centered at blue points, such that their union covers all the blue points and avoids the red ones, is studied. In [84], the authors propose a simple classifier. Once two sets of disks D_B and D_R with the above properties are given, the classifier does the following: Given a new data z, if z is covered by a disk in D_B and by no disk in D_R, then z is classified blue. Otherwise, if there is a disk in D_R that covers z and no disk in D_B that covers z, then z is classified red. Moreover, if z is covered by both a disk in D_B and a disk in D_R, then z is pre-classified "ambiguous", otherwise z is not covered by any disk and is pre-classified "outlier". Following this, if z is "ambiguous" or "outlier", then it is finally classified red or blue according to the closest disk to z (by using a scaled distance function) in D_R or in D_B. Theoretical results related to the class cover problem can be found in [9, 24, 87]. Chapter 4 deals with a non-constrained Class Cover Problem with boxes.

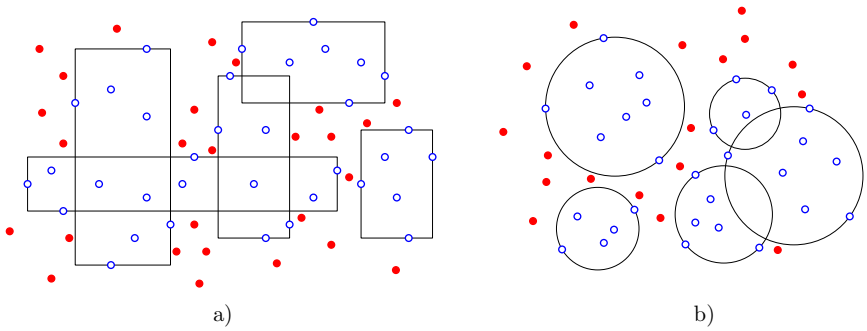


Figure 1.4: Covering the blue class: a) with boxes, b) with disks.

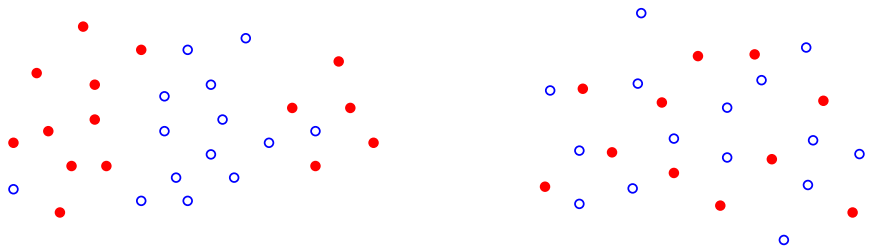
In many applications, data are given in a dynamic scenario. For instance, in

fixed wireless telephony access and driving assistance, detection and recognition of patterns for moving objects are key functions [21]. In fact, with the continued proliferation of wireless communication and advances in positioning technologies, algorithms to efficiently solve optimization problems about large populations of moving data are gaining interest. New devices offer companies an opportunity of providing a diverse range of e-services, many of which will exploit knowledge of the user's changing location. This results in new challenges to database and classification technology in order to model, index, and query moving objects [23, 33, 96, 97]. For more information about real applications of databases with moving objects see [5] and the references therein. In fact, Computational Geometry's techniques can be applied in classification problems under dynamic scenarios. *Dynamic Computational Geometry* studies the combinatorial changes in a geometric structure when its defining objects move according to prescribed motions. For example, given points on the plane moving by straight lines and with known constant velocities, compute the instant of time in which the distance of the two furthest points is minimum. Those kind of problems were introduced in [11], and other problems that arise from collision detection or minimum separation can be found in [62, 93].

One area of research within the Computational Geometry is the design of efficient data structures that maintain certain attributes of a set of continuously moving objects. These structures are known as *Kinetic Data Structures* and they were introduced by Basch et al. [13]. Definitions and results can be found in [5, 13, 60, 61]. Essentially, a kinetic data structure (KDS) for a geometric attribute, is a collection of simple geometric relations that certifies the combinatorial structure of the attribute, as well as a set of rules to repair the attribute and its certifying relations once one relation fails because of the motion [5, 60]. The KDS framework can be applied in many problems concerning extent, proximity, collision detection, connectivity, clustering, etc. of moving objects [60]. In some problems, it is possible that in certain periods of time the attribute of interest might be not needed. For example, given a set of moving points on the plane, find which points are inside a fixed rectangle during only a given period of time. In those cases, a mixture of kinetic and static techniques can be applied [60].

Motivated by recent advances in mobile computing and telecommunications, we are interested in the application of the KDS framework to cluster in classification problems over moving data. Chapter 3 is dedicated to solving a geometric optimization problem in this context.

Finally, we show the motivation for the problem studied in Chapter 5. In database management systems, clustering is often an initial stage for data classification [65]. Suppose we have a set of points, each point classified red or blue, and we want to know if it is possible to divide the set into big monochromatic groups. In this case, we could run a clustering procedure to, for example, use the clusters as a training set in data classification. However, it is not possible if the colored points are *blended* or *uniformly distributed*, and then we could say that the given data set is not suitable to be a training set. Moreover, we have seen that the classification methods depend on the distribution of points, which determines the effectiveness of the classification pattern used. For example, a configuration of points that is good (if fact, the best) for classification is when the point set admits linear separability. On the other corner, the worst case might be when the red and blue points are "well mixed or blended". Intuitively speaking and in order to exemplify, a bicolored set of points in the plane is "suitable" for classification if, at a glance, one can detect clusters formed by many red points and a few blue points, and vice versa (Figure 1.5 a)). On the contrary, we say that a set of points is "unsuitable" for classification when the mixture of red and blue points is high and those clusters are not observed (Figure 1.5 b)). Currently, it is unknown in literature a criterion that allows us to classify a set of points in "suitable" or "unsuitable" for classification, depending of a parameter that measures how much blended the points are. A proposal to meet this concept is presented in Chapter 5.



a) b)

Figure 1.5: a) A "suitable" set for classification, b) a "unsuitable" set for classification.

1.4.1 Notation

Unless otherwise specified, the following terminology is used in this thesis.

Given two sets X and Y, we will denote by $X \setminus Y$ the set that is obtained

by removing from X the set $X \cap Y$. If X is finite, we will denote by $|X|$ the cardinality of X. We also say that X and Y are disjoint if they do not have elements in common, that is, if $X \cap Y$ is the empty set.

In this thesis, $S = R \cup B$ will always denote a set of n points on the plane in general position whose elements are colored either red (the elements of R), or blue (the elements of B). We will assume that R and B are non-empty and have r and b elements respectively. In the figures, red points are always represented by solid dots, and blue points by circles.

Given $X, Y \subset \mathbb{R}^2$, $\mathrm{Red}(X)$ (resp. $\mathrm{Blue}(X)$) denotes the subset of red (resp. blue) points of S that belong to X. We say that X is Y-empty if X does not contain any element of R. We denote as $\mathrm{x}(p)$ (resp. $\mathrm{y}(p)$) the abscissa (resp. ordinate) of the point p.

We use the common notation $O(f(n))$ and $\Omega(f(n))$ (and also $o(f(n))$) to denote the worst case asymptotic upper and lower bounds, respectively. Standard definitions can be found in [35].

1.5 Contribution of this Thesis

In this thesis we study some optimization problems concerning data mining and pattern recognition. By using a geometric modeling, we explore the implicit geometry to efficiently solve the problems from an algorithmic point of view. In some cases, we establish and prove combinatorial and hardness results.

We focus on two-dimensional problems of interest in the classification area. It is worth noticing that, although typically data mining operates with database of points in a multidimensional space, the most famous procedure in classification, the SVM approach, appeared from the geometric interpretation of the classification problem in two dimensions. From the SVM approach, the Computational Geometry techniques began to be applied in Classification, Computer Learning, and Data Mining [16]. In this way, intuitive geometric interpretations of the problems, even in two dimensions, generate interesting methods to be generalized to high dimensions. Moreover, although the running time of the efficient geometric solutions grows exponentially with the dimension, it is possible to obtain efficient algorithms in high dimensions, if we are satisfied with approximated answers. See for example [72], where the authors give efficient approximation algorithms for geometric problems in high dimensions whose running times have polynomial dependence on the dimension and sublinear or subquadratic dependence on

the number of input points.

This section highlights the major contributions of my thesis research. It also provides a summary of the organization of material into chapters.

1.5.1 Chapter 2: Bichromatic Separability with two Boxes

Inspired by the concepts of symmetric classification and erroneous data, we study the problem of, given a bicolored set of n points on the plane, finding two axis-aligned boxes (a red box and a blue box) such that the number of red points that are in the red box and not in the blue box, plus the number of blue points in the blue box that are not in the red, is maximized. This problem is called the *2-EB-problem* (2-Enclosing Boxes problem). We solve this problem in $O(n^2 \log n)$ time, for which the key idea is to dynamically solve instances of the one-dimensional Bentley's problem. The problem of Bentley, the *maximum subsequence problem*, is well known in Computer Science, and is the problem of finding the interval of maximum weight in a given sequence of weighted elements [17]. To our knowledge, the dynamic version of this problem has not been considered before. The dynamic version solved in this thesis is the computation of the solution once the weight of one element is modified, and we prove that it can be done in $O(\log n)$ time. In order to achieve this, we provide the first known data structure, the *MCS-tree*, to solve this problem. The MCS-tree is a powerful and simple data structure that uses linear memory. It can be generalized and used to give efficient solutions to a collection of problems in classification. Moreover, we hope this tool can be useful to efficiently solve many other problems in Computer Science. A basic problem we solve by using MCS-tree is the following: Given a set of weighted points on the plane, find an axis-parallel box such that the sum of the weights of the points contained in the box is maximized. This problem, named *The Maximum Weighted Box problem*, can be solved in $O(n^2 \log n)$ time and $O(n)$ space. Two related problems concerning bicolored point sets are particular cases. The first one is to find a box containing the maximum number of blue points and no red point, which is solved in $O(n^2 \log n)$ time and $O(n)$ space, matching the result of [83]. The second problem is to find a box of maximum *bichromatic discrepancy*, that is, the box such that the absolute difference between red an blue points inside it is maximum. We give an $O(n^2 \log n)$-time and $O(n)$-space solution, that matches the result given in [44]. Another problem solved concerns points colored red, blue, or green. The problem consists in finding three axis-parallel disjoint boxes such that the number of red points inside the

first box, plus the number of blue points inside the second one, and plus the number of green points inside the third one, is maximized. An efficient $O(n \log n)$-time solution is given due to the use of the MCS-tree. This tool is also used to solve our main problem in three dimensions. By reducing the three-dimensional problem to instances in the plane, we provide an algorithm that runs in $O(n^4 \log n)$ time and uses $O(n)$ space. Other applications of the MCS-tree are given in this chapter.

1.5.2 Chapter 3: The Maximum Box Problem for Moving Points on the Plane

Finding a pattern that contains a maximum number of data of one class but avoiding the other is a key operation in data mining and pattern recognition. One example is the aforementioned Maximum Box problem, that is to compute an axis-aligned box (the maximum box) with above condition. This problem has been studied in the cited literature [48, 83] in the static version, that is, data has no movement. As we have mentioned in the above Section, in some real applications data are given in a dynamic scenario. Due to this, we state the problem of maintaining the maximum box over a bicolored point set such that its elements move according known continuous trajectories. The solution proposed in this chapter is a compact and local Kinetic Data Structure (KDS) [5, 13, 60, 61] that allows us to update the maximum box in $O(r \log r + r \log b + b)$ time once two points change the x- or the y-order, where r and b are the number of red and blue points, respectively. The KDS has space complexity $O(r^2 + rb)$ and can be built in $O(r^2 \log r + r^2 \log b + rb \log b)$ time. The static version of the problem, in which the box is not necessarily axis-parallel, is an open problem in [10], and by using our KDS, we present the first nontrivial solution that runs in $O((r + b)^2(r \log r + r \log b + b))$ time. In order to reduce the quadratic space of our KDS, we propose an extension that permits the maintenance of an approximated maximum box that contains at least half of the points contained by the optimal one. This new KDS uses $O((r+b) \log r)$ space and can be built in $O(r \log^2 r + (r + b) \log r \log b))$ time. Each time two points change either their x- or y-order, the solution is updated in $O(r + b)$ time in the worst case.

1.5.3 Chapter 4: The Class Cover Problem with Boxes

In this chapter we study the *Class Cover problem* with boxes, that is, given a set of bicolored points on the plane, find a smallest set of open axis-parallel

rectangles (boxes) which cover all blue points but avoiding red points. We prove the NP-hardness of this problem by a reduction from the *Rectilinear Polygon Covering problem* [38, 85]. We show that an optimal solution has at most $\min\{r+1, b\}$ boxes, where r and b are the number of red and blue points, respectively; and based on this, we present a very simple algorithm that runs in $O(\min\{r^{2r+3}b, r^{2b}b^2\})$ time and has good performance if either r or b is small. We review the theory of ε-nets, which has strong applications to our problem [24, 34, 68, 101], and show that the non-constrained class cover problem admits an $O(\log c)$-approximation, where c is the size of an optimal covering. In case the covering boxes are axis-aligned rectangles there exists an $O(1)$-approximation. Due to the hardness of our problem, we consider the problem for special types of boxes. Firstly, if the covering rectangles are axis-parallel strips we prove the problem is polynomially solvable and give an exact algorithm running in $O(r \log r + b \log b + \sqrt{r}b)$ time. In the case in which the boxes are half-strips oriented in one direction, we present an algorithm that solves the problem in $O((r+b) \log \min\{r, b\})$ time. However, if the covering boxes are half-strips in the four possible directions, then the problem remains NP-hard, and we prove this by a reduction from the 3-SAT-problem [57]. By using results in [34], we show that in this case there exists a $O(1)$-approximation algorithm. To this point, we are unable, in case the rectangles are vertical half-strips, to give either a polynomial-time exact algorithm or a hardness proof. Nevertheless, we provide a 2-approximation approach and leave the problem to further research. We also show that the version in which the covering boxes are axis-aligned squares is NP-hard by a reduction from the problem of covering a binary image with the minimum number of squares [12]. In the definition of the class cover problem we always allow intersection of the covering boxes. It is also proven in this thesis, by reducing from the planar 3-SAT-problem [81], that our problem remains NP-hard even if the boxes are pairwise disjoint. In this case there exists an $O(1)$-approximation algorithm due to results in [88, 89].

1.5.4 Chapter 5: Bichromatic Discrepancy via Convex Partitions

In database management systems, clustering is often an initial stage for data classification [65]. Sometimes, the clustering is obtained by dividing the points (data) into big monochromatic groups. However, it is not possible if the colored points are *blended* or *uniformly distributed*. In this chapter we introduce a new parameter, which is called *discrepancy of a point set*, to measure how *mixed* (or more precisely *blended*) a two-class point set is. Let

R be a set of red points and B a set of blue points on the plane. Let $X \subseteq S$, and $\nabla(X) = ||X \cap R| - |X \cap B||$. We say that a partition $\Pi = \{S_1, S_2, \ldots, S_k\}$ of S is convex if the convex hulls of its members are pairwise disjoint. The discrepancy of a convex partition Π of S is the minimum $\nabla(S_i)$ over the elements of Π and then the *discrepancy* of S, $d(S)$, is the discrepancy of the convex partition of S with maximum discrepancy. We prove several combinatorial properties of the discrepancy, and provide a complete characterization of discrepancy if R and B are linearly separable. Concretely, we show that: $d(S) = \min\{r, b\}$ if the cardinality of the majority color in S is less than twice the minority one, and $d(S) = |r - b|$, otherwise. If the elements of S are in convex position, we obtain that $d(S) = \max_{k=1,2,3} d_k(S)$, where $d_k(S)$ is the discrepancy of S if we restrict to convex partitions of S with exactly k elements. This result implies that the discrepancy of points in convex position can be computed in polynomial time. In fact, we give an efficient $O(n \log n)$-time algorithm. The key idea to efficiently solve the problem is a reduction to instances of problems concerning circular sequences of weighted elements, which are of interest by themselves.

The problem of determining the discrepancy of point sets in general position seems to be non-trivial. Nowadays, we are unable to even characterize point sets with discrepancy one. It is worth noticing that if $d(S)$ is large with respect to the cardinality of S, then there exists an optimal convex partition of S with few elements. However, for point sets with small discrepancy, the minimum cardinality of an optimal convex partition can be small or large.

We also study the particular case in which the discrepancy of S is induced by partitions of S with a straight line (i.e. the *linear discrepancy*), and we provide exact combinatorial lower and upper bounds of the value of discrepancy. Furthermore, we show that computing the linear discrepancy is 3SUM-hard [55] and give an $O(n^2)$-time algorithm. Additionally and as a consequence, we prove that the well-known *Weak Separator problem* [29, 50, 70] belongs to the 3SUM-hard class as well.

1.5.5 Publications

Most of the results of this thesis can be found in the following papers:

- C. Cortés, J. M. Díaz-Báñez, P. Pérez-Lantero, C. Seara, J. Urrutia, and I. Ventura. Bichromatic separability with two boxes: a general approach. *Journal of Algorithms. Cognition, Informatics and Logic.* Vol. 64, No. 2–3, pp. 79–88, 2009.

- S. Bereg, J. M. Díaz-Báñez, P. Pérez-Lantero, and I. Ventura. The Maximum Box Problem for moving points in the plane. *Journal of Combinatorial Optimization.* Published online, 2010.

- S. Bereg, S. Cabello, J. M. Díaz-Báñez, P. Pérez-Lantero, C. Seara, and I. Ventura. The Class Cover Problem with Boxes. *In Proc. 26th European Workshop on Computational Geometry.* Dortmund, 2010.

- S. Bereg, J. M. Díaz-Báñez, D. Lara, P. Pérez-Lantero, C. Seara, and J. Urrutia. Bichromatic Discrepancy Via Convex Partitions. *In Proc. XIII Encuentros de Geometría Computacional.* pp. 259–265, Zaragoza, Spain, 2009.

Chapter 2

Bichromatic Separability with two Boxes

Motivated by the concepts of red-blue separation and outlier data, shown in the introductory chapter, we state the following problem of symmetric classification by using two boxes, one to "cover" red points and the other to "cover" blue points. It is as follows:

The Two Enclosing Boxes problem (2-EB-problem): *Let S be a set of n points on the plane in general position such that the points are colored red or blue. Compute two open axis-aligned rectangles $\mathcal{R}$ and $\mathcal{B}$ in such a way the number of red points in $\mathcal{R} \setminus \mathcal{B}$ plus the number of blue points in $\mathcal{B} \setminus \mathcal{R}$ is maximized.*

We notice here that in the definition of our problem we require $\mathcal{R}$ and $\mathcal{B}$ to be open. This will facilitate our presentation, but in practice we may proceed in a similar way if our boxes are closed. Observe that $\mathcal{R}$ and $\mathcal{B}$ may intersect and that any point in $\mathcal{R} \cap \mathcal{B}$ is not counted. The 2-EB-problem is equivalent to removing from S the minimum number of points (i.e. points considered outliers) so that the intersection between the minimum enclosing box of the red points, and the minimum enclosing box of the blue ones, is S-empty. For example, the solution to the 2-EB-problem for the point set S illustrated in Figure 2.1 a) is $n - 2$, where n is the cardinality of the input set. By removing p and q from S, we can obtain two rectangles $\mathcal{R}$ and $\mathcal{B}$, each of them containing only red and blue points, respectively, and with no point in their intersection.

Notice that an asymmetric separation approach as the one used by Aronov and Har-Peled [10] does not give a solution to our problem (Figure 2.1 b)),

so we must design a procedure which consider $\mathcal{R}$ and $\mathcal{B}$ simultaneously. Bespamyatnikh and Segal [19] studied a two-box covering problem but using a different min-max criterion. They find two axis-parallel boxes that together cover a given set of points in such a way the measure of the largest box is minimized, where the measure is a monotone function of the box.

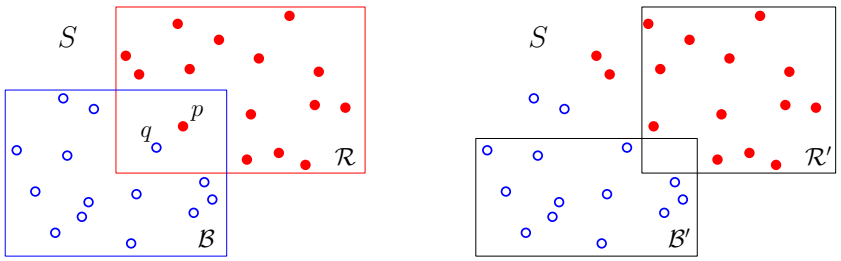


Figure 2.1: a) A solution to the 2-EB-problem for the point set S is given by the boxes $\mathcal{R}$ and $\mathcal{B}$, and is equal to $n-2$, where n is the cardinality of S. The points p and q do not count in the solution. b) For the same set of points S, a B-empty box $\mathcal{R}'$ containing the maximum number of red points, and an R-empty box $\mathcal{B}'$ that contains the maximum number of blue points, do not form a solution.

The 2-EB-problem was first introduced by Cortés et al. [36], solving the problem with an $O(n^3)$-time and space algorithm. In this chapter we show that the 2-EB-problem can be solved in $O(n^2 \log n)$ time and $O(n)$ space. We also introduce a new data structure that allows us to dynamically solve Bentley's [17] well-known *Maximum-Sum Consecutive Subsequence* problem (MCS-problem for short) together with some other variants of this problem that have applications, for instance, in sequence analysis in Bioinformatics [6, 32, 51, 71, 82]. We also show a generalization of our approach that can be used to solve a general type of problem which is of interest in areas such as computer graphics or machine learning [42, 43, 44, 45].

The outline of this chapter is as follows: In Section 2.1 we introduce a data structure, the MCS-tree, to dynamically maintain an optimal solution of the MCS-problem. In Section 2.2 we introduce some notation and present the first results on the 2-EB-problem. In Section 2.3 we show our main result, an $O(n^2 \log n)$-time and linear-space algorithm that solves the 2-EB-problem. In Section 2.5 we show how to solve the following related problem: *Let S be a set of points on the plane in general position such that each element of S is colored red, blue, or green. Find three pairwise-disjoint axis-aligned rectangles $\mathcal{R}$, $\mathcal{B}$, and $\mathcal{G}$ such that the total number of red, blue, and green points contained in $\mathcal{R}$, $\mathcal{B}$, and $\mathcal{G}$ respectively is maximized.* In Section 2.6

we present a generalization of our technique and show how to apply it to several variants of the original problem. In Section 2.7 we show how to solve the 2-EB-problem in three dimensions in $O(n^4 \log n)$ time and $O(n)$ space. Finally, in Section 2.8, we present the conclusions and state future work.

2.1 The dynamic MCS-problem

In this section we describe the main tool that will allow us to solve the 2-EB-problem in $O(n^2 \log n)$ time and $O(n)$ space. Later we show how this technique can be applied to other variants of the same problem. The key idea to solve the 2-EB-problem is a reduction to the computation of some dynamic instances of the following one-dimensional Bentley's problem [17]:

The Maximum-Sum Consecutive Subsequence problem (MCS-problem): *Given a sequence $X = (x_1, x_2, \ldots, x_n)$ and a real weight function w over its elements where for each i, $w(x_i)$ is not necessarily positive, find the consecutive subsequence $(x_i, x_{i+1}, \ldots, x_j)$ of X such that $w(x_i) + w(x_{i+1}) + \cdots + w(x_j)$ is maximum.*

Next we show how to construct a binary tree, the MCS-tree, that allows us to solve the MCS-problem in a dynamic way. The MCS-tree is the first known data structure in literature that dynamically solves the MCS-problem.

2.1.1 The MCS-tree

The MCS-tree is a balanced binary tree with n leaves representing the sequence $X = (x_1, x_2, \ldots, x_n)$. Assume that n is a power of two, otherwise add a few elements with negative weights at the end of the sequence $X = (x_1, \ldots, x_n)$ until we get a sequence of 2^k elements, $n < 2^k < 2n$.

The k-th leaf (from left to right) represents x_k. Each internal node u represents the consecutive subsequence formed by the descendants (leaves) of u. At each node u of the MCS-tree, we store some values, one of which will be the weight of the *maximum weight consecutive subsequence* contained in the subsequence of X defined by the descendants of u. The root vertex will have the solution of the MCS-problem.

Because we focus on the indexes of the *first* and the *last* elements of a subsequence, we use the term *interval* $[x_i, x_j]$ to denote the consecutive subsequence $(x_i, x_{i+1}, \ldots, x_j)$, $1 \leq i \leq j \leq n$ (Figure 2.2).

We build the MCS-tree as follows: Each node u stores the following intervals, as well as their weights, that is, the sum of the weights of their elements:

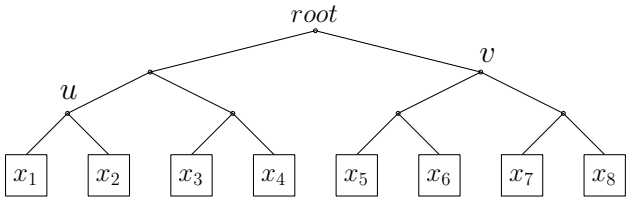


Figure 2.2: The MCS-tree for a sequence of 8 elements. The nodes u, v, and *root* represent the intervals $[x_1, x_2]$, $[x_5, x_8]$, and $[x_1, x_8]$, respectively.

1. $I(u)$: The interval formed by the descendants of u. If u is a leaf representing x_i, $I(u) = [x_i]$.

2. $L(u)$: The interval of maximum weight sum contained in $I(u)$ that contains the leftmost element of $I(u)$. If all the intervals in $I(u)$ containing the leftmost element of $I(u)$ have negative weight, set $L(u) = \emptyset$ and its weight to 0.

3. $R(u)$: The interval of maximum weight sum contained in $I(u)$ that contains the rightmost element of $I(u)$. If all the intervals in $I(u)$ containing the rightmost element of $I(u)$ have negative weight, set $R(u) = \emptyset$ and its weight to 0.

4. $M(u)$: The interval of maximum weight sum that is a subinterval of $I(u)$. If all the intervals in $I(u)$ have negative weight, set $M(u) = \emptyset$ and its weight to 0.

Observe that, if u is the root of the MCS-tree, then $I(u)$ is $[x_1, x_n]$. $L(u)$, $R(u)$, and $M(u)$ are respectively the intervals of the form $[x_1, x_i]$, $[x_j, x_n]$, and $[x_i, x_j]$ with maximum weight sum.

Note that in each node we only store the indexes of the *first* and *last* elements of each of the above intervals as well the weight sum of its elements, and the implicit references to the two children of the node and to the parent. It means that the total used memory for each node is constant. Hence, since there are $2n - 1$ nodes in total (i.e. n leaves plus $n - 1$ interior nodes) the total memory used by the MCS-tree is $O(n)$.

Let u be an internal node of the MCS-tree, and let v_1 and v_2 be its left and right children, respectively. It is clear that if we have the values for $I(v_1)$, $I(v_2)$, $L(v_1)$, $L(v_2)$, $R(v_1)$, $R(v_2)$, $M(v_1)$, and $M(v_2)$, then we can calculate in constant time each of the values of $I(u)$, $L(u)$, $R(u)$, and $M(u)$. Notice that for $M(u)$ there are three possible cases:

(i) $M(u)$ is contained in $I(v_1)$.

(ii) $M(u)$ is contained in $I(v_2)$.

(iii) $M(u)$ overlaps both of $I(v_1)$ and $I(v_2)$.

Thus, $M(u) = M(v_1)$, $M(u) = M(v_2)$, or $M(u) = R(v_1) \cup L(v_2)$ respectively (Figure 2.3). We choose $M(u)$ to be the interval of maximum weight among $M(v_1)$, $M(v_2)$, and $R(v_1) \cup L(v_2)$. Similarly $L(u)$ is the interval of maximum weight among $L(v_1)$ and $I(v_1) \cup L(v_2)$, and $R(u)$ is the interval of maximum weight among $R(v_1) \cup I(v_2)$ and $R(v_2)$.

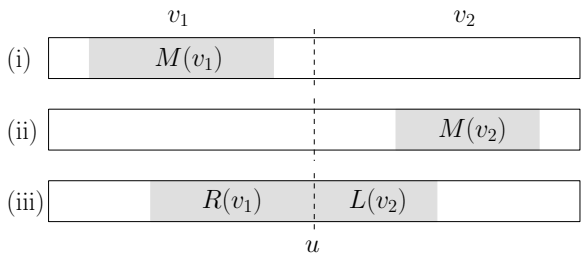


Figure 2.3: The three possible cases for $M(u)$. The nodes v_1 and v_2 are the left child and right child of u respectively. The node v_1 represents the first half of the subsequence represented by u, and v_2 represents the second half.

If a leaf x_i of the MCS-tree has negative weight $w(x_i)$, then $L(x_i) = \emptyset$, $R(x_i) = \emptyset$ and $M(x_i) = \emptyset$. Otherwise $L(x_i) = [x_i]$, $R(x_i) = [x_i]$ and $M(x_i) = [x_i]$. By using these values for the leaves of the MCS-tree and a standard bottom-up traversal [35, 76], we can calculate $L(u)$, $R(u)$, and $M(u)$ for all the nodes u of the tree in linear time.

For more details we can see Algorithm 1 that builds a node of the MCS-tree. Denote by $\text{MaxW}\{I_1, I_2, \ldots, I_k\}$ the interval of maximum weight among the intervals $I_1, I_2, \ldots, I_k$. By $\text{left}(u)$, $\text{right}(u)$ and $\text{parent}(u)$ we denote respectively the left child, right child and parent of the node u.

Note that all the nodes in the MCS-tree whose represented subsequence contains a given leaf x_i are those in the path from x_i to the root. Therefore, if the weight $w(x_i)$ of a leaf x_i changes, the MCS-tree can be updated in a bottom-up traversal from x_i to the root, and thus the weight of the maximum weight consecutive subsequence, as well its weight value, are recalculated in $O(\log n)$ time. The complexity follows because the height of the MCS-tree is $O(\log n)$ since it is a complete balanced tree with n leaves [76].

Moreover, for a given $x_i \in X$, by traversing from x_i to the root, the MCS-tree structure allows us to obtain in $O(\log n)$ time: (i) the optimal interval

Algorithm 1 Construction of the node u of the MCS-tree

if u is a leaf **then**
  $I(u) \leftarrow [u]$
  if $w(u) < 0$ **then**
    $L(u) \leftarrow R(u) \leftarrow M(u) \leftarrow \emptyset$
  else
    $L(u) \leftarrow R(u) \leftarrow M(u) \leftarrow [u]$
  end if
else
  $I(u) \leftarrow I(\text{left}(u)) \cup I(\text{right}(u))$
  $L(u) \leftarrow \text{MaxW}\{L(\text{left}(u)), I(\text{left}(u)) \cup L(\text{right}(u))\}$
  $R(u) \leftarrow \text{MaxW}\{R(\text{left}(u)) \cup I(\text{right}(u)), R(\text{right}(u))\}$
  $M(u) \leftarrow \text{MaxW}\{M(\text{left}(u)), R(\text{left}(u)) \cup L(\text{right}(u)), M(\text{right}(u))\}$
end if

Algorithm 2 Maximum weight sum interval containing x_k, where x_k is a leaf of the MCS-tree

- I_L (resp. I_R) is the interval of maximum weight ending (resp. beginning) at x_k
- S_L (resp. S_R) is the interval beginning (resp. ending) at the element of the leftmost (resp. rightmost) leaf of the subtree rooted at x and ending (resp. beginning) at x_k

$I_L \leftarrow I_R \leftarrow S_L \leftarrow S_R \leftarrow [x_k]$
$u \leftarrow x_k$
repeat
  $v \leftarrow \text{parent}(u)$
  if $u = \text{left}(v)$ **then**
    $I_R \leftarrow \text{MaxW}\{I_R, S_R \cup L(\text{right}(v))\}$
    $S_R \leftarrow S_R \cup I(\text{right}(v))$
  else
    $I_L \leftarrow \text{MaxW}\{I_L, S_L \cup R(\text{left}(v))\}$
    $S_L \leftarrow S_L \cup I(\text{left}(v))$
  end if
  $u \leftarrow v$
until $u =$ the root of the MCS-tree
return $I_L \cup I_R$

for the MCS-problem which contains x_i, and (ii) the maximum weight of the interval of X starting or ending at x_i. The operation (i) is stated in Algorithm 2, and the operation (ii) is a special case of the operation (i).

These properties of the MCS-tree structure will be used to solve the 2-EB-problem. From the discussion above we get the following result.

Theorem 2.1 *Given the sequence $X = (x_1, \ldots, x_n)$ with a real weight function w over its elements, the MCS-tree can be built in $O(n)$ time. Moreover, if any $w(x_i)$ is modified, the MCS-tree can be updated in $O(\log n)$ time. The MCS-tree allows computation, in $O(\log n)$ time, of the interval of maximum weight sum of consecutive elements that includes, starts or ends at a specific element $x_k \in X$.*

Other operations that can be performed with the MCS-tree, by reasoning similarly as above, are the following:

1. Given two indexes i and j ($1 \leq i \leq j \leq n$) of the sequence $X = (x_1, \ldots, x_n)$, find in $O(\log n)$ time the interval of maximum weight contained in the interval $[x_i, x_j]$.

2. Given four indexes i, j, k and l ($1 \leq i \leq j \leq n$ and $1 \leq k \leq \ell \leq n$) of the sequence $X = (x_1, \ldots, x_n)$, find in $O(\log n)$ time the interval of maximum weight such that its first and last elements are inside the intervals $[x_i, x_j]$ and $[x_k, x_\ell]$, respectively.

The two above operations were studied in [32]. The first one as *The Range Maximum-Sum Segment Query Problem* and the second one as *The Range Maximum-Sum Segment Query Problem with Two Query Intervals*. They were solved with the following method: Make a linear preprocessing to the sequence X in such a way that subsequent queries, for the given indexes, are answered each in constant time. The approach in [32] does not consider changes in the elements of the sequence as is done in the MCS-tree but with logarithmic-time queries.

On the other hand, if we consider that the sequence $X = (x_1, x_2, \ldots, x_n)$ is circular (i.e. if $1 \leq j < i \leq n$ the interval $[x_i, x_j]$ is the union of the intervals $[x_i, x_n]$ and $[x_1, x_j]$), then we can augment the MCS-tree in order to dynamically compute the (circular) interval of maximum weight when an element of X changes. In fact, store in each node u of the MCS-tree the interval $E(u)$ and its weight, where $E(u)$ is the interval of minimum

weight contained in $I(u)$. If all the intervals in $I(u)$ have positive weight, set $E(u) = \emptyset$ and its weight to 0. Notice that the interval of maximum weight in $I(u)$ is among $M(u)$ and the interval obtained by removing $E(u)$ from $I(u)$. If a leaf x_i of the MCS-tree has negative weight, then $E(x_i) = [x_i]$, otherwise $E(x_i) = \emptyset$. If u is an internal node, with left and right children and v_1 and v_2 respectively, then $E(u)$ is among $E(v_1)$, $E(v_2)$, and the interval obtained by removing both $L(v_1)$ and $R(v_2)$ from $I(u)$.

The idea in the design of the MCS-tree considers that the sequence is static in the sense that it admits neither insertions nor deletions of elements, that is, from the beginning the MCS-tree contains the same elements that change their weights. Both operations can be included if we apply the techniques of augmenting trees in [35]. In fact, we can use a variant of either a *Red-Black-tree* [35] or a *AVL-tree* [76] (in which the elements of the sequence are stored at the leaves and each internal node contains the number of leaves descendants of it) as supporting tree for the MCS-tree, and so include the insertion and deletion of elements in the represented sequence. We do not give details in order to maintain the course of this work.

2.1.2 Conclusions

The idea of this section, for the purpose of dynamically solving the Bentley's problem, is based on simple observations. Its result, the MCS-tree, is a simple and powerful data structure. The $O(n)$-time static solution to the Bentley's problem was proposed by Jay Kadane [17]. The same time complexity can be obtained if we apply the *Divide and Conquer Approach* [35] and make a recursive algorithm. This idea has lead us to a generalization approach that is stated in Section 2.6.

2.2 Notation and preliminary results

An optimal solution to the 2-EB-problem for S consists of two axis-aligned rectangles $\mathcal{R}$ and $\mathcal{B}$ and the subset of S containing the red points of S in $\mathcal{R} \setminus \mathcal{B}$ together with the blue points of S in $\mathcal{B} \setminus \mathcal{R}$. For technical reasons, which become necessary in Proposition 2.3, we assume that $\mathcal{R}$ and $\mathcal{B}$ are *open*, i.e., they do not include their boundaries, thus we do not count points on the boundaries of $\mathcal{R}$ and $\mathcal{B}$.

Up to symmetry, there are three possible relative positions of $\mathcal{R}$ and $\mathcal{B}$. A pair $(\mathcal{R}, \mathcal{B})$ is called: a *corner-type* if $\mathcal{R}$ overlaps exactly one corner of $\mathcal{B}$ (Figure 2.4 a)), a *sandwich-type* if $\mathcal{R}$ intersects only two parallel sides of

$\mathcal{B}$ (Figure 2.4 b)), and a *disjoint-type* if $\mathcal{R}$ and $\mathcal{B}$ are disjoint (Figure 2.4 c)). A fourth type could exist, however it can be reduced to a disjoint-type (Figure 2.4 d)). Likewise, we say that a solution is *corner-type*, *sandwich-type* or *disjoint-type* if the pair of rectangles defining it are *corner-type*, *sandwich-type* or *disjoint-type*, respectively.

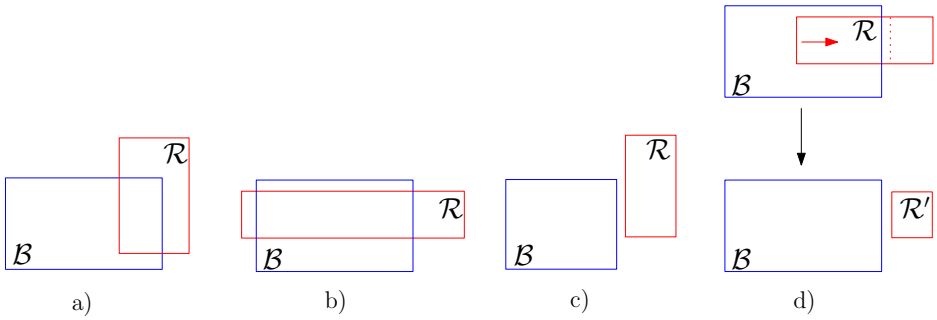


Figure 2.4: a) Corner-type, b) sandwich-type, c) disjoint-type, and d) getting a disjoint-type.

We show now how to obtain optimal solutions of the corner-type case. In Section 2.3 we describe the efficient algorithm not only for this case but also for the other cases. Therefore, from now on $(\mathcal{R}, \mathcal{B})$ denotes a corner-type pair of rectangles. We assume without loss of generality (w.l.o.g) that $\mathcal{R}$ always contains the top-right corner of $\mathcal{B}$, as in Figure 2.5 a)).

For a given $u \in \mathbb{R}^2$, we respectively denote by $\mathrm{SW}(u)$, $\mathrm{SE}(u)$, $\mathrm{NW}(u)$ and $\mathrm{NE}(u)$, the South-West, South-East, North-West and North-East open quadrants with respect to u, e.g., if $u = (a, b)$, then $\mathrm{NE}(u) = \{(x, y) \mid a < x, b < y\}$. The point u is the *apex* of $\mathrm{SW}(u)$, $\mathrm{SE}(u)$, $\mathrm{NW}(u)$, and $\mathrm{NE}(u)$.

Let $(\mathcal{Q}_\mathcal{R}, \mathcal{Q}_\mathcal{B})$ be a North-East and South-West pair of quadrants. We say that $(\mathcal{Q}_\mathcal{R}, \mathcal{Q}_\mathcal{B})$ is a *corner-type* pair of quadrants if the apex of $\mathcal{Q}_\mathcal{R}$ belongs to $\mathcal{Q}_\mathcal{B}$ (Figure 2.5 b)).

The following proposition establishes the relation between the optimal solutions formed by a corner-type pair of rectangles, and optimal solutions formed by a corner-type pair of quadrants.

Proposition 2.2 *If the 2-EB-problem for S has an optimal solution formed by a corner-type pair of rectangles, then it has an optimal solution formed by a corner-type pair of quadrants, or it has an optimal disjoint-type solution.*

Proof. Let $(\mathcal{R}, \mathcal{B})$ be an optimal corner-type pair of rectangles for S. Let $\mathcal{Q}_\mathcal{R}$ be the North-East quadrant whose apex is the bottom-left corner of

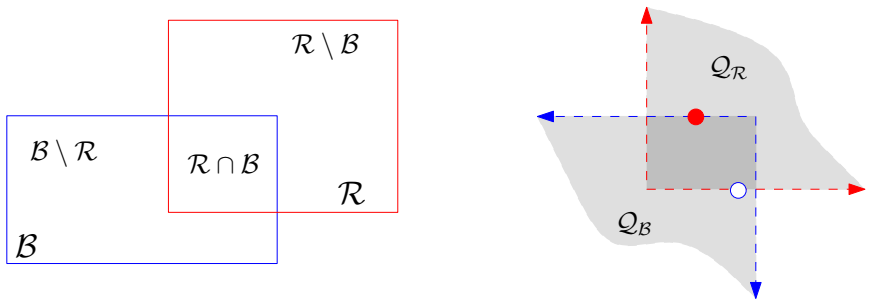


Figure 2.5: a) A corner-type pair of rectangles, b) a corner-type pair of quadrants.

$\mathcal{R}$ and let $\mathcal{Q}_\mathcal{B}$ be the South-West quadrant whose apex is the top-right corner of $\mathcal{B}$ (Figure 2.5). Notice that $|\operatorname{Red}(\mathcal{Q}_\mathcal{R} \setminus \mathcal{R})| = |\operatorname{Blue}(\mathcal{Q}_\mathcal{B} \setminus \mathcal{B})| = 0$, otherwise $(\mathcal{R}, \mathcal{B})$ would not form an optimal corner-type solution for S. Hence $|\operatorname{Red}(\mathcal{Q}_\mathcal{R} \setminus \mathcal{Q}_\mathcal{B})| + |\operatorname{Blue}(\mathcal{Q}_\mathcal{B} \setminus \mathcal{Q}_\mathcal{R})| = |\operatorname{Red}(\mathcal{R} \setminus \mathcal{B})| + |\operatorname{Blue}(\mathcal{B} \setminus \mathcal{R})|$ and thus, the corner-type pair of quadrants $(\mathcal{Q}_\mathcal{R}, \mathcal{Q}_\mathcal{B})$ is an optimal solution for S. ■

Now, the idea is to find optimal solutions formed by a corner-type pair of quadrants. The following proposition gives us a discretization of the search space of solutions of this type.

Proposition 2.3 *There is an optimal corner-type pair of quadrants $(\mathcal{Q}_\mathcal{R}, \mathcal{Q}_\mathcal{B})$ of the 2-EB-problem for S such that the horizontal ray bounding $\mathcal{Q}_\mathcal{B}$ contains a red point and the horizontal ray bounding $\mathcal{Q}_\mathcal{R}$ contains a blue point.*

Proof. Let $(\mathcal{Q}_\mathcal{R}, \mathcal{Q}_\mathcal{B})$ be an optimal corner-type pair of quadrants of the 2-EB-problem for S. Let $S' \subseteq S$ be the set of points of S in $\mathcal{Q}_\mathcal{R} \setminus \mathcal{Q}_\mathcal{B}$. Translate $\mathcal{Q}_\mathcal{B}$ vertically in the upward direction until its boundary hits a point $p \in S'$. If p is a blue point, ignore it (as it will not change the solution given by $\mathcal{Q}_\mathcal{B}$ and $\mathcal{Q}_\mathcal{R}$). Thus, we can translate $\mathcal{Q}_\mathcal{B}$ upwards until its horizontal ray hits a red point in S'. If no such red point exists, then $\mathcal{Q}_\mathcal{B}$ can become a half-plane, say $\mathcal{HP}_\mathcal{B}$. As no element of S' can be in $\mathcal{HP}_\mathcal{B} \cap \mathcal{Q}_\mathcal{R}$, we can then move $\mathcal{Q}_\mathcal{R}$ to the right until it no longer intersects $\mathcal{HP}_\mathcal{B}$. Then $\mathcal{Q}_\mathcal{R}$ can become a half-plane $\mathcal{HP}_\mathcal{R}$ which is disjoint with $\mathcal{HP}_\mathcal{B}$. Thus, we obtain an optimal solution to the 2-EB-problem for S which is not a corner-type quadrant. Analogously, we can prove that the horizontal ray bounding $\mathcal{Q}_\mathcal{B}$ contains a blue point (Figure 2.5 b)). ■

For the sake of clarity, here we include a first approximation to our techniques by using a simple method to compute a corner-type solution in

$O(n^4)$ time and $O(n^2)$ space. First, observe that the size of the subset of S, say S', we are seeking will be equal to $|S'| = |\operatorname{Red}(\mathcal{Q}_\mathcal{R} \setminus \mathcal{Q}_\mathcal{B})| + |\operatorname{Blue}(\mathcal{Q}_\mathcal{B} \setminus \mathcal{Q}_\mathcal{R})|$. Consider the orthogonal grid G formed by horizontal and vertical lines passing through the points of S. Using range search techniques as in [18] we can perform a quadratic time preprocessing on the nodes of G such that for each node $u \in G$ we calculate and store the values: $|\operatorname{Red}(\mathrm{SW}(u))|$, $|\operatorname{Blue}(\mathrm{SW}(u))|$, $|\operatorname{Red}(\mathrm{SE}(u))|$, $|\operatorname{Blue}(\mathrm{SE}(u))|$, $|\operatorname{Red}(\mathrm{NW}(u))|$, $|\operatorname{Blue}(\mathrm{NW}(u))|$, $|\operatorname{Red}(\mathrm{NE}(u))|$, and $|\operatorname{Blue}(\mathrm{NE}(u))|$.

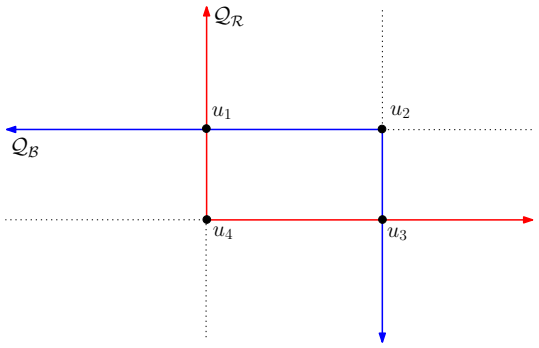


Figure 2.6: Looking for a corner-type solution.

Let $(\mathcal{Q}_\mathcal{R}, \mathcal{Q}_\mathcal{B})$ be a corner-type pair of quadrants on G. Denote by u_1, u_2, u_3, and u_4 the four vertices of the rectangle $\mathcal{Q}_\mathcal{R} \cap \mathcal{Q}_\mathcal{B}$ as in Figure 2.6. From the following formulas:

$$|\operatorname{Red}(\mathcal{Q}_\mathcal{R} \setminus \mathcal{Q}_\mathcal{B})| = |\operatorname{Red}(\mathrm{NE}(u_1))| + |\operatorname{Red}(\mathrm{NE}(u_3))| - |\operatorname{Red}(\mathrm{NE}(u_2))|,$$

$$|\operatorname{Blue}(\mathcal{Q}_\mathcal{B} \setminus \mathcal{Q}_\mathcal{R})| = |\operatorname{Blue}(\mathrm{SW}(u_1))| + |\operatorname{Blue}(\mathrm{SW}(u_3))| - |\operatorname{Blue}(\mathrm{SW}(u_4))|,$$

it follows that $|S'|$ can be computed in constant time. This gives us an $O(n^4)$-time and $O(n^2)$-space algorithm to solve the 2-EB-problem.

2.3 Exact and efficient solution

In this section we describe an efficient algorithm which solves the 2-EB-problem for S in $O(n^2 \log n)$ time and $O(n)$ space.

The corner-type solution

We start with the more complex and general case. We show how to find a corner-type pair of quadrants $(\mathcal{Q}_\mathcal{R}, \mathcal{Q}_\mathcal{B})$ that yields an optimal solution to the 2-EB-problem for S.

Let h_p denote the horizontal line passing through a point $p \in S$. We color h_p as follows: if p is red (resp. blue), then color h_p blue (resp. red). By observing Proposition 2.3 we can assume that the horizontal ray that bounds $\mathcal{Q}_\mathcal{B}$ has a red point on it, and that the horizontal ray bounding $\mathcal{Q}_\mathcal{R}$ has a blue point. Thus, for each pair (p, q) of red and blue points of S, w.l.o.g. supposing that the y-coordinate of p is larger than the y-coordinate of q, we solve the following problem.

Let H be the horizontal strip bounded by the lines h_p and h_q. Let v_p be the vertical line passing through p (Figure 2.7 a)). For each red point p' on the right of v_p and below h_p, let $\mathcal{Q}_\mathcal{B}(p')$ be the South-West blue quadrant defined by h_p and the vertical line passing through p'. Similarly, for each blue point q' on the left of v_p and above h_q, let $\mathcal{Q}_\mathcal{R}(q')$ be the North-East red quadrant bounded by h_q and the vertical line passing through q' (Figure 2.7 b)).

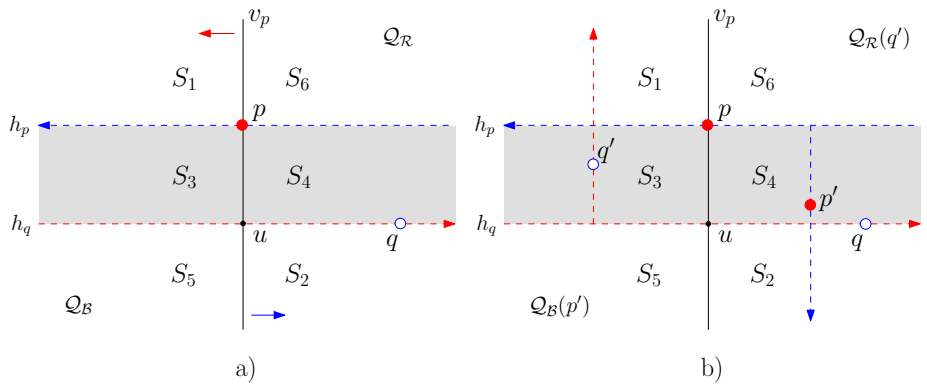


Figure 2.7: a) Starting position, b) optimal position.

We consider the following problem for a given input H.

The Horizontal Strip problem (HS-problem): *Find* $(\mathcal{Q}_\mathcal{R}(q'), \mathcal{Q}_\mathcal{B}(p'))$ *such that: (i) the horizontal ray of* $\mathcal{Q}_\mathcal{B}(p')$ *is contained in* h_p*, (ii) the horizontal ray of* $\mathcal{Q}_\mathcal{R}(q')$ *is contained in* h_q *and passes through* p*, (iii)* p *belongs to the interior of* $\mathcal{Q}_\mathcal{R}(q')$*, and (iv)* $|\,\mathrm{Red}(\mathcal{Q}_\mathcal{R}(q') \setminus \mathcal{Q}_\mathcal{B}(p'))| + |\,\mathrm{Blue}(\mathcal{Q}_\mathcal{B}(p') \setminus \mathcal{Q}_\mathcal{R}(q'))|$ *is maximized.*

By Propositions 2.2 and 2.3, it is possible to find an optimal corner-type pair of quadrants for the 2-EB-problem for S by solving the HS-problem for $O(n^2)$ instances, where each instance corresponds to a pair (p, q) of points of S defining a horizontal strip H. Next we show how to solve the HS-problem for the $O(n^2)$ instances in a dynamic way by solving $O(n^2)$ instances of the MCS-problem.

Let u be the intersection point of v_p and h_q, and consider the quadrants NE(u) and SW(p). To solve the HS-problem for H, we slide NE(u) to the left and SW(p) to the right, until we reach an optimal solution. In order to do this task, we assign weights to the points of S as follows: The lines v_p, h_p and h_q divide the plane into six regions $S_1, \ldots, S_6$ as in Figure 2.7.

- All the red points in S_1 and all the blue points in S_2 receive weight 1.
- All the blue points in S_3 and all the red points in S_4 receive weight -1.
- All the remaining points receive weight 0.

Store in p the number of blue points in SW(p) and in u the number of red points in NE(u). Project vertically all the red and blue points on h_q. The following result is obtained:

Lemma 2.4 *To obtain an optimal* $(\mathcal{Q}_\mathcal{R}(q'), \mathcal{Q}_\mathcal{B}(p'))$ *to the HS-problem for* H *is equivalent to finding the maximum weight interval* I *on* h_q *such that* I *contains* p*. Moreover,* $|\operatorname{Red}(\mathcal{Q}_\mathcal{R}(q') \setminus \mathcal{Q}_\mathcal{B}(p'))| + |\operatorname{Blue}(\mathcal{Q}_\mathcal{B}(p') \setminus \mathcal{Q}_\mathcal{R}(q'))| = |\operatorname{Red}(\mathrm{NE}(u))| + |\operatorname{Blue}(\mathrm{SW}(p))| + w(I)$*, where* $w(I)$ *is the weight of* I*.*

Proof. Let $(\mathcal{Q}_\mathcal{R}(q'), \mathcal{Q}_\mathcal{B}(p'))$ be a solution to the HS-problem for a given H. $\mathcal{Q}_\mathcal{R}(q')$ is obtained by sliding leftwards $\mathcal{Q}_\mathcal{R} = \mathrm{NE}(u)$. During the sliding red points in S_1 enter inside $\mathcal{Q}_\mathcal{R}$, and blue points in S_3 enter inside $\mathcal{Q}_\mathcal{R} \cap \mathcal{Q}_\mathcal{B}$. That is to say, red points in S_1 will be counted into the solution and blue points of S_3 will not. Therefore, q' is the position where the number of red points of S_1 that have been included inside the solution minus the number of blue points of S_3 that have been discarded is maximum. In other words, since the weights of all the red points to the left of p in h_q value 0, except the weights of all red points in S_1 that value 1 and the weights of all blue points in S_3 that value -1, we obtain that q' is such that the interval of maximum weight I_1, that is on h_q and ends at p, starts at the element consecutive to q' in h_q. Analogously, p' is such that the interval of maximum weight I_2, that is on h_q and starts at p, ends at the element preceding p' in h_q. Evidently, $I = I_1 \cup I_2$ is the interval of maximum weight in h_q that contains p, and that $|\operatorname{Red}(\mathcal{Q}_\mathcal{R}(q') \setminus \mathcal{Q}_\mathcal{B}(p'))| + |\operatorname{Blue}(\mathcal{Q}_\mathcal{B}(p') \setminus \mathcal{Q}_\mathcal{R}(q'))| = |\operatorname{Red}(\mathrm{NE}(u))| + w(I_1) + |\operatorname{Blue}(\mathrm{SW}(p))| + w(I_2) = |\operatorname{Red}(\mathrm{NE}(u))| + |\operatorname{Blue}(\mathrm{SW}(p))| + w(I)$. ■

The general idea of the algorithm is the following: We make a top-bottom sweep of all the points of S with the blue horizontal line h_p, that is, at the

beginning of the sweep, h_p is above all the points of S, and at the end it will be below them. The line h_p stops at each red point p. For each stop of h_p, we make another top-bottom sweep of the points of S that are below h_p with a red horizontal line h_q. The line h_q stops at blue points, thus for each position of h_p and h_q we have an instance of the HS-problem which is dynamically solved by doing the above assignment of weights, by building a MCS-tree, and by the application of Theorem 2.1 and Lemma 2.4.

Suppose that h_p is fixed. We slide h_q down, stopping each time h_q meets a point q of S, and following the rules: (i) if q is a red point in S_2, q will enter into S_4 and its weight will change to -1; (ii) if q is a red point in S_5, q will enter into S_3 and its weight will remain 0; (iii) if q is a blue point and it enters into S_4, its weight changes to 0; and (iv) if q is blue point and it enters into S_3, its weight changes to -1. Each time h_q crosses a red point to the right of v_p we update in constant time the number of red points in $\mathrm{NE}(u)$ by incrementing it by one, and each time h_q hits a blue point, we recalculate the optimal solution of the HS-problem for the new H (bounded above by h_p and below by the new position of h_q).

This immediately suggests using the dynamic version of the computation of the interval of maximum weight sum on h_q. By using the MCS-tree and Theorem 2.1, the interval of maximum weight sum containing the current p can be computed in $O(\log n)$ time per stop-point of the slide line h_q. Thus, the time cost for the fixed red point p is $O(n) + O(n \log n) = O(n \log n)$.

Now, for each one of the $O(n)$ red points p in S, we set the horizontal line h_p, calculate in linear time the number of blue points in $\mathrm{SW}(p)$ and the number of red points in $\mathrm{NE}(u)$ (which is initially equal to $|\,\mathrm{Red}(\mathrm{NE}(p))|$), and start again the process by rebuilding the MCS-tree in linear time, and then dynamically solving the MCS-problem. Therefore, the overall time complexity of the algorithm is $O(n)(O(n) + O(n \log n)) = O(n^2 \log n)$. The following result is then obtained.

Theorem 2.5 *An optimal corner-type solution for the 2-EB-problem for S can be found in $O(n^2 \log n)$ time and $O(n)$ space.*

The sandwich-type solution

We now show how to find a sandwich-type solution. The method is similar to the corner-type solution, and it is based on the following propositions whose proofs are similar to the proofs of Proposition 2.2 and Proposition 2.3, respectively.

Proposition 2.6 *There exists a sandwich-type solution $(\mathcal{R}, \mathcal{B})$ to the 2-EB-problem if and only if there exists a pair of strips $(\mathcal{S}_\mathcal{R}, \mathcal{S}_\mathcal{B})$, one vertical and the other horizontal, that maximize the sum $|\operatorname{Red}(\mathcal{S}_\mathcal{R} \setminus \mathcal{S}_\mathcal{B})| + |\operatorname{Blue}(\mathcal{S}_\mathcal{B} \setminus \mathcal{S}_\mathcal{R})|$.*

Proposition 2.7 *There exists a pair of strips $(\mathcal{S}_\mathcal{R}, \mathcal{S}_\mathcal{B})$, $\mathcal{S}_\mathcal{R}$ vertical and $\mathcal{S}_\mathcal{B}$ horizontal, that maximize the sum $|\operatorname{Red}(\mathcal{S}_\mathcal{R} \setminus \mathcal{S}_\mathcal{B})| + |\operatorname{Blue}(\mathcal{S}_\mathcal{B} \setminus \mathcal{S}_\mathcal{R})|$ such that there is a red element of S on the top side of the rectangle determined by their intersection.*

Therefore, we can proceed in a similar way to the corner-type solution. For each pair of blue lines h_p and h_q (Figure 2.8 a)), we consider the blue strip $\mathcal{S}_\mathcal{B}$ bounded by them, plus a starting red strip $\mathcal{S}_\mathcal{R}$ consisting of the vertical red line v_p passing through a red point p in h_p (Figure 2.8 a)). They form an initial candidate sandwich-type solution with value $|\operatorname{Blue}(\mathcal{S}_\mathcal{B})|$. As we widen $\mathcal{S}_\mathcal{R}$ by translating two vertical lines v_1 and v_2 to the left and right of p, respectively, the value of the solution will change according to the following weight rules applied to the points of S, where the regions S_1, S_2, and S_3 are as in Figure 2.8 b):

- Blue points in S_1, S_2, and S_3 receive weight 0, -1, and 0, respectively.
- Red points in S_1, S_2, and S_3 receive weight 1, 0, and 1, respectively.

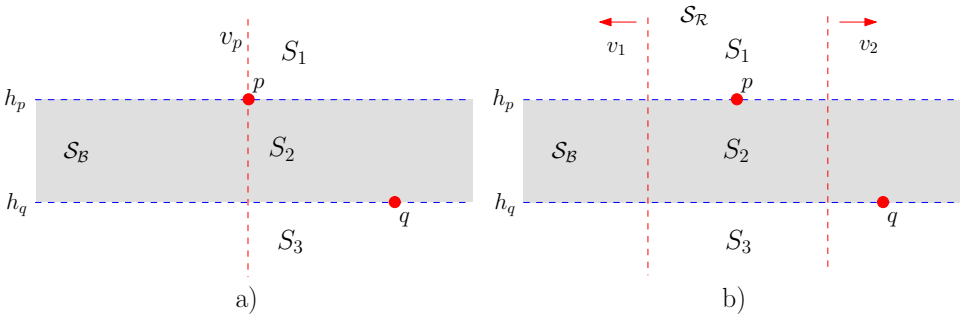


Figure 2.8: a) Starting position, b) finding an optimal position.

If we store in p the number of blue points in $\mathcal{S}_\mathcal{B}$ and project the points of S on h_q, we can state the following lemma whose proof is very similar to the proof of Lemma 2.4.

Lemma 2.8 *Given the horizontal lines h_p and h_q bounding the strip $\mathcal{S}_\mathcal{B}$, to obtain the positions of v_1 and v_2 that optimally define $\mathcal{S}_\mathcal{R}$, is equivalent*

to finding the maximum weight interval I on h_q that contains p. Moreover, $|\operatorname{Blue}(\mathcal{S}_\mathcal{B} \setminus \mathcal{S}_\mathcal{R})| + |\operatorname{Red}(\mathcal{S}_\mathcal{R} \setminus \mathcal{S}_\mathcal{B})| = |\operatorname{Blue}(\mathcal{S}_\mathcal{B})| + w(I)$ *where $w(I)$ is the weight of I.*

For each position of h_p we assign weight 0 to all the blue points of S and weight 1 to all the red ones. Initially, We locate the line h_q in the same position of h_p and build in linear time a MCS-tree T for the projected sequence of points. With h_q we make a sweep of the points below h_p. Each time h_q crosses a point q we do the following: If q is blue we change its weight to -1 and update the number of blue points in $\mathcal{S}_\mathcal{B}$. If q is red its weight is changed to 0 and then we dynamically calculate in $O(\log n)$ time by using T, because of Theorem 2.1, the maximum weight sum interval in h_q that contains the current point p. Therefore, we obtain a candidate to the sandwich-type solution if we apply Lemma 2.8. As a consequence, we obtain the following result.

Theorem 2.9 *An optimal sandwich-type solution for the 2-EB-problem for S can be found in $O(n^2 \log n)$ time and $O(n)$ space.*

The disjoint-type solution

Finding a disjoint-type solution is straightforward. It can be done in time $O(n \log n)$ by making two plane sweeps, the first one with a horizontal line and the next one with a vertical line. In the two sweeps we keep the position of the sweeping line that maximizes the number of red points to one of its sides plus the number of blue points to the another one. This is due to the observation that there is always a vertical or horizontal line that separates the boxes of a disjoint-solution.

Therefore, putting together the results above, we get to the following theorem as the main result of this chapter

Theorem 2.10 *The 2-EB-problem for S can be solved in $O(n^2 \log n)$ time and $O(n)$ space.*

2.4 Approximated solution

As we have seen, the 2-EB-problem has an $O(n^2 \log n)$-time solution. For big values of n the time complexity would be very high. A way to reduce it is by using an approximated algorithm, that is, an algorithm that reports

a solution that is an approximation to the optimal one. We state as future work the analysis of the computational complexity of the 2-EB-problem to see if there is a better time complexity algorithm for it. We conjecture that an $o(n^2)$-time complexity is impossible to achieve. In order to improve the time complexity, we propose in this section an approximated solution that consists of finding only disjoint-type solutions.

Before we show the main result of this section, we state the following definition from [35, 100]:

Definition 2.11 *Let A be an algorithm that computes a feasible solution for every instance I of a given problem P. Let $A(I)$ and $Opt(I)$ be respectively the value of the solution reported by A and the value of the optimal solution for I. If P is a maximization problem (resp. minimization problem) then A is an k-approximation to P if for every instance I of P we have that $k \cdot Opt(I) \leq A(I) \leq Opt(I)$ (resp. $Opt(I) \leq A(I) \leq k \cdot Opt(I)$). In this case k is the factor of approximation of A.*

Notice in the above definition that the closer to one k is the better the approximation algorithm. The following result states the factor of approximation obtained when we compute only disjoint-type solutions for the 2-EB-problem.

Theorem 2.12 *The algorithm to find a disjoint-type solution is a 3/4-approximation to the 2-EB-problem.*

Proof. Let $\mathcal{S}_{apx}$ and $\mathcal{S}_{opt}$ be respectively the value of the disjoint-type solution and the value of the general optimal solution. Suppose that the general optimal solution, $(\mathcal{Q}_{\mathcal{R}}, \mathcal{Q}_{\mathcal{B}})$, is a corner-type. Consider the lines h_1, h_2, v_1 and v_2 dividing $\mathcal{Q}_{\mathcal{R}}$ and $\mathcal{Q}_{\mathcal{B}}$ into the six regions S_1, S_2, S_3, S_4, S_5 and S_6, as in Figure 2.9 a).

For h_1, since $\mathcal{S}_{apx}$ is the value of the disjoint-type solution, it results that $\mathcal{S}_{apx}$ is greater than or equal to the number of red points above h_1 plus the number of blue points below h_1, and in turn, this quantity is greater than or equal to $|\operatorname{Blue}(\mathcal{Q}_{\mathcal{B}} \setminus \mathcal{Q}_{\mathcal{R}})| + |\operatorname{Red}(S_5)| + |\operatorname{Red}(S_6)|$. Thus we obtain:

$$\mathcal{S}_{apx} \geq |\operatorname{Blue}(\mathcal{Q}_{\mathcal{B}} \setminus \mathcal{Q}_{\mathcal{R}})| + |\operatorname{Red}(S_5)| + |\operatorname{Red}(S_6)|$$

Analogously, for the lines h_2, v_1 and v_2, the following three inequalities are obtained:

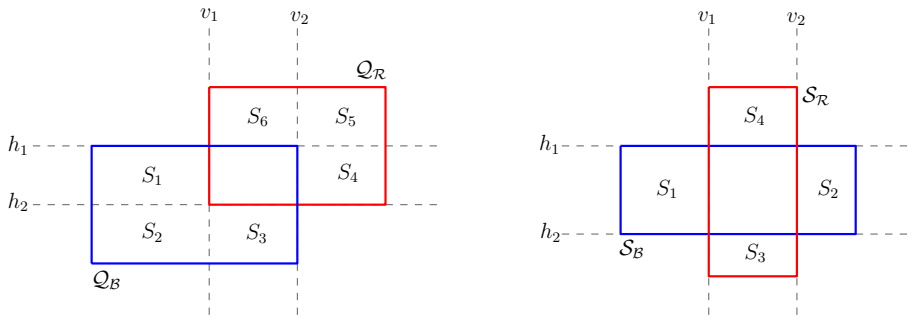


Figure 2.9: a) Corner type-solution, b) sandwich-type solution.

$$\mathcal{S}_{apx} \geq |\operatorname{Blue}(S_2)| + |\operatorname{Blue}(S_3)| + |\operatorname{Red}(\mathcal{Q}_\mathcal{R} \setminus \mathcal{Q}_\mathcal{B})|$$
$$\mathcal{S}_{apx} \geq |\operatorname{Blue}(S_1)| + |\operatorname{Blue}(S_2)| + |\operatorname{Red}(\mathcal{Q}_\mathcal{R} \setminus \mathcal{Q}_\mathcal{B})|$$
$$\mathcal{S}_{apx} \geq |\operatorname{Blue}(\mathcal{Q}_\mathcal{B} \setminus \mathcal{Q}_\mathcal{R})| + |\operatorname{Red}(S_4)| + |\operatorname{Red}(S_5)|$$

Since $\mathcal{S}_{opt} = |\operatorname{Blue}(\mathcal{Q}_\mathcal{B} \setminus \mathcal{Q}_\mathcal{R})| + |\operatorname{Red}(\mathcal{Q}_\mathcal{R} \setminus \mathcal{Q}_\mathcal{B})| = |\operatorname{Blue}(S_1)| + |\operatorname{Blue}(S_2)| + |\operatorname{Blue}(S_3)| + |\operatorname{Red}(S_4)| + |\operatorname{Red}(S_5)| + |\operatorname{Red}(S_6)|$, we obtain the following by adding the four inequalities:

$$4\mathcal{S}_{apx} \geq 3\mathcal{S}_{opt} + |\operatorname{Blue}(S_2)| + |\operatorname{Red}(S_5)| \geq 3\mathcal{S}_{opt}$$

Suppose now that the optimal solution to the 2-EB-problem, $(\mathcal{S}_\mathcal{R}, \mathcal{S}_\mathcal{B})$, is a sandwich-type. Consider the lines h_1, h_2, v_1, and v_2 dividing $\mathcal{S}_\mathcal{R}$ and $\mathcal{S}_\mathcal{B}$ into the regions S_1, S_2, S_3 and S_4, as in Figure 2.9 b). This gives us the following:

$$\mathcal{S}_{apx} \geq |\operatorname{Red}(S_4)| + |\operatorname{Blue}(\mathcal{S}_\mathcal{B} \setminus \mathcal{S}_\mathcal{R})|$$
$$\mathcal{S}_{apx} \geq |\operatorname{Red}(S_3)| + |\operatorname{Blue}(\mathcal{S}_\mathcal{B} \setminus \mathcal{S}_\mathcal{R})|$$
$$\mathcal{S}_{apx} \geq |\operatorname{Blue}(S_1)| + |\operatorname{Red}(\mathcal{S}_\mathcal{R} \setminus \mathcal{S}_\mathcal{B})|$$
$$\mathcal{S}_{apx} \geq |\operatorname{Blue}(S_2)| + |\operatorname{Red}(\mathcal{S}_\mathcal{R} \setminus \mathcal{S}_\mathcal{B})|$$

Since $\mathcal{S}_{opt} = |\operatorname{Blue}(\mathcal{S}_\mathcal{B} \setminus \mathcal{S}_\mathcal{R})| + |\operatorname{Red}(\mathcal{S}_\mathcal{R} \setminus \mathcal{S}_\mathcal{B})| = |\operatorname{Blue}(S_1)| + |\operatorname{Blue}(S_2)| + |\operatorname{Red}(S_3)| + |\operatorname{Red}(S_4)|$ we obtain:

$$4\mathcal{S}_{apx} \geq 3\mathcal{S}_{opt}$$

In both cases $\frac{3}{4}\mathcal{S}_{opt} \leq \mathcal{S}_{apx} \leq \mathcal{S}_{opt}$, resulting that the optimal disjoint-type solution is a 3/4-approximation to the 2-EB-problem. ■

2.5 The three chromatic case with three disjoint boxes

In this section we study the following problem as an extension of the 2-EB-problem in which the points are colored red, blue or green. To this end, we define Green(Y), as was done for Red(Y) and Blue(Y), as the subset of green points of S that are in Y.

The Disjoint Three-Chromatic Enclosing Boxes problem (DTEB-problem): *Let S be a set of n points on the plane in general position such that the points are colored red, blue, and green. The DTEB-problem for S consist of finding three pairwise-disjoint isothetic rectangles $\mathcal{R}$, $\mathcal{B}$, and $\mathcal{G}$ such that $|\,\mathrm{Red}(\mathcal{R})| + |\,\mathrm{Blue}(\mathcal{B})| + |\,\mathrm{Green}(\mathcal{G})|$ is maximum.*

Theorem 2.13 *The DTEB-problem for S can be solved in $O(n \log n)$ time and $O(n)$ space.*

Proof. First, we observe that given any three pairwise-disjoint isothetic rectangles $\mathcal{R}$, $\mathcal{B}$, and $\mathcal{G}$ we can always find two isothetic lines (or a line and a half-line) ℓ_1 and ℓ_2 such that for any pair of elements of $\{\mathcal{R}, \mathcal{B}, \mathcal{G}\}$ either ℓ_1 or ℓ_2 separates them (Figure 2.10). The two cases are solved separately.

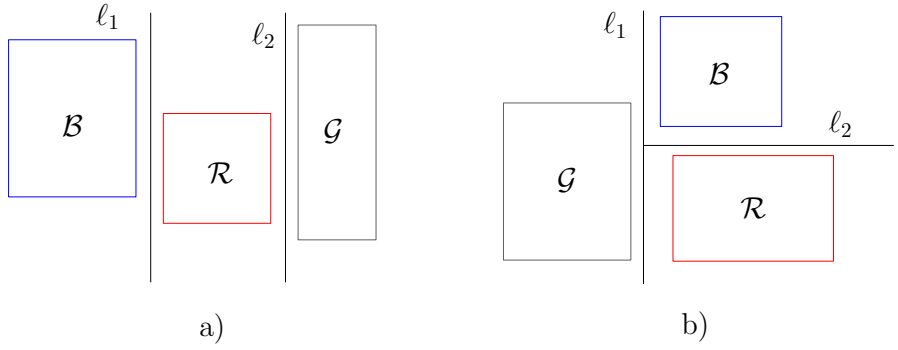


Figure 2.10: Separation by two isothetic lines: a) parallel lines, b) perpendicular lines.

Parallel case: First, we show how to solve the parallel case by reducing it to the *Longest Increasing Subsequence problem* [76]: *Given a sequence $(x_1, \ldots, x_n)$ of numbers, find a largest subset of indexes $\alpha_1 < \cdots < \alpha_k$ such that $x_{\alpha_1} \leq \cdots \leq x_{\alpha_k}$*. It is well known that this problem can be solved in $O(n \log n)$ time [54]. In fact, the instance we have to solve here can be solved

in linear time, as it involves a sequence whose elements have values 1, 2, or 3. First, suppose that ℓ_1 and ℓ_2 are vertical and that $\mathcal{R}$, $\mathcal{B}$, and $\mathcal{G}$ form an optimal solution. We have to consider six cases for the relative positions of $\mathcal{R}$, $\mathcal{B}$, and $\mathcal{G}$ with respect to ℓ_1 and ℓ_2. Suppose that $\mathcal{B}$ is to the left of ℓ_1, that $\mathcal{R}$ lies between ℓ_1 and ℓ_2, and that $\mathcal{G}$ is to the right of ℓ_2 (Figure 2.10 a)). The remaining five cases are solved in a similar way.

Assign weight 1 to the points colored blue, weight 2 to those colored red, and weight 3 to the green points. Project all the points in S on the x-axis obtaining a sequence Σ of 1's, 2's, and 3's. Observe that all the blue, red, and green points contained in $\mathcal{B}$, $\mathcal{R}$, and $\mathcal{G}$, respectively, as projected on the x-axis induce an increasing subsequence of Σ. The result follows.

Perpendicular case: Suppose, w.l.o.g., that the relative positions of ℓ_1, ℓ_2, $\mathcal{R}$, $\mathcal{B}$, and $\mathcal{G}$ are as in Figure 2.10 b)). We show how to solve this case using dynamic binary trees.

First, suppose that the elements of S are labeled $p_1, \ldots, p_n$ such that the y-coordinate of p_i is smaller than the y-coordinate of p_j, $i < j$. Suppose that we are given the line ℓ_1 and introduce an auxiliary color, the black, for recoloring all the green points and also all the red and blue points to the left of ℓ_1. Given an index i, $1 \leq i \leq n$, let $R(i)$ (resp. $B(i)$) be the number of red elements p_j with $j \leq i$ (resp. blue p_j's with $j \geq i$). Then, the best position of ℓ_2, given the position of ℓ_1, is determined by the index i ($1 \leq i \leq n$) that maximizes $R(i) + B(i)$.

By using the above idea, we construct a balanced binary tree T such that its set of leaves $S = \{p_1, \ldots, p_n\}$ are colored red, blue, and black. Our objective is to store information on the vertices of T such that the following problem, which we call the *Maximum Sum problem*, or the MS-problem for short, can be solved dynamically in $O(\log n)$ time:

The Maximum Sum problem (MS-problem): *Find an index i that maximizes $R(i) + B(i)$. At each point in time, a red or blue point can change color to black.*

As in Section 2.1.1, for every internal node u of T let $I(u) = [p_{l_u}, p_{r_u}]$ be the interval of S formed by the descendants of u in T. If $u = p_j$ for some j, $I(u) = [p_j]$. For an index i, $l_u \leq i \leq r_u$, let $R_u(i)$ (resp. $B_u(i)$) be the number of red p_j's such that $l_u \leq j \leq i$ (resp. the number of blue p_k's with $i \leq k \leq r_u$). We define $I_R(u)$ (resp. $I_B(u)$) as the number of red (resp. blue) points in $I(u)$.

At every node u of T we will store the following information: an index i_u,

$l_u \leq i_u \leq r_u$, such that $R_u(i_u) + B_u(i_u)$ is maximized. If u is a leaf p_j of T, then $i_u = j$.

If u is an internal node of T whose left and right children are v_1 and v_2, respectively, it is easy to see that i_u is either i_{v_1} or i_{v_2}, according to the following criterion: If $R(i_{v_1}) + B(i_{v_1}) + I_B(v_2) \geq I_R(v_1) + R(i_{v_2}) + B(i_{v_2})$ then $i_u = i_{v_1}$, otherwise $i_u = i_{v_2}$.

Following this, we can immediately see that by using a bottom-up traversal of T, we can calculate the values $I_R(u)$, $I_B(u)$, i_u, $R(i_u)$, and $B(i_u)$ for all nodes u of T in linear time by Theorem 2.1. Moreover, by Theorem 2.1, it is straightforward to see that if a red or blue point p_i of T is re-colored black, then we can update T in $O(\log n)$ time by traversing the path from p_i to the root of T, and that if *root* is the root vertex of T, $R(i_{root}) + B(i_{root})$ is the solution to the MS-problem.

We are now ready to solve the perpendicular case. Take a copy S' of S and re-color black to all the green elements of S'. Construct a binary tree T as described above such that the elements of S' are the leaves of T. In this way, $R(i_{root}) + B(i_{root})$ is the optimal solution for which the green box contains no points.

We now perform a line sweep using a vertical line ℓ_1 from left to right. Initially all the elements of S are to the right of ℓ_1, at the end all the elements of S are to the left of ℓ_1. Each time ℓ_1 meets a point in S we change the color of its corresponding copy in S' to black. In $O(\log n)$ time, we update T, and recalculate the boxes $\mathcal{R}$ and $\mathcal{B}$ that maximize the number of blue points plus the number of red points contained in them. Each time ℓ_1 meets a green point, the number of elements in $\mathcal{G}$ increases by one. By keeping the maximum number of red points plus blue points plus green points contained in $\mathcal{R}$, $\mathcal{B}$, and $\mathcal{G}$, respectively, we obtain an optimal solution to our problem and the algorithm requires $O(n \log n)$ of time and $O(n)$ of space. ■

2.6 Generalization and Applications

We have shown how to solve the 2-EB-problem and the DTEB-problem by using dynamic trees over a sequence of elements for which some attribute is dynamically maintained. Notice that in the case of the 2-EB-problem (resp. the DTEB-problem) the interval of maximum weight (resp. the position in which the number of red elements to its left plus the number of blue points to its right is maximum) is maintained in $O(\log n)$ time. The approach can

easily be generalized as follows.

2.6.1 Generalization

It is easy to see that for computing in a given sequence, both the interval of maximum weight and the position in which the number of red elements to its left plus the number of blue points to its right is maximum, we can apply the *Divide and Conquer Approach* [35] and obtain a recursive linear-time algorithm. This is due to the fact that if we have computed the corresponding attribute for the two halves of the sequence, then we can compute the same attribute for the entire sequence in constant time. Under this observation the generalization idea follows.

Generalization: *Let $X = (x_1, x_2, \ldots, x_n)$ be a sequence of n elements and let $\mathcal{A}(X)$ be a set of attributes of X that depends on its elements. Suppose that $\mathcal{A}(X)$ can be obtained by applying a recursive $O(n)$-time algorithm as follows: if the length of X is at most one compute $\mathcal{A}(X)$ in constant time, otherwise $\mathcal{A}(X)$ is computed in constant time from $\mathcal{A}(X_1)$ and $\mathcal{A}(X_2)$, where X_1 and X_2 are the two halves of X. Then, the recursive tree having the elements of X as its leaves is a balanced binary tree and by representing it we can, whenever some x_i changes, recompute $\mathcal{A}(X)$ in $O(\log n)$ time by traversing the path from x_i to the root.*

Note that although the goal may be to maintain only one attribute, we maintain a set of them because in many applications the calculation of an attribute of the sequence depends on others. For the sake of clarity see the MCS-tree (Section 2.1.1), where the property of the sequence is the weight of its elements and other three attributes are considered in order to maintain the interval of maximum weight.

By applying this generalization we are now ready to present efficient algorithms for a collection of problems.

2.6.2 Applications

The Maximum Weighted Box problem (MWB-problem): *Given a set S of n points on the plane and a weight function $w : S \to \mathbb{R}$, compute the axis-aligned box H such that $\sum_{x \in H \cap S} w(x)$ is maximized.*

Note that there exists a box H that gives an optimal solution such that H contains on its boundary only elements of S with positive weight. A solution can be calculated as follows:

Make a top-bottom sweep of the elements of S with a horizontal line ℓ_1 and whenever it stops at a positive element of S, make a sweep of the elements of S that lie below ℓ_1 with another horizontal line ℓ_2 that starts at ℓ_1 and stops only on positive-weight points. Let X be the points of S ordered by abscissa and let w' be a weight function for each $x \in S$ defined as follows:

$$w'(x) = \begin{cases} w(x) & \text{if } x \text{ lies between } \ell_1 \text{ and } \ell_2; \\ 0 & \text{if } x \text{ lies either above } \ell_1 \text{ or below } \ell_2. \end{cases}$$

For a given position of ℓ_1 and ℓ_2, the optimal box, whose top and bottom sides lie on ℓ_1 and ℓ_2, respectively, is determined by the interval of maximum weight sum on X. This interval is dynamically computed in $O(\log n)$ time, by using a MCS-tree, when the weight w' of an element of S changes while the line ℓ_2 moves. Like this, we obtain a simple $O(n^2 \log n)$-time algorithm since there are $O(n^2)$ possible positions of ℓ_1 and ℓ_2.

If we restrict H to be a quadrant, say the North-East quadrant, the solution is even easier. We use only a horizontal sweeping line ℓ_1 and define the weight function w' as follows: $w'(x) = w(x)$ if x is above or in ℓ_1, and $w'(x) = 0$ otherwise. The problem is reduced to the dynamic computation of the interval of maximum weight in X that is a prefix of X. The time and space complexities are $O(n \log n)$ and $O(n)$, respectively.

The solutions to the following two problems are found by solving one or two instances of the MWB-problem. Thus, they also have solutions when we restrict the corresponding box to a quadrant.

The Maximum Box problem (MB-problem): *Given a set of blue points B and a set of red points R on the plane, where $|R \cup B| = n$, find an R-empty axis-aligned box H such that $|H \cap B|$ is maximized.*

See Figure 2.11 a) for an example. The MB-problem can be solved in $O(n^2 \log n)$ time by using $O(n)$ space since it is an instance of the MWB-problem by considering $S = R \cup B$ and the weight function:

$$w(x) = \begin{cases} 1 & \text{si } x \in B; \\ -\infty & \text{si } x \in R. \end{cases}$$

The MB-problem was solved in [83] with $O(b^2 \log b + br + r \log r)$ time, where $r = |R|$ and $b = |B|$. However our approach is a simpler method with the same complexity in the worst case.

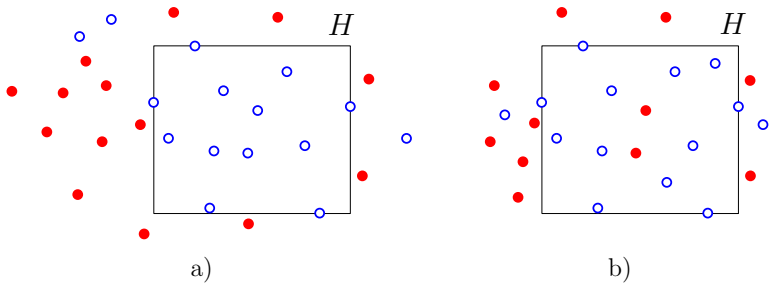


Figure 2.11: a) Solution H to the MB-problem, b) solution H to the MBDB-problem.

The Maximum Bichromatic Discrepancy Box problem (MBDB-problem): *Given a set of blue points B and a set of red points R on the plane, where $|R \cup B| = n$, find an axis-aligned box H such that $||H \cap B| - |H \cap R||$ is maximized.*

In Figure 2.11 b) we depict an instance of the MBDB-problem and its solution.

A solution to the MBDB-problem can be obtained by solving the following two instances of the MWB-problem, in both consider that $S = R \cup B$:

$$1) \quad w(x) = \begin{cases} 1 & \text{si } x \in R, \\ -1 & \text{si } x \in B; \end{cases} \qquad 2) \quad w(x) = \begin{cases} -1 & \text{si } x \in R, \\ 1 & \text{si } x \in B. \end{cases}$$

In this way, the MBDB-problem can be solved in $O(n^2 \log n)$ time by using $O(n)$ space. This complexity matches the result given in [44].

The Weak Strip Separation problem (WSS-problem): *Let S be a set of n points on the plane in general position such that its elements are colored red or blue. Find a corridor C bounded by two parallel lines in any direction such that the number of blue points inside C plus the number of red points outside C is maximized.*

An instance of the WSS-problem and its corresponding solution are shown in Figure 2.12.

Suppose that we have a direction given by a line ℓ and we want to compute the best corridor C_ℓ that is orthogonal to ℓ. It can be done as follows: Project the points of S on ℓ, obtaining the sequence of elements $X = (x_1, \ldots, x_n)$ ordered from left to right. Given an index i, $1 \leq i \leq n$, let $R^-(i)$ (resp.

$R^+(i)$) be the number of red elements x_k with $k \leq i$ (resp. $k \geq i$) and given two indexes i and j, $1 \leq i \leq j \leq n$, let $B(i,j)$ be the number of blue elements x_k with $i \leq k \leq j$. It is easy to see that C_ℓ is determined by the indexes i and j, $1 \leq i \leq j \leq n$, such that $R^-(i) + B(i,j) + R^+(j)$ is maximum. Define $V(i,j) = R^-(i) + B(i,j) + R^+(j)$.

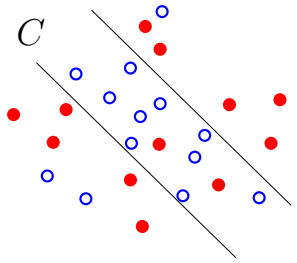


Figure 2.12: Solution to the WSS-problem.

Now we explain the generalization idea. For the sequence X let i_x y j_x, $i_x < j_x$, be the indexes of X that maximizes $V(i_x, j_x)$. For a given index i, let $B^-(i)$ (resp. $B^+(i)$) be the number of blue elements x_k with $k \leq i$ (resp. $k \geq i$). Let p_x and q_x be the indexes of X that maximize $R^-(p_x) + B^+(p_x)$ and $B^-(q_x) + R^+(q_x)$ respectively. Abusing of the notation, let $R(X)$ and $B(X)$ be respectively the number of red and blue elements in X. The attribute of interest is the pair of indexes i_x and j_x, but for their dynamic computation we consider the set of attributes $\mathcal{A}(X) = \{i_x, j_x, V(i_x, j_x), p_x, q_x, R^-(p_x) + B^+(p_x), B^-(q_x) + R^+(q_x), R(X), B(X)\}$. Calculate $\mathcal{A}(X)$ is trivial when the length of X is one. For computing $\mathcal{A}(X)$ from $\mathcal{A}(X_1)$ and $\mathcal{A}(X_2)$, where X_1 and X_2 are the two halves of X, there are three cases:

(i) $i_x = i_{x_1}$, $j_x = j_{x_1}$ and $V(i_x, j_x) = V(i_{x_1}, j_{x_1}) + R(X_2)$.

(ii) $i_x = p_{x_1}$, $j_x = q_{x_2}$ and $V(i_x, j_x) = R^-(p_{x_1}) + B^+(p_{x_1}) + B^-(q_{x_2}) + R^+(q_{x_2})$.

(iii) $i_x = i_{x_2}$, $j_x = j_{x_2}$ and $V(i_x, j_x) = R(X_1) + V(i_{x_2}, j_{x_2})$.

In all of them, $R(X) = R(X_1) + R(X_2)$ and $B(X) = B(X_1) + B(X_2)$. Moreover, the attributes $p_x, q_x, R^-(p_x) + B^+(p_x)$ and $B^-(q_x) + R^+(q_x)$ are calculated as was done for the DTEB-problem in Section 2.5.

If we rotate the line ℓ about the origin, the order of the projected points on ℓ changes a quadratic number of times. Thus, we make a rotational sweep passing from the current critical direction to the next one by swapping two consecutive elements of X. When the swap occurs, the solution is

dynamically computed by using a tree constructed by following the above arguments. The swap involves two changes of color in two consecutive elements and hence, two updates on the tree. This method has $O(n^2 \log n)$-time and $O(n^2)$-space complexities.

The Weak Cross Separation problem (WCS-problem): *Let S be a point set on the plane in general position such that its elements are colored red or blue. Find a point u on the plane such that* $|\text{Blue}(\text{NE}(u))| + |\text{Red}(\text{NW}(u))| + |\text{Blue}(\text{SW}(u))| + |\text{Red}(\text{SE}(u))|$ *is maximized.*

The point u can be seen as the intersection point between a horizontal line ℓ_h and a vertical line ℓ_v defining a cross (Figure 2.13). Suppose we are given ℓ_h and we want to compute the best location of ℓ_v. It can be done as follows: Project the points of S on ℓ_h, obtaining the sequence of elements $X = (x_1, \ldots, x_n)$ ordered from left to right. Given an index i, $1 \leq i \leq n$, let $R_a(i)$ (resp. $R_b(i)$) be the number of red elements x_k with $k \leq i$ (resp. $k \geq i$) such that its corresponding point in S is above (resp. below) ℓ_h, and let $B_a(i)$ (resp. $B_b(i)$) be the number of blue elements x_k with $k \geq i$ (resp. $k \leq i$) such that its corresponding point in S is above (resp. below) ℓ_h. Note that the best position of ℓ_v is determined by the index i such that $R_a(i) + B_a(i) + B_b(i) + R_b(i)$ is maximum. Define $V(i) = R_a(i) + B_a(i) + B_b(i) + R_b(i)$.

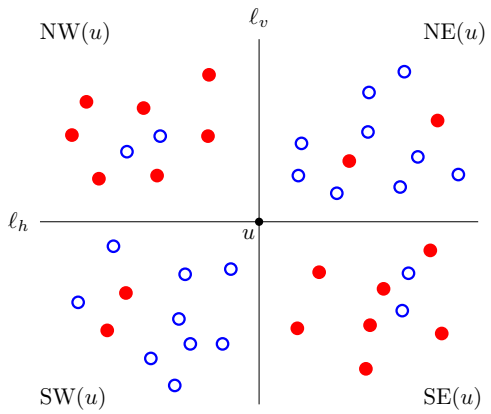


Figure 2.13: A solution to the WCS-problem is defined by the point u that is seen as the intersection point between ℓ_v and ℓ_h.

In order to apply the generalization idea, we associate to each element $x_k \in X$, among its color, a binary property $\text{pos}(x_k)$ that is equal to one if and only if the point of S that corresponds to x_k is above ℓ_h. Denote as i_x the index of X that maximizes $V(i_x)$, and let $R_a(X)$ and $B_a(X)$ (resp. $R_b(X)$ and $B_b(X)$) be respectively the number of red and blue elements in X that are

above (resp. below) ℓ_h. $\mathcal{A}(X)$ is $\{i_x, V(i_x), R_a(X), B_a(X), R_b(X), B_b(X)\}$, and it is easy to compute from the color and the property pos$(\cdot)$ of its elements. Calculate $\mathcal{A}(X)$ is trivial when the length of X is one, otherwise there are two cases for computing it from $\mathcal{A}(X_1)$ and $\mathcal{A}(X_2)$, where X_1 and X_2 are the two halves of X:

(i) $i_x = i_{x_1}$ and $V(i_x) = V(i_{x_1}) + B_a(X_2) + R_b(X_2)$.

(ii) $i_x = i_{x_2}$ and $V(i_x) = R_a(X_1) + B_b(X_1) + V(i_{x_2})$.

In both cases $R_a(X) = R_a(X_1) + R_a(X_2)$, $B_a(X) = B_a(X_1) + B_a(X_2)$, $R_b(X) = R_b(X_1) + R_b(X_2)$ y $B_b(X) = B_b(X_1) + B_b(X_2)$.

Initially, the line ℓ_h is above all the points of S, that is, pos$(x_k) = 0$ for all elements x_k in X. We represent X in a tree T as the generalization idea establishes, and make a top-bottom sweep of S with ℓ_h changing in each step, from zero to one, the property pos$(\cdot)$ of an element of X. It permits us, by using T, to compute dynamically the best position of ℓ_v on each step of ℓ_h. Thus, an $O(n \log n)$-time and $O(n)$-space algorithm is obtained.

2.7 The problem in three dimensions

In this section we show that the 2-EB-problem in three dimensions can be solved in $O(n^4 \log n)$ time and $O(n)$ space. There are five solution cases, in some of them we make reductions to instances in two dimensions and in others we apply the generalization approach of Section 2.6.

An optimal solution to the 2-EB-problem in three dimensions consists of two isothetic three-dimensional boxes $\mathcal{R}$ and $\mathcal{B}$. Up to symmetry, there are five possible relative positions of $\mathcal{R}$ and $\mathcal{B}$. Their names were taken from [7] where the problem is to find two boxes in three dimensions that enclose a given set of points and minimize some measure such as total volume, total surface area, etc.

According to [7], a pair of boxes $(\mathcal{R}, \mathcal{B})$ is called: a *vertex-in-type* if exactly one vertex of each box lies inside the other box (Figure 2.14 a)); a *edge-in-type* if exactly one edge of one box is fully contained within the other box (Figure 2.14 b)); a *piercing-type* if one box completely pierces the other one, so that no box contains any vertex of the other box (Figure 2.14 c)); a *crossing-type* if $\mathcal{R}$ and $\mathcal{B}$ have an "edge interaction" in which two edges of one box pass through the other, and viceversa (Figure 2.14 d)); and a *disjoint-type* if $\mathcal{R}$ and $\mathcal{B}$ are disjoint (Figure 2.14 e)).

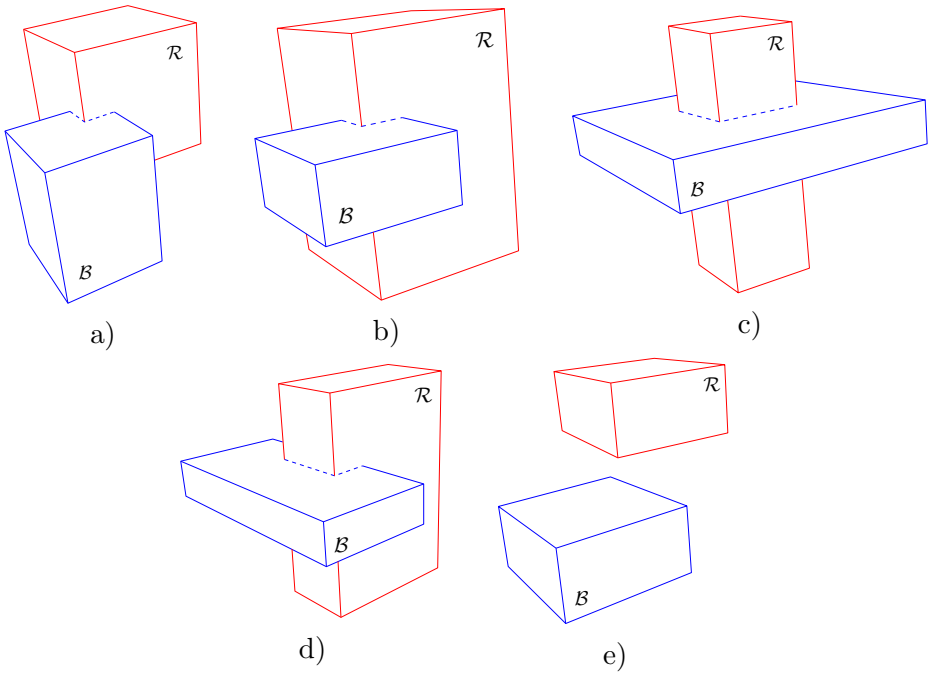


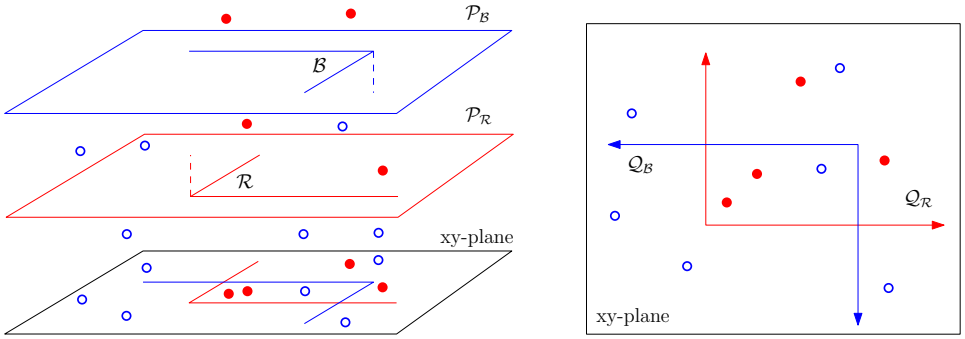
Figure 2.14: The five cases of solution in $\mathbb{R}^3$. a) Vertex-in-type, b) edge-in-type, c) piercing-type, d) crossing-type, e) disjoint-type.

Now we describe how to solve each of the five cases of solution.

The vertex-in-type solution

Let $\mathcal{P}_\mathcal{R}$ and $\mathcal{P}_\mathcal{B}$ be two half-planes that are parallel to the xy-plane, and such that $\mathcal{P}_\mathcal{B}$ is above $\mathcal{P}_\mathcal{R}$ in the direction given by the z-axis. Suppose we want to find the vertex-in-type solution $(\mathcal{R}, \mathcal{B})$ such that the bottom facet of $\mathcal{R}$ and the top facet of $\mathcal{B}$ are contained in $\mathcal{P}_\mathcal{R}$ and $\mathcal{P}_\mathcal{B}$ respectively. We say that a red point p above $\mathcal{P}_\mathcal{R}$ is of type-1 if it is above $\mathcal{P}_\mathcal{B}$, and of type-2 otherwise. Analogously, we say that a blue point q below $\mathcal{P}_\mathcal{B}$ is of type-1 if it is below $\mathcal{P}_\mathcal{R}$, and of type-2 otherwise. Project the red points above $\mathcal{P}_\mathcal{R}$ and the blue points below $\mathcal{P}_\mathcal{B}$ on the xy-plane (Figure 2.15 a)).

Then, $(\mathcal{R}, \mathcal{B})$ is determined by the corner-type pair of quadrants $(\mathcal{Q}_\mathcal{R}, \mathcal{Q}_\mathcal{B})$ in the xy-plane such that: the number of red points of type-1 in $\mathcal{Q}_\mathcal{R}$, plus the number of blue points of type-1 in $\mathcal{Q}_\mathcal{B}$, plus the number of red points of type-2 in $\mathcal{Q}_\mathcal{R} \setminus \mathcal{Q}_\mathcal{B}$, and plus the number of blue points of type-2 in $\mathcal{Q}_\mathcal{B} \setminus \mathcal{Q}_\mathcal{R}$ is maximized (Figure 2.15 b)). Computing $(\mathcal{Q}_\mathcal{R}, \mathcal{Q}_\mathcal{B})$ is slightly different to solving the corner-type case in two dimensions (Section 2.3). Since there are $O(n^2)$ different positions for $\mathcal{P}_\mathcal{R}$ and $\mathcal{P}_\mathcal{B}$, this type of solution can be



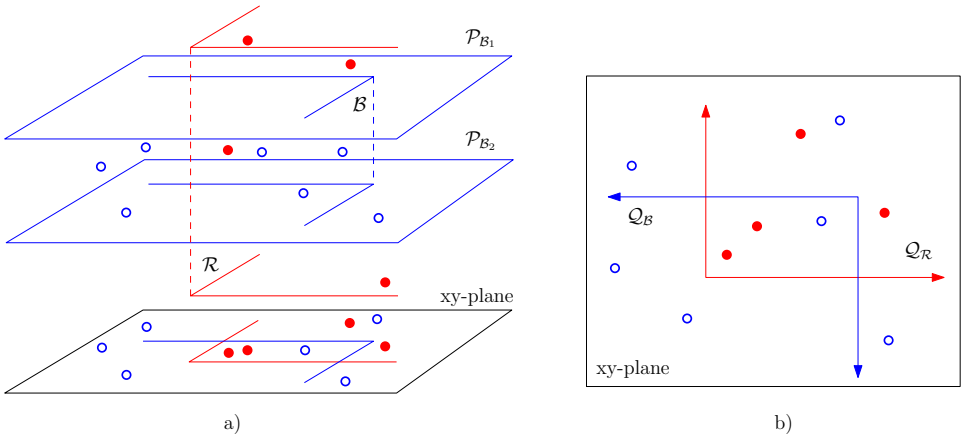
a) b)

Figure 2.15: How to compute a vertex-in-type solution.

obtained in $O(n^4 \log n)$ time and $O(n)$ space.

The edge-in-type solution

Let $\mathcal{P}_{\mathcal{B}_1}$ and $\mathcal{P}_{\mathcal{B}_2}$ be two half-planes that are parallel to the xy-plane, and such that $\mathcal{P}_{\mathcal{B}_1}$ is above $\mathcal{P}_{\mathcal{B}_2}$ in the direction given by the z-axis. Suppose we want to find the edge-in-type solution $(\mathcal{R}, \mathcal{B})$ such that the top and bottom facets of $\mathcal{B}$ are contained in $\mathcal{P}_{\mathcal{B}_1}$ and $\mathcal{P}_{\mathcal{B}_2}$ respectively, and that an edge of $\mathcal{B}$ is inside $\mathcal{R}$. We say that a red point p is of type-1 if it is either above $\mathcal{P}_{\mathcal{B}_1}$ or below $\mathcal{P}_{\mathcal{B}_2}$, and of type-2 otherwise. Project on the xy-plane all the red points and also the blue points that are both below $\mathcal{P}_{\mathcal{B}_1}$ and above $\mathcal{P}_{\mathcal{B}_2}$ (Figure 2.16 a)).



a) b)

Figure 2.16: How to compute an edge-in-type solution.

Then $(\mathcal{R}, \mathcal{B})$ is determined by the corner-type pair of quadrants $(\mathcal{Q}_\mathcal{R}, \mathcal{Q}_\mathcal{B})$

in the xy-plane such that: the number of red points of type-1 in $\mathcal{Q}_\mathcal{R}$, plus the number of red points of type-2 in $\mathcal{Q}_\mathcal{R} \setminus \mathcal{Q}_\mathcal{B}$, and plus the number of blue points in $\mathcal{Q}_\mathcal{B} \setminus \mathcal{Q}_\mathcal{R}$ is maximized (Figure 2.16 b)). Computing $(\mathcal{Q}_\mathcal{R}, \mathcal{Q}_\mathcal{B})$ is quite similar to solving the corner-type case in two dimensions (Section 2.3). This type of solution can be obtained in $O(n^4 \log n)$ time and $O(n)$ space since there are $O(n^2)$ different positions for $\mathcal{P}_{\mathcal{B}_1}$ and $\mathcal{P}_{\mathcal{B}_2}$.

The piercing-type solution

The solution here is very similar to the solution of the sandwich-type case in two dimensions (Section 2.3). Let $\mathcal{P}_{\mathcal{B}_1}$ and $\mathcal{P}_{\mathcal{B}_2}$ be two half-planes that are parallel to the xy-plane, in such a way $\mathcal{P}_{\mathcal{B}_1}$ is above $\mathcal{P}_{\mathcal{B}_2}$ in the direction given by the z-axis. Suppose we want to find the piercing-type solution $(\mathcal{R}, \mathcal{B})$ such that the top and bottom facets of $\mathcal{B}$ are contained in $\mathcal{P}_{\mathcal{B}_1}$ and $\mathcal{P}_{\mathcal{B}_2}$ respectively, and that $\mathcal{R}$ pierces $\mathcal{B}$. We fix $\mathcal{B}$ to be the space in between $\mathcal{P}_{\mathcal{B}_1}$ and $\mathcal{P}_{\mathcal{B}_2}$. Now, we assign weight $+1$ to the red points that are either above $\mathcal{P}_{\mathcal{B}_1}$ or below $\mathcal{P}_{\mathcal{B}_2}$, weight -1 to the blue points in $\mathcal{B}$, and weight 0 to the other points. Project on the xy-plane all the weighted points, as depicted in Figure 2.17 a).

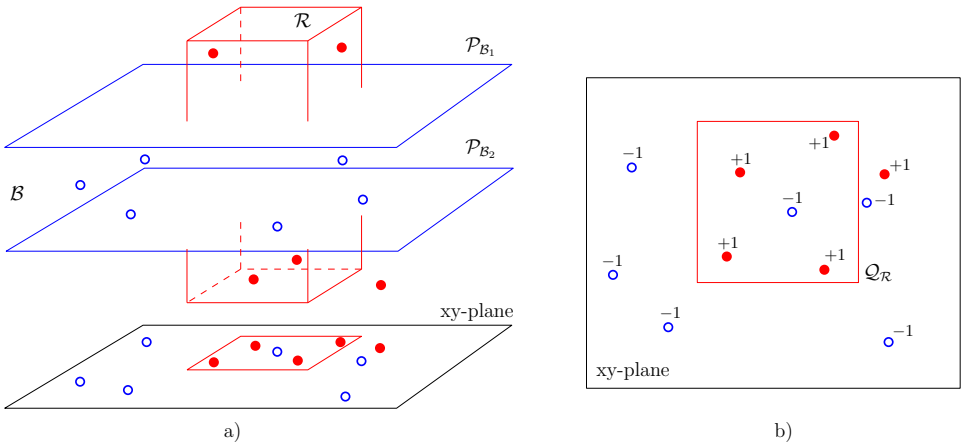


Figure 2.17: How to compute an piercing-type solution.

Then the best $\mathcal{R}$ for the fixed $\mathcal{B}$ is determined by the box $\mathcal{Q}_\mathcal{R}$ in the xy-plane that has maximum weight (Figure 2.17 b)). This problem in the plane is *The Maximum Weighted Box problem* that was solved in Section 2.6 in $O(n^2 \log n)$ time and $O(n)$ space. Since there are $O(n^2)$ different positions for $\mathcal{P}_{\mathcal{B}_1}$ and $\mathcal{P}_{\mathcal{B}_2}$, a piercing-type solution can be found in $O(n^4 \log n)$ time and $O(n)$ space.

The crossing-type solution

Let $\mathcal{P}_{\mathcal{B}_1}$ and $\mathcal{P}_{\mathcal{B}_2}$ be two half-planes that are parallel to the xy-plane, and such that $\mathcal{P}_{\mathcal{B}_1}$ is above $\mathcal{P}_{\mathcal{B}_2}$ in the direction given by the z-axis. Suppose we want to find the crossing-type solution $(\mathcal{R}, \mathcal{B})$ such that the top and bottom facets of $\mathcal{B}$ are contained in $\mathcal{P}_{\mathcal{B}_1}$ and $\mathcal{P}_{\mathcal{B}_2}$ respectively. We say that a red point p is of type-1 if it is either above $\mathcal{P}_{\mathcal{B}_1}$ or below $\mathcal{P}_{\mathcal{B}_2}$, and of type-2 otherwise. Project on the xy-plane all the red points and also the blue points that are both below $\mathcal{P}_{\mathcal{B}_1}$ and above $\mathcal{P}_{\mathcal{B}_2}$ (Figure 2.18 a)).

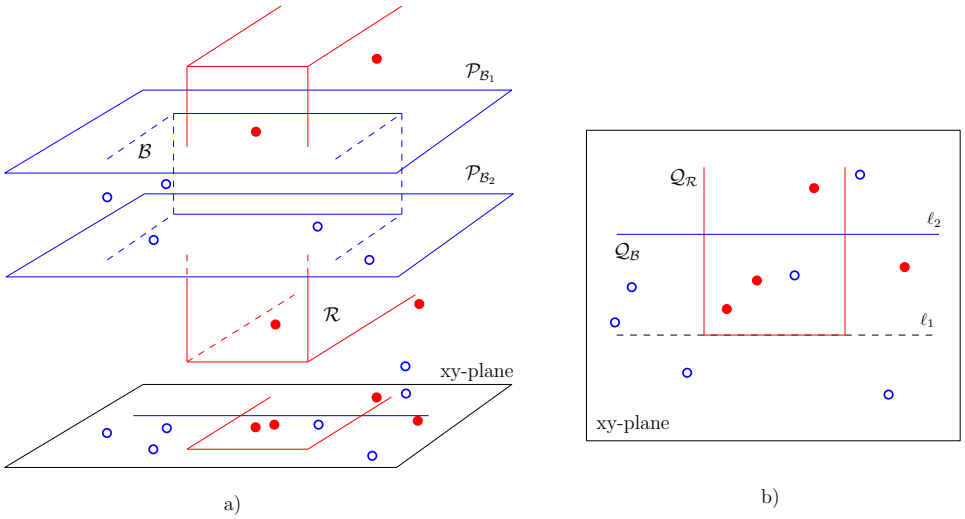


Figure 2.18: How to compute an crossing-type solution.

Then, the problem is reduced to finding in the xy-plane an isothetic half-plane $\mathcal{Q}_{\mathcal{B}}$ and an isothetic half-strip $\mathcal{Q}_{\mathcal{R}}$ such that: the two parallel sides of $\mathcal{Q}_{\mathcal{R}}$ intersect the boundary of $\mathcal{Q}_{\mathcal{B}}$; the two vertices of $\mathcal{Q}_{\mathcal{R}}$ belong to the interior of $\mathcal{Q}_{\mathcal{B}}$; and the number of blue points in $\mathcal{Q}_{\mathcal{B}} \setminus \mathcal{Q}_{\mathcal{R}}$, plus the number of red points of type-1 in $\mathcal{Q}_{\mathcal{R}}$, and plus the number of red points of type-2 in $\mathcal{Q}_{\mathcal{R}} \setminus \mathcal{Q}_{\mathcal{B}}$ is maximized. This problem in two dimensions can be solved as follows.

Let ℓ_1 and ℓ_2 be two isothetic parallel lines and consider w.l.o.g. that they are horizontal and that ℓ_1 is below ℓ_2. Suppose now that $\mathcal{Q}_{\mathcal{B}}$ is the lower half-plane defined by ℓ_2 and that the finite side of $\mathcal{Q}_{\mathcal{R}}$ lies on ℓ_1 (Figure 2.18 b)). Take a copy S' of S and re-color black all its elements except the blue points in between ℓ_1 and ℓ_2, the red points of type-1 above ℓ_1 and the red points of type-2 above ℓ_2. Now project the elements of S' on ℓ_1 obtaining a sequence X of red, blue and black elements. Then, given the current location of $\mathcal{Q}_{\mathcal{B}}$, the best $\mathcal{Q}_{\mathcal{R}}$ is determined by the two indexes i and

j ($i < j$) of X such that: the number of blue elements before index i, plus the number of red elements from index i to index j, and plus the number of blue elements after index j is maximized. By using the generalization approach of Section 2.6, this problem in one dimension can be dynamically solved in $O(\log n)$ time when the color of some element changes. Note that the same problem was solved in the solution of the *Weak Strip Separation problem* presented in Section 2.6. Fixing ℓ_1, the above one-dimensional subproblem is dynamically solved when we sweep with ℓ_2 the points of S above ℓ_1. This is done in $O(n \log n)$ time and $O(n)$ space in total. Since there are $O(n)$ different positions for ℓ_1 the total complexity of the two-dimensional subproblem is $O(n^2 \log n)$ time and $O(n)$ space. This implies a total $O(n^4 \log n)$ time and $O(n)$ space for the problem in three dimensions because there are $O(n^2)$ different positions for $\mathcal{P}_{\mathcal{B}_1}$ and $\mathcal{P}_{\mathcal{B}_2}$.

The disjoint-type solution

Solving this case is as straightforward as solving the disjoint-type case in two dimensions because there is always an isothetic plane that separates the boxes. It can be done by sweeping S with a plane in each direction given by the coordinate axes. The complexity is $O(n \log n)$ time and $O(1)$ space.

Summarizing the complexity of the five above cases, the following result is obtained:

Theorem 2.14 *The 2-EB-problem in three dimensions can be solved in $O(n^4 \log n)$ time and $O(n)$ space.*

2.8 Conclusions and open problems

In this chapter we have shown the connection between maximum boxes problems and Bentley's maximum consecutive sum problem. We have developed a dynamic data structure that allows us to maintain the solution of the Bentley's maximum consecutive sum problem in $O(\log n)$ time when an element of the sequence changes its value. This data structure can be computed in linear time. The key idea used to dynamically solve Bentley's maximum consecutive sum problem was extended as a general technique useful for solving other data analysis problems.

A natural problem, that requires further research, is the design of more efficient algorithms to the three-dimensional case. As we have seen, by using projections of the bicolored point set on the plane, the solutions to

the vertex-in-type and edge-in-type cases can be reduced to solve $O(n^2)$ instances of the corner-type-case of the 2-EB-problem in $\mathbb{R}^2$. Those projections are determined by parallel planes and we can sweep the elements of S with those planes and pass from an instance I in $\mathbb{R}^2$ to another I' in such a way I and I' differ by only one point (i.e. a point is added, removed or its color is changed). Then, in order to reduce the time complexity, it would be interesting to design a dynamic data structure that permits us to dynamically compute corner-type solutions in $\mathbb{R}^2$ when we add or remove a point, and also when we change the color of a point. A good time complexity for this dynamic operations would be $o(n \log n)$.

The solution to the piercing-type case of the 2-EB-problem in $\mathbb{R}^3$ is obtained by solving $O(n^2)$ instances of the *Maximum Weighted Box problem*. In order to reduce the time complexity, we leave to further research the problem of dynamically computing in $o(n \log n)$ time the maximum weighted box when the weight of a point changes. The *Maximum Weighted Box problem* can be seen as a two-dimensional version of the Bentley's maximum consecutive sum problem, but by considering a set of weighted points instead of a two-dimensional array [98, 99].

Finally, in order to study the complexity of the 2-EB-problem, it is worthy to know if the 2-EB-problem in $\mathbb{R}^2$ is 3SUM-hard [55].

Chapter 3

The Maximum Box Problem for Moving Points on the Plane

In Pattern Recognition and Classification, a natural method for selecting prototypes that represent a class is to perform cluster analysis on the training data [46]. Typically, two sets of points are given, and one would like to find patterns which intersect exactly one of these sets. The clustering can be obtained by using simple geometric shapes such as disks or boxes. A basic problem is the so-called *Maximum Box Problem*, where the clustering is done by considering maximum boxes, that is, boxes containing the maximum number of points in the given data set (Figure 3.1). See [83, 95] for example. This problem was introduced in Subsection 2.6.2 as the MB-problem.

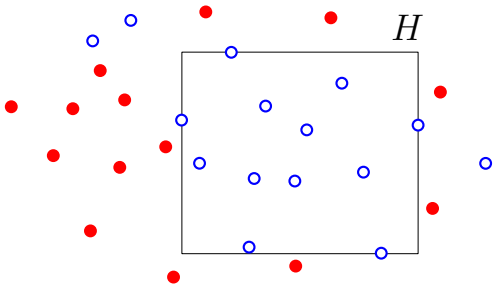


Figure 3.1: The box H is the maximum box. It contains the maximum number of blue points and does not contain any red point.

In many applications, data are given in a dynamic scenario [21, 33], that is, they change over time. Motivated by this concept, in this chapter we in-

troduce and investigate the dynamic version of the MB-problem, where the dataset is modeled by points moving along bounded degree algebraic trajectories. We present a *Kinetic Data Structure* (KDS) to efficiently maintain the maximum box for a set of bicolored moving points. A KDS is used to keep track of the attributes of interest in a system of moving objects [61]. The main idea in the kinetic framework is that even though the objects move continuously, the relevant combinatorial structure changes only at certain predictable discrete events. Therefore, one does not have to update the data structure continuously.

The chapter is organized as follows. In Section 3.1 we present the KDS framework. In Section 3.2 we give an algorithm for the static version of the MB-problem that will be useful for the dynamic version. A new data structure for maintaining the maximum box in a kinetic setting is proposed in Section 3.3. In Section 3.4, an algorithm for the MB-problem in a static setting, in which the box can be arbitrarily oriented, is presented. In Section 3.5 we present an efficient data structure to maintain an approximated solution of the MB-problem when the points move. The number of blue points contained in our approximated solution is at least half of the number of blue points contained in an exact solution. Finally, in Section 3.6, the conclusions and further research are presented.

3.1 The KDS framework

In this section the KDS framework is presented. The *Kinetic Data Structures* (KDS) were introduced by Basch et al. [13]. In the following, we cite definitions, information, and examples, taken from [5, 13, 60, 61].

A KDS is a data structure that maintains a certain attribute of a set of continuously moving objects [61]. It consists of two parts: a combinatorial description of the attribute and a set of certificates. The certificates are elementary tests on the input objects with the property which, as long as their outcomes do not change, the attribute does not change either. The main idea in the kinetic framework is that even though the objects move continuously, the relevant combinatorial structure changes only at certain predictable discrete events. Therefore, one does not have to update the data structure continuously. These events have a natural interpretation in terms of the underlying structure, for example, when the x- or y- projections of two points coincide. All the events are managed in an event queue, and they are pairs $\langle c, t\rangle$, where c is the certificate and t is the failure time of c. Events in which the attribute changes are called *external events*, other events are

called *internal events*. Any moving object is allowed to change its motion, and when this happens, all certificates involving the object must re-evaluate their failure times and the event queue is updated.

A good, very simple example, is a KDS that maintains the rightmost point (the point with maximum x-coordinate) in a set S of n continuously moving points on the plane. The simplest approach is to sort S by x-coordinate and store it in a list P, thus the rightmost point is given by the last element of P. There is a certificate (the x-order condition) for every two consecutive elements in P, and we have a certificate failure every time two consecutive elements in P, p_i and p_{i+1}, have the same x-coordinate. Then, p_i and p_{i+1} are swapped in P, and at most three new certificates arise involving p_{i-1}, p_i, p_{i+1}, and p_{i+2}.

The performance of a KDS is measured according to four properties. A KDS is *compact* if the total number of stored certificates is near-linear in the total number of objects (say $O(n^{1+\varepsilon})$, where n is the number of objects), and *local* if no object participates in *too many* (say $O(n^{\varepsilon})$) certificates, that is, at any one time, the maximum number of events in the event queue that depend on a single object is small. A KDS with this property can be updated quickly when the flight plan (motion) of an object changes. A KDS is *responsive* if the time needed to update it (as well the event queue), when a certificate fails, is poly-logarithmic in the problem size, and a KDS is *efficient* if the ratio of the maximum total number of internal and external events, to the maximum total number of external events, is poly-logarithmic in the problem size. Analyzing the efficiency of a KDS is slightly more complicated than the other performance measures.

Revisit above example of a KDS that maintains the rightmost point. The KDS is local since every point is involved in at most two certificates, and hence is compact, and it is responsive since the list P is updated in constant time and the event queue can be managed by an efficient priority queue in the failure time [35]. Finally, see that the KDS is not efficient. For instance, if the points move in the same line, then the total number of external events (those that change the rightmost element) can be linear, and the total number of internal events can be quadratic. This kinetic problem was efficiently solved in [13] with the well-known *kinetic tournament*.

As we have mentioned, the KDS framework encounters applications in many problems concerning extent, proximity, collision detection, connectivity, and clustering, etc. of moving objects [60].

3.2 The static version of the Maximum Box Problem

The static version of the MB-problem in the plane has already been studied. Liu et al. [83] give an $O(b^2 \log b + br + r \log r)$-time algorithm. The same problem was solved in Subsection 2.6.2 in $O(n^2 \log n)$ time and $O(n)$ space. Unfortunately, the approach used in Subsection 2.6.2 could not be applied for the dynamic environment.

In the following we put forward for consideration a simple algorithm, that is similar to the algorithm in [83], and with a slightly different complexity, which will be used later in Section 3.3 to design a data structure for the kinetic version. That is why, as we will see below, in our variant we elaborate more than in [83].

First, we observe that a maximum box for R and B can be enlarged until each of its sides either contains a red point or reaches infinity. Thus, the maximum box can be transformed to one of the following isothetic objects with red points on its boundary and no red points inside: a rectangle, a half-strip, a strip, a quadrant or a half-plane (Figure 3.2).

We show how to compute the maximum box when it is a rectangle, a half-strip or a strip. The cases of a quadrant and a half-plane are even easier. The case of a quadrant can be addressed by applying the techniques based on the *Dynamic Bentley's Maximum Subsequence Sum Problem* presented in Chapter 2. Both cases can be solved in $O(n \log n)$ time (see the MWB-problem in Subsection 2.6.2). Actually, both can be solved by using our below approach if we allow sweeping to infinity, that is, the sweep stops once all points in $R \cup B$ are swept.

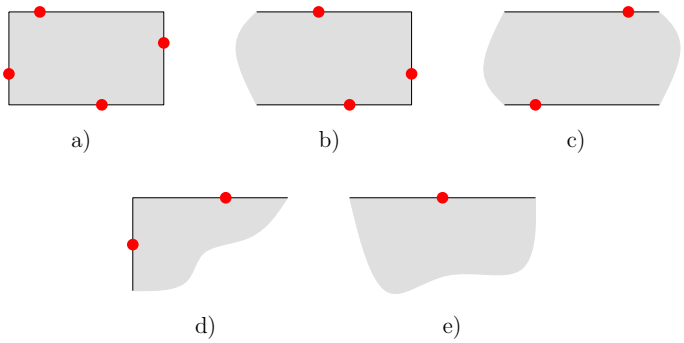


Figure 3.2: Configurations of the maximum box. a) A rectangle, b) a half-strip, c) a strip, d) a quadrant, e) a half-plane.

We proceed as follows. For every red point p, we compute a rectangle $H(p)$ which satisfies the following:

(i) the top side of $H(p)$ contains p,
(ii) the interior of $H(p)$ contains the maximum number of blue points and no red points, and
(iii) the boundary of $H(p)$ contains only red points.

Then, we take the best of all $H(p)$, that is, the one that contains the maximum number of blue points. To do this, for each red point p, we draw a horizontal line ℓ passing through p and sweep the points below ℓ by moving ℓ downwards (Figure 3.3).

Let h_p be the horizontal line that passes through p. During the sweep, we maintain two balanced binary search trees T_R and T_B such that T_R (resp. T_B) contains all the red (resp. blue) points that lie between h_p and ℓ sorted by x-coordinate. In T_B, for each node u, we also maintain a counter of the number of blue points on the subtree rooted at u. When ℓ passes through a point q we perform the following. Let left(q) (resp. right(q)) be the rightmost (resp. leftmost) red point in T_R located to the left (resp. right) of the vertical line that passes through q, and let x_{left} (resp. x_{right}) be its x-coordinate. If left(q) (resp. right(q)) does not exist then x_{left} (resp. x_{right}) is $-\infty$ (resp. $+\infty$). We only process q if the x-coordinate of p lies in the interval $(\mathrm{x}_{left}, \mathrm{x}_{right})$. In that case, q is inserted in T_R or in T_B according to its color and, if q is a red point, then we consider a candidate to $H(p)$ the isothetic rectangle (half-strip or strip) H_q whose sides pass through left(q), right(q), p and q. Note that H_q is a half-strip if either left(q) or right(q) does not exist, and that H_q is a strip if neither of them exists. The number of blue points inside H_q (i.e. $|\operatorname{Blue}(H_q)|$) is equal to the number of blue points in T_B whose x-coordinate lies in (x_{left}, x_{right}). At the end of the sweep, we consider the candidate to $H(p)$ whose bottom side is at infinity (i.e. $H(p)$ is a vertical top-bottom half-strip, quadrant, or half-plane), that is, we stop the sweeping line ℓ at a dummy red point q that is in the same vertical as p and below every point in $R \cup B$. It is equivalent to say that ℓ stops at infinity.

In above procedure, left(q), right(q), and the number of blue points inside H_q, are obtained in logarithmic time each [35]. Then, for every red point q below h_r we obtain: both left(q) and right(q) in $O(\log r)$ time, and $|\operatorname{Blue}(H_q)|$ in $O(\log b)$ time. Inserting each blue point below h_p in T_B takes $O(\log b)$ time. Thus each top-bottom sweep from a red point p is done in $O(r \log r + r \log b + b \log b)$ time.

The overall process requires, firstly, to sort S by y-coordinate in $O(r \log r + b \log b)$ time (i.e. first, sort R, after that sort B, and finally merge the sorted

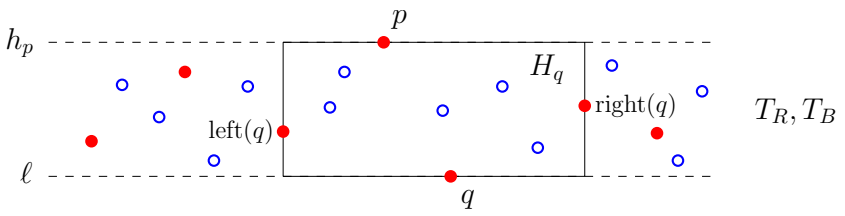


Figure 3.3: Algorithm for the static case. The line ℓ sweeps all the points below h_p. The trees T_R and T_B maintain respectively all red and blue points lying between h_p and ℓ. Both left(q) and right(q) are obtained from T_R, and the number of blue points in the box H_q is computed from T_B. The box H_q is a half-strip if either left(q) or right(q) does not exist, and H_q is a strip if neither of them exists.

lists of R and B) in order to do the sweeps, thus the total space complexity is $O(r+b)$ and the total time complexity is:

$$\begin{aligned} & O(r\log r + b\log b) + r\cdot O(r\log r + r\log b + b\log b) \\ = \; & O(r\log r + b\log b) + O(r^2\log r + r^2\log b + rb\log b) \\ = \; & O(r^2\log r + r^2\log b + rb\log b) \end{aligned}$$

In order to consider in the above procedure all types of boxes as depicted in Figure 3.2, we can repeat it in the bottom-top direction, and work symmetrically. It remains to show that the algorithm is correct. This can be shown by taking an optimal axis-parallel rectangle and enlarging it (horizontally and vertically) until it hits red points or reaches infinity. The algorithm checks all such boxes and, therefore, is correct.

Notice we can omit the use of T_R in above procedure if we apply the same idea of the algorithm in [83]. Namely, initially set $(\mathrm{x}_{left}, \mathrm{x}_{right}) = (-\infty, +\infty)$ and $(q_l, q_r) = (\text{null}, \text{null})$, and every time the sweeping line ℓ encounters a red point q, such that $\mathrm{x}(q)$ is in $(\mathrm{x}_{left}, \mathrm{x}_{right})$, set $\text{left}(q) = q_l$ and $\text{right}(q) = q_r$. After q is processed, set $\mathrm{x}_{left} = \mathrm{x}(q)$ and $q_l = q$ if $\mathrm{x}(q) < \mathrm{x}(p)$, and $\mathrm{x}_{right} = \mathrm{x}(q)$ and $q_r = q$ otherwise. With this, the time and space complexities reduce to $O(r^2\log b + rb\log b)$ and $O(b)$, respectively. Nevertheless, we keep using T_R because it is useful in the kinetic version.

In many real applications data is imbalanced [56, 59, 75], that is, there are many more instances of one class than the other. In those where the negative data (the red points) has few elements with, respect to the positive data (the blue points), meaning that $r \ll b$, our time complexity is essentially $O(b\log b)$ and it is better than the complexity of [83]. The contrary happens when the negative data are the majority.

3.2.1 The Smallest-Area Maximum Box Problem

In this subsection we consider the problem of computing the box of smallest area that maximizes the number of covered points from B and does not contain any point from R. We refer to this problem as the SAMB-problem. Segal [95] designed an algorithm for the SAMB-problem with $O(n^3 \log^4 n)$ running time.

We show that our approach can be applied to solve the SAMB-problem. Note that none of the boxes $H(p)$ is the solution of the SAMB-problem since every $H(p)$ contains red points on its boundary (or the boundary extends to infinity). We augment the tree T_B as follows. For each node u in T_B, we store the smallest and the largest y-coordinates of the points in the subtree rooted at u. Since the smallest and the largest x-coordinates of these points can be computed by using the x-order, the smallest bounding box of the blue points in $H(p)$ can be computed in $O(\log b)$ time. The additional cost is $O(\log b)$ per one sweep step, and the asymptotic overall time complexity does not change.

Finally, the following theorem is obtained,

Theorem 3.1 *The SAMB-problem can be solved in* $O(r^2 \log r + r^2 \log b + rb \log b)$ *time by using* $O(r + b)$ *space.*

Note that our algorithm improves the running time of the previous algorithm in [95] by a factor of $O(n \log^3 n)$.

3.3 The Maximum Box Problem for moving points

In this section we introduce a new kinetic data structure (KDS), for maintaining the maximum box. First of all, we notice that the optimal solution is not unique, and furthermore, that when the points move, a few changes can make the number of optimal solutions increase or decrease in an $O(n)$ factor. For example, suppose that $r = 2k+2$ and $b = 2k$, and distribute two sets of bicolored points S_1 and S_2, with $k+1$ red points and k blue points each, as depicted in Figure 3.4 a). Then, we have an optimal solution with the value of 2 for every two consecutive red points in S_1 and two consecutive red points in S_2, thus $k^2 = O(n^2)$ in total. Now, if two consecutive red and blue points in S_1 change (Figure 3.4 b)), then there is an optimal solution that values 3 for every two pairs of consecutive red points in S_2, and they are $k = O(n)$ in total. Finally, if two consecutive red and blue

points in S_2 exchange their positions (Figure 3.4 c)), then there is only one optimal maximum box with the value of 4. Therefore, with a few changes in the distribution of points, the number of different solutions of the MB-problem can decrease (or increase if we reverse the process) in at least an $O(n)$ factor. From this information, we also note that the maximum box for moving points might not change continuously over time, and that designing an efficient and responsive KDS could be hard.

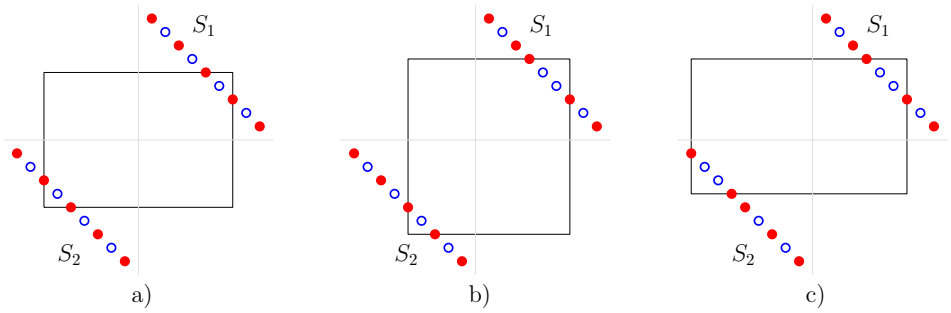


Figure 3.4: Each of the sets S_1 and S_2 consists of $k+1$ red points and k blue points. The number of different maximum boxes is: a) $k^2 = O(n^2)$, b) $k = O(n)$, c) one. Note that configuration b) is obtained from a) with a few changes (and viceversa), as well c) is obtained from b).

Let X (resp. Y) be the elements of $R \cup B$ sorted by abscissa (resp. ordinate), that is, X and Y are the x-order and the y-order of $R \cup B$. To make the presentation of the next results clear, we assume that no two points in $R \cup B$ have the same x- or y-coordinate. This condition can be avoided by using standard techniques. Our KDS design is based on the following lemma.

Lemma 3.2 *The maximum box for R and B is univocally determined by X and Y, and it does not change combinatorially over time as long as X and Y do not.*

Proof. Suppose we do not know the exact x- and y-coordinates of the points in R and B, but do know X and Y. Next, we can compute a maximum box for R and B by considering that i is the x-coordinate of X_i and the y-coordinate of Y_i, where X_i and Y_i denote the i-th element of X and Y, respectively. ■

We design a KDS called *Maximum Box Kinetic Data Structure* (MBKDS). In our problem, the set of moving objects is $R \cup B$ and the attribute is one of the maximum boxes. According to Lemma 3.2, we consider as a certificate the condition that two consecutive points in X (resp. Y) satisfy the x-order

(resp. y-order). An event is the failure of a certificate at some instant of time t, and each event implies an update of the MBKDS. We call an event a *flip*.

We obtain the following result.

Lemma 3.3 *The MBKDS is compact and local.*

Proof. The total number of certificates is $2n - 2 = O(n)$ since there is a certificate for each two consecutive elements in X and Y. Thus, the MBKDS is compact. The MBKDS is local since any point p in $R \cup B$ is involved in at most four certificates, that is, at most two in the x-order and at most two in the y-order. ■

The MBKDS essentially consists of the event queue $\mathcal{E}$ and the structure $\mathcal{D} = \{\mathcal{D}(p) \mid p \in R\}$, where $\mathcal{D}(p)$ is a data structure associated with the red point p. When an event, corresponding to the certificate of two consecutive elements of X, say $[X_i, X_{i+1}]$, occurs, it is removed from $\mathcal{E}$. Then, we remove from $\mathcal{E}$ the events corresponding to $[X_{i-1}, X_i]$ and $[X_{i+1}, X_{i+2}]$. After that, we swap X_i and X_{i+1}, and insert in $\mathcal{E}$ new events (at most three) for $[X_{i-1}, X_i]$, $[X_i, X_{i+1}]$, and $[X_{i+1}, X_{i+2}]$. The event time is computed based on our knowledge of the motions of X_{i-1}, X_i, X_{i+1} and X_{i+2}. The event queue $\mathcal{E}$ is designed as a priority queue in the failure time with its basic operations in logarithmic time [35]. The certificates $[Y_i, Y_{i+1}]$ are treated similarly.

The data structure $\mathcal{D}$ focuses on finding the maximum box whose four sides pass through red points, although boxes of other types are considered as a by-product of its own design. Following the ideas of $\mathcal{D}$, simpler data structures for half-strips, strips, quadrants, or half-planes can be designed, in such a way their space and event time processing complexities are smaller than the corresponding complexities of $\mathcal{D}$.

The definition and details of $\mathcal{D}(p)$ are as follows. First, consider the horizontal line h_p passing through the red point p, and let S_p be the subset of points of S lying below h_p (Figure 3.5). For each q in $\mathrm{Red}(S_p)$ we define:

- $\mathrm{V}(q)$ as the vertical half-line for which q is its top-most point.
- $\mathrm{I}(q)$ as the maximum-length horizontal segment that contains q and does not intersect any other $\mathrm{V}(q')$ for q' in $\mathrm{Red}(S_p) \setminus \{q\}$.
- $\mathrm{H}(q)$ as the rectangle whose bottom side is $\mathrm{I}(q)$ and its top side is contained in h_p.

- left(q) (resp. right(q)) as the rightmost (resp. leftmost) red point in S_p located to the left (resp. right) of the vertical line that passes through q and so that y(left(q)) (resp. y(right(q))) is greater than y(p) (Figure 3.5). Notice that the left and the right points of I(q) lie on V(left(q)) and V(right(q)), respectively.

- the set Candidates(p) of the points q in Red(S_p), such that I(q) intersects the vertical line that passes through p. Observe that, according to the previous section, $H(p)$ is the box whose top side lies on h_p and its bottom side is the interval I(q) in which q is in Candidates(p) and $|\,\text{Blue}(\text{H}(q))|$ is maximized. We say that a point q in Red(S_p) is candidate if and only if q is in Candidates(p).

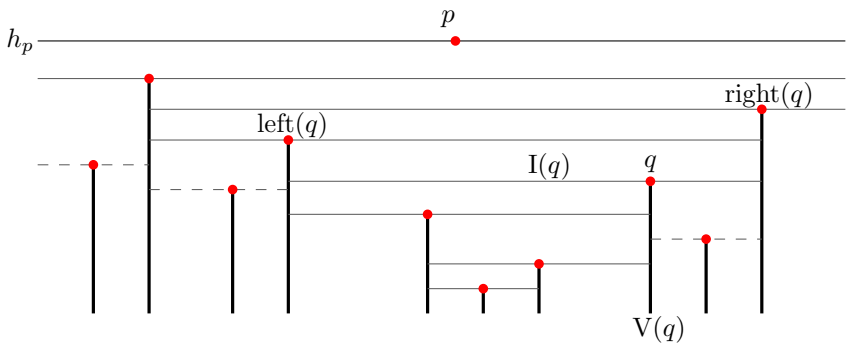


Figure 3.5: $\mathcal{D}(p)$. Candidate points are the points q such that I(q) is drawn with a continuous line.

The key idea for $\mathcal{D}(p)$ is to dynamically compute $H(p)$ when a flip, which involves two points below h_p, occurs.

The sequence $(q_1, q_2, \ldots, q_m)$ of the elements of Candidates(p) in decreasing order of y-coordinate are from left to right the leaves of a dynamic balanced binary search tree $\mathcal{Q}$. Every node u of $\mathcal{Q}$ is labeled with α such that for all q_i in Candidates(p), $|\,\text{Blue}(\text{H}(q_i))|$ is equal to the sum of the α labels of the nodes in the path from q_i to the root. In addition, every internal node u is labeled with β whose value is the candidate q' in the subtree rooted at u that maximizes $|\,\text{Blue}(\text{H}(q'))|$. In this way, the label β of the root is the candidate red point that determines $H(p)$. The idea in the design of $\mathcal{Q}$ is to maintain a priority queue over $(q_1, q_2, \ldots, q_m)$ (where the priority of every q_i is $|\,\text{Blue}(\text{H}(q_i))|$) with the following operations: (i) compute in constant time the element with maximum priority, (ii) given q_i compute its priority in $O(\log m)$ time, and (iii) given indexes i and j ($i \leq j$) increase or decrease, in the same amount, the priorities of $q_i \ldots, q_j$ in $O(\log m)$ time.

The operation (i) is done by returning the β label of the root of $\mathcal{Q}$. To carry on the operation (ii) we traverse the path from q_i to the root of $\mathcal{Q}$ and return the sum of the α labels. The operation (iii) can be done as follows. Take two pointers l_1 and l_2 pointing respectively to the corresponding nodes of q_i and q_j. Modify in $\Delta\alpha$ the α labels of l_1 and l_2 and repeat, while parent(l_1) is not equal to parent(l_2), the following: if l_1 is a left child then modify in $\Delta\alpha$ the α label of the right child of parent(l_1), make l_1 point to parent(l_1), and update the β label of l_1 (do symmetrically with l_2). After the loop has ended, make a traversal from parent(l_1) to the root updating the β labels.

The elements of $\mathrm{Red}(S_p)$ sorted in increasing order of x-coordinate are from left to right the leaf nodes of a balanced binary search tree T_R. Every node u in T_R is labeled with y_{max}, that is, the element that has the maximum ordinate on the subtree rooted at u. It enables us to obtain in logarithmic time left(q) and right(q) for all q in $\mathrm{Red}(S_p)$ as follows. In order to obtain right(q), we first locate the leaf l containing q by applying a binary search, and find the deepest ancestor u of l such that: l is located on the left subtree of u, and the y_{max} label of u is greater than $\mathrm{y}(q)$. Let l_u be a pointer to u. After that, set l_u equal to its right child and perform, while l_u is not a leaf, the following: if $\mathrm{y}(q)$ is less than the y_{max} label of the left child of l_u, then set l_u equal to its left child, otherwise make l_u equal to its right child. When the loop ends, return the point stored at the leaf pointed by l_u. We proceed symmetrically to obtain left(q).

For each q in $\mathrm{Red}(S_p)$, we divide $\mathrm{I}(q)$ at q, obtaining the horizontal segments $\mathrm{I}_{left}(q)$ and $\mathrm{I}_{right}(q)$ (Figure 3.6). Define $\mathrm{H}_{left}(q)$ (resp. $\mathrm{H}_{right}(q)$) as the box whose bottom side is $\mathrm{I}_{left}(q)$ (resp. $\mathrm{I}_{right}(q)$) and its top side is contained in h_p. Let $\mathrm{T}_{left}(q) = \{\mathrm{I}_{right}(q') \mid q' \in \mathrm{Red}(S_p) \setminus \{q\} \wedge \mathrm{right}(q') = q\}$ and $\mathrm{T}_{right}(q) = \{\mathrm{I}_{left}(q') \mid q' \in \mathrm{Red}(S_p) \setminus \{q\} \wedge \mathrm{left}(q') = q\}$. The set $\mathrm{T}_{left}(q)$ is represented in a dynamic balanced binary search tree where the leaves from left to right are its elements $\mathrm{I}_{right}(q')$ in decreasing order of $\mathrm{y}(q')$. Every node u is labeled with α as was done for $\mathcal{Q}$. In this case, $|\,\mathrm{Blue}(\mathrm{H}_{right}(q'))|$ is the sum of the α labels of the nodes in the path from $\mathrm{I}_{right}(q)$ to the root. Modifying in $\Delta\alpha$ the value $|\,\mathrm{Blue}(.)|$ of the elements in an interval of consecutive leaves in $\mathrm{T}_{right}(q)$ for a given q in $\mathrm{Red}(S_p)$ is done similarly as in $\mathcal{Q}$. Also, every node u is labeled with the boolean c indicating if all the elements $\mathrm{I}_{right}(q')$ in the subtree rooted at u are such that q' is a candidate. The c labels allow, given a value of ordinate y_0, to find the interval of consecutive leaves $\mathrm{I}_{left}(q')$ such that $\mathrm{y}(q') < y_0$ and q' is candidate. This operation can be done in logarithmic time as follows. Apply a binary search to find the leaf l that contains the element $\mathrm{I}_{left}(q_0)$ such that $\mathrm{y}(q_0) < y_0$

and $y(q_0)$ is maximized. If l is labeled with $c = 0$, then there is no interval. Otherwise, find the deepest ancestor u of l such that l is located on the left subtree of u, and the c label of the right child of u is equal to 0. Let l_u be a pointer to u. Set l_u equal to its right child and do, while l_u is not a leaf, the following: if the c label of the left child of l_u is equal to 0, then make l_u point to its left child, and if not, make l_u point to its right child. When the loop ends, the interval is determined by l and the predecessor leaf of l_u. Furthermore, $\mathrm{T}_{left}(q)$ is designed to support the operators JOIN[1] and SPLIT[2], both in logarithmic time [35]. The set $\mathrm{T}_{right}(q)$ is represented in the same manner.

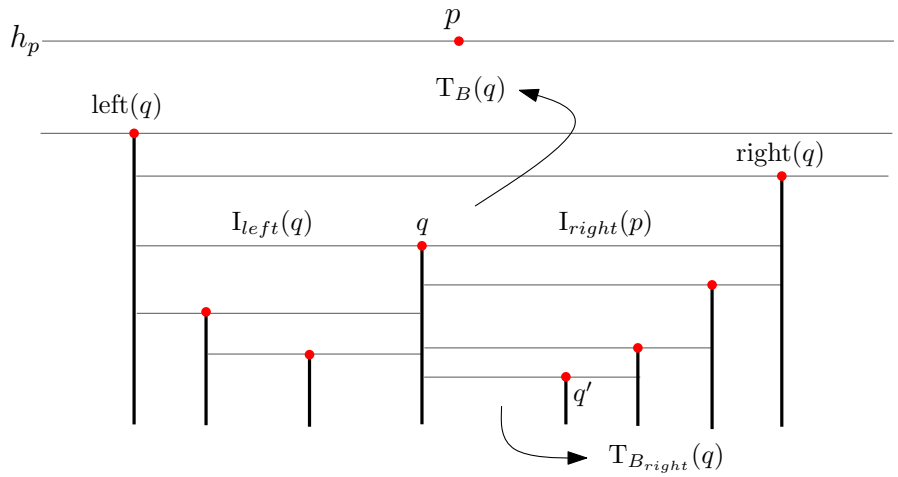


Figure 3.6: Details of $\mathcal{D}(p)$.

The set $\{\mathrm{I}(q) \mid q \in \mathrm{Red}(S_p)\} \cup \{\mathrm{V}(q) \mid q \in \mathrm{Red}(S_p)\}$ determines a partition $\mathcal{P}$ of the lower half-plane defined by h_p. For each cell $\mathcal{C}$ of $\mathcal{P}$, we store in a balanced binary search tree $\mathrm{T}_B(\mathcal{C})$ (supporting operators JOIN and SPLIT) the elements of $B \cap \mathcal{C}$ in decreasing order of y-coordinate. Each node u is labeled with the number of blue points on the subtree rooted at u in such a way the label of the root is $|B \cap \mathcal{C}|$. Each blue point q below h_p has a reference $\mathrm{T}_{\mathcal{C}}(q)$ to the tree $\mathrm{T}_B(\mathcal{C})$ where $\mathcal{C}$ is the cell that contains q. Also, we associate with each q in $\mathrm{Red}(h_p)$ the tree $\mathrm{T}_B(q)$, which is the tree of blue points that corresponds to the cell of $\mathcal{P}$ that is bounded below by $\mathrm{I}(q)$, and

[1]Given two balanced binary search trees T_1 and T_2, in which every value in T_1 is less than any value in T_2, JOIN operator constructs a balanced binary search tree by merging the nodes of T_1 and T_2.

[2]Operator SPLIT is like the inverse of JOIN. Given a balanced binary search tree T and a value x, splits the nodes of T in such a way that those nodes whose value are less than (resp. greater than) x become the nodes of a balanced binary search tree T_1 (resp. T_2).

the tree $\mathrm{T}_{B_{right}}(q)$ that corresponds to the cell of $\mathcal{P}$ bounded by $\mathrm{V}(q)$ and $\mathrm{V}(q')$, and has no bottom boundary, where q' is the element that follows q in T_R (Figure 3.6).

Each of the trees that are used in $\mathcal{D}(p)$ can be implemented by using *Red-Black-trees*, where all the elements are stored in the leaves [35]. Because of the similarity between the structure of $\mathcal{D}(p)$ and the operations of the algorithm of Section 3.2, a similar algorithm to build $\mathcal{D}(p)$ can be designed. The following result is established:

Theorem 3.4 *Given R, B and p, such that $|R| = r$, $|B| = b$, and $p \in R$, the data structure $\mathcal{D}(p)$ requires $O(r+b)$ space and can be built in $O(r \log r + r \log b + b \log b)$ time.*

Proof. Both $\mathcal{Q}$ and T_R have $O(r)$ space complexity. The total complexity of $\mathrm{I}_{left}(q)$ and $\mathrm{I}_{right}(q)$ for all q in $\mathrm{Red}(S_p)$ is $O(2|\,\mathrm{Red}(S_p)|) = O(r)$. The total space required by $\mathrm{T}_B(\mathcal{C})$ for all cells $\mathcal{C}$ of $\mathcal{P}$ is $O(\sum_{\mathcal{C}\in\mathcal{P}} |B \cap C|) = O(|B \cap (\bigcup_{\mathcal{C}\in\mathcal{P}} \mathcal{C})|) = O(b)$. Then the space complexity follows. The construction of $\mathcal{D}(p)$ can be done by making the following changes in the procedure presented in Section 3.2 for the static case. Process all points q below h_p from top to bottom by inserting q in T_R or T_B according to its color, and, if q is a red point, do the following. Obtain $\mathrm{left}(q)$ and $\mathrm{right}(q)$ in $O(\log r)$ time, compute $|\,\mathrm{Blue}(\mathrm{H}(q))|$, $|\,\mathrm{Blue}(\mathrm{H}_{left}(q))|$ and $|\,\mathrm{Blue}(\mathrm{H}_{right}(q))|$ by querying T_B in $O(\log b)$ time, and insert $\mathrm{I}_{left}(q)$ and $\mathrm{I}_{right}(q)$ in the trees $\mathrm{T}_{right}(\mathrm{left}(q))$ and $\mathrm{T}_{left}(\mathrm{right}(q))$, respectively, in $O(\log r)$ time. This part of the construction has time complexity $O(r \log r + r \log b + b \log b)$. Finally, construct $\mathrm{T}_B(q)$ and $\mathrm{T}_{B_{right}}(q)$ for all the red points q in $\mathrm{Red}(S_p)$ as follows. First, preprocess B in $O(b \log b)$ time to use $O(b \log b)$ space in order to answer orthogonal range search queries in $O(\log b + k)$ time, where k is the number of reported points [31, 40]. To construct $\mathrm{T}_B(q)$ for a given red point q in S_p, search $\mathrm{left}(q)$ and $\mathrm{right}(q)$ in $O(\log r)$ time, and then, answer an orthogonal range search query with the box H'_q whose bottom side is $\mathrm{I}(q)$ and one of its top vertices is the point with minimum y-coordinate among $\mathrm{left}(q)$ and $\mathrm{right}(q)$. The range query reports in $O(\log b + k_q)$ time the k_q blue points in H'_q, and constructing $\mathrm{T}_B(q)$ can be done in $O(k_q \log k_q)$ time. Similarly, construct $\mathrm{T}_{B_{right}}(q)$. The overall time complexity is:

$$\begin{aligned} & \sum_{q\in\mathrm{Red}(S_p)} O(\log r + \log b) + \sum_{q\in\mathrm{Red}(S_p)} O(k_q \log k_q) + \sum_{q\in\mathrm{Blue}(S_p)} O(\log b) \\ = \;& O(r \log r + r \log b + b \log b) \end{aligned}$$

■

Hence, the following theorem is obtained,

Theorem 3.5 *Given R and B, such that $|R| = r$ and $|B| = b$, the data structure $\mathcal{D}$ has space complexity $O(r^2 + rb)$ and can be built in $O(r^2 \log r + r^2 \log b + rb \log b)$ time.*

The following will show all the possible types of flips that can occur when the elements of $R \cup B$ move. We say that a flip is an x-*flip* (resp. a y-*flip*) if the involved points change their x-order (resp. y-order). Furthermore, according to the colors of the flipping points, we classify every flip in *blue-blue*, *red-blue*, or *red-red*. The details to process any of them depend on its type. In some flips we apply insertions and/or deletions of points in $\mathcal{D}(\cdot)$. As we shall see in Subsection 3.5.1, inserting or removing a blue point in a given $\mathcal{D}(p)$ can be done in $O(r + \log b)$ time, and inserting or removing a red point in $O(r + b)$ time. Furthermore, in some cases we use the operations GetPriorities and RebuildCandidates, described also in Subsection 3.5.1, that both run in $O(r)$ time.

The blue-blue x-flip

In this flip we do nothing because it does not affect the maximum box, neither the structure of $\mathcal{D}$.

The blue-blue y-flip

Two blue points p_1 and p_2 make a y-flip. Exchange p_1 and p_2 in every tree of blue points $\mathrm{T}_B(.)$ or $\mathrm{T}_{B_{right}}(.)$ that contains both.

The red-blue y-flip

A red point q_1 and a blue point q_2 make a y-flip. Consider five cases as depicted in Figure 3.7. Procedure *red-blue*-y-*flip-1* (Algorithm 3) solves case 1). Note that case 5) only occurs in one structure $\mathcal{D}(\cdot)$, and it can be processed by inserting/removing q_2 in $\mathcal{D}(q_1)$. Procedures for the other cases are similar.

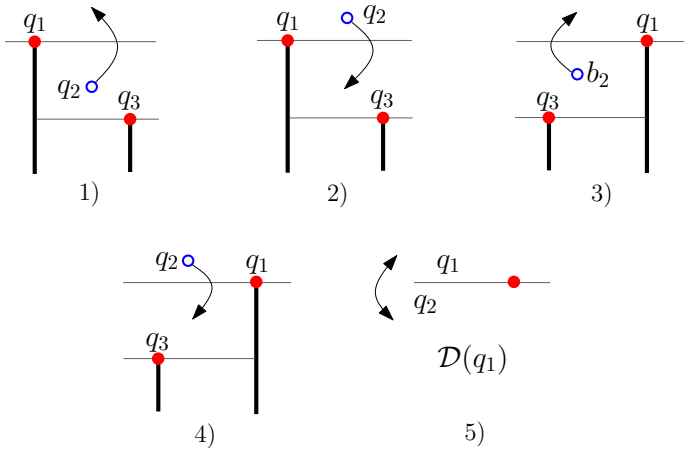


Figure 3.7: y-flip cases between a red point q_1 and a blue point q_2.

Algorithm 3 red-blue-y-flip-1(q_1, q_2)

Remove q_2 from $\mathcal{D}(q_1)$ (see Subsection 3.5.1)
for all red points p such that q_1 and q_2 are in $\mathcal{D}(p)$ **do**
  Add $+1$ to the α label of the leaf $\mathrm{I}_{right}(q_1)$ in $\mathrm{T}_{left}(\mathrm{right}(q_1))$.
  if q_1 is a candidate point **then**
    Add $+1$ to the α label of the leaf q_1 in $\mathcal{Q}$ and update the β labels from q_1 to the root. $H(p)$ is now determined by the β label of the root.
  end if
  Let $T = \mathrm{T}_B(q_3)$ if q_3 exists, otherwise $T = \mathrm{T}_{B_{right}}(q_1)$
  Pass q_2 from T to $\mathrm{T}_B(q_1)$
end for

The red-blue x-flip

A red point q_1 and a blue point q_2 make an x-flip. Consider the two cases as depicted in Figure 3.8, and their symmetric cases. Procedure *red-blue*-x-*flip-1* (Algorithm 4) solves case 1).

The red-red flips

Two red points q_1 and q_2 make a flip. We consider two cases as depicted in Figure 3.9, and their symmetric cases. We use the operators *join* and *split* [35] for x-flips. Procedures *red-red*-y-*flip-1* (Algorithm 5) and *red-red*-x-*flip-2* (Algorithm 6) solve cases 1) and 2), respectively.

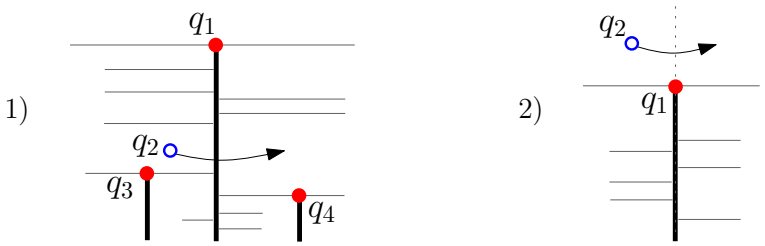


Figure 3.8: Two cases of x-flip between a red point q_1 and a blue point q_2.

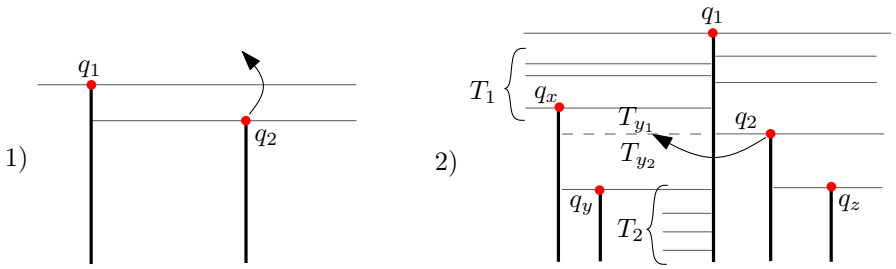


Figure 3.9: A flip of two red points q_1 and q_2. 1) y-flip, 2) x-flip.

In the processing of the flips we first make (if any) some update operations in at most two structures $\mathcal{D}(\cdot)$, and each update costs $O(r+b)$ time in the worst case. After that, we update each $\mathcal{D}(\cdot)$ that contains the flipping points. Each of these updates runs in $O(\log r + \log b)$ time. Thus we have an $O(r \log r + r \log b + b)$-time update for $\mathcal{D}$ when two points make a flip. Finally, the following result is obtained:

Theorem 3.6 *Given a set R of r moving red points and a set B of b moving blue points, whenever two points in $R \cup B$ make a flip, the data structure MBKDS can be updated in $O(r \log r + r \log b + b)$ time.*

3.3.1 A particular case

Suppose that $|R| = 1$, $|B| = n$ and let p be the only one red point. We take p as a fixed point by considering the relative motions of the points in B with respect to p. Under this circumstances, the maximum box corresponds to an isothetic half-plane that contains p on its boundary. In this case, maintaining the maximum box over time is easier. The only thing we have to do is to maintain, between the four possible isothetic half-planes containing p in the boundary, the one that contains the maximum number of blue points inside. We only process the flips between any point of B and p, making an $O(1)$-

Algorithm 4 red-blue-x-flip-1(q_1, q_2)

for all red points p such that q_1 and q_2 are in $\mathcal{D}(p)$ **do**
  Find in $\mathrm{T}_{left}(q_1)$ (resp. $T_{right}(q_1)$) the elements lying below q_2 and by using the α labels modify their $|\mathrm{Blue}(.)|$ value in -1 (resp. $+1$).
  if q_1 is a candidate point **then**
    if $\mathrm{x}(p) < \mathrm{x}(q_1)$ **then**
      Find in $\mathrm{T}_{left}(q_1)$, by using c labels, the elements $\mathrm{I}_{right}(q)$ lying below q_2 where q is a candidate red point, and, by using α labels, modify in -1 their $|\mathrm{Blue}(\mathrm{H}(q))|$ values in $\mathcal{Q}$.
    else
      Find in $\mathrm{T}_{right}(q_1)$, by using c labels, the elements $\mathrm{I}_{left}(q)$ lying below q_2 where q is a candidate red point, and, by using α labels, modify in $+1$ their $|\mathrm{Blue}(\mathrm{H}(q))|$ values in $\mathcal{Q}$.
    end if
    Let $T_2 = \mathrm{T}_B(q_3)$ if q_3 exists, otherwise $T_2 = \mathrm{T}_{B_{right}}(\mathrm{left}(q_1))$
    Let $T_3 = \mathrm{T}_B(q_4)$ if q_4 exists, otherwise $T_3 = \mathrm{T}_{B_{right}}(q_1)$
    Pass q_2 from T_2 to T_3.
  end if
end for

time update of the solution per each. Then, we can design a responsive, compact and local KDS [61].

3.4 The Arbitrarily Oriented Maximum Box Problem

In this section we consider, as an application of our method for the dynamic case, the problem of finding the maximum box for a static set $R \cup B$ on the plane, when the box can be oriented according to any angle (direction) in $[0, \pi]$ (Figure 3.10). This is an open problem [10] and here we give the first non trivial solution.

An orientation of the maximum box can be determined by a rotation of the coordinate axes, and two orientations are equal if and only if their x- and y-orders are respectively equal. Thus, there are $O((r+b)^2)$ critical directions. The method of Section 3.2 can be applied for each critical direction obtaining a simple $O((r+b)^2(r^2 \log r + r^2 \log b + rb \log b))$-time algorithm. However, we can do it better by iterating all the critical directions and computing dynamically the Maximum Box per each. We start with the direction given by the angle $\theta = 0$. Compute the isothetic Maximum Box and the data structure $\mathcal{D}$. Then make a rotational sweep of the coordinate axis maintain-

Algorithm 5 red-red-y-flip-1(q_1, q_2)

Remove q_2 from $\mathcal{D}(q_1)$
Remove q_1 from $\mathcal{D}(q_2)$ (see Subsection 3.5.1)
for all red points p such that q_1 and q_2 are in $\mathcal{D}(p)$ **do**
  if q_1 is candidate and it is not after the flip **then**
    Remove q_1 from $\mathcal{Q}$
  end if
  Find $c_1 = |\,\mathrm{Blue}(\mathrm{H}_{left}(q_1))|$ in $\mathrm{T}_{right}(\mathrm{left}(q_1))$.
  Do the same for $c_2 = |\,\mathrm{Blue}(\mathrm{H}_{right}(q_1))|$ and $c_3 = |\,\mathrm{Blue}(\mathrm{H}_{left}(q_2))|$.
  Remove $\mathrm{I}_{\mathrm{left}}(q_2)$ from $\mathrm{T}_{right}(q_1)$ and insert it in $\mathrm{T}_{right}(\mathrm{left}(q_1))$
  ensuring that $|\,\mathrm{Blue}(\mathrm{H}_{left}(q_2))| = c_1 + c_3$.
  Remove $\mathrm{I}_{right}(q_1)$ from $\mathrm{T}_{left}(\mathrm{right}(q_1))$ and insert it in $\mathrm{T}_{left}(q_2)$
  ensuring that $|\,\mathrm{Blue}(\mathrm{H}_{right}(q_1))| = c_3$.
  if q_1 is candidate after the flip **then**
    Change $|\,\mathrm{Blue}(\mathrm{H}(q_1))|$ in $\mathcal{Q}$ to $c_1 + c_3$.
  end if
  if q_1 was candidate before the flip and not after **then**
    Update c labels in $\mathrm{T}_{left}(\mathrm{right}(q_1))$ and in $\mathrm{T}_{right}(\mathrm{left}(q_1))$.
  end if
  if q_2 appears as candidate after the flip **then**
    Insert q_2 in $\mathcal{Q}$ such that $|\,\mathrm{Blue}(\mathrm{H}(q_2))| = c_1 + c_2$.
    Update c labels in $\mathrm{T}_{left}(\mathrm{right}(q_2))$ and in $\mathrm{T}_{right}(\mathrm{left}(q_2))$.
  end if
  Swap $\mathrm{T}_B(q_1)$ and $\mathrm{T}_B(q_2)$
  Swap q_1 and q_2 in T_R
end for

Algorithm 6 red-red-x-flip-2(q_1, q_2)

Apply GetPriorities and RebuildCandidates in $\mathcal{D}(q_1)$ (see Subsection 3.5.1)
for all red points p such that q_1 and q_2 are in $\mathcal{D}(p)$ **do**
Define q_x, q_y and q_z as depicted in Figure 3.9 b). Note that $\text{left}(q_y) = q_x$, $\text{right}(q_x) = \text{right}(q_y) = q_1$ and $\text{left}(q_z) = q_2$.

Let $T_z = \text{T}_B(q_z)$ if q_z exists, otherwise $T_z = \text{T}_{B_{right}}(q_2)$
Set $T_z = \text{JOIN}(T_z, \text{T}_B(q_2))$

If q_x exists let $v_1 = |\,\text{Blue}(\text{H}_{right}(q_x))|$, otherwise let $v_1 = |\,\text{Blue}(\text{H}_{left}(q_1))|$ and $q_x = \text{left}(q_1)$
Let $T_y = \text{T}_B(q_y)$ if q_y exists, otherwise $T_y = \text{T}_{B_{right}}(q_x)$
Apply SPLIT on T_y by using $\text{y}(q_2)$ to obtain the trees T_{y_1} and T_{y_2} as shown in Figure 3.9 b), and set $T_y = T_{y_2}$ and $\text{T}_B(q_2) = T_{y_1}$.

Apply SPLIT on $\text{T}_{left}(q_1)$ by using $\text{y}(q_2)$ to obtain the trees T_1 and T_2 as shown in Figure 3.9 b).
Set $\text{T}_{left}(q_1) = T_1$. Remove $\text{I}_{right}(q_2)$ from $\text{T}_{left}(\text{right}(q_2))$ and insert it in $\text{T}_{left}(q_1)$ such that $|\,\text{Blue}(\text{H}_{right}(q_2))| = 0$.
Set $\text{T}_{right}(q_1) = \text{JOIN}(\text{T}_{right}(q_1), \text{T}_{right}(q_2))$, $\text{T}_{left}(q_2) = T_2$ and $\text{T}_{right}(q_2) = \text{null}$.
Remove $\text{I}_{left}(q_2)$ from $\text{T}_{right}(q_1)$ and insert it in $\text{T}_{right}(q_x)$ ensuring that $|\,\text{Blue}(\text{H}_{left}(q_2))|$ is equal v_1 plus the number of elements in T_{y_1}

if q_2 is candidate and it is not after the flip **then**
Remove q_2 from $\mathcal{Q}$
Update c labels in $\text{T}_{left}(\text{right}(q_2))$ and in $\text{T}_{right}(\text{left}(q_2))$.
else if q_2 appears as candidate after the flip **then**
Insert q_2 in $\mathcal{Q}$ such that $|\,\text{Blue}(\text{H}(q_2))| = |\,\text{Blue}(\text{H}_{left}(q_2))|$
Update c labels in $\text{T}_{left}(\text{right}(q_2))$ and in $\text{T}_{right}(\text{left}(q_2))$.
end if
Swap q_1 and q_2 in T_R
end for

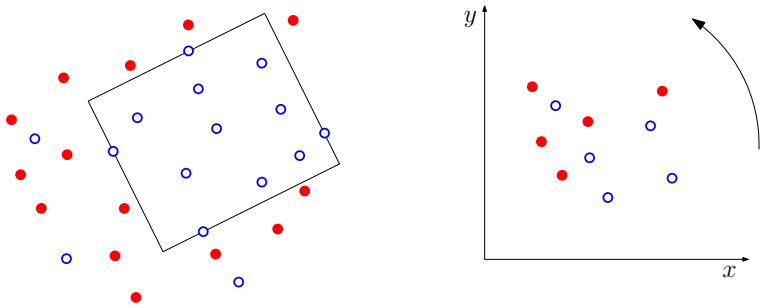


Figure 3.10: The arbitrarily oriented Maximum Box.

ing the x-order and the y-order of the elements of $R \cup B$. The x-order (resp. y-order) changes whenever two points are at the same distance to the y-axis (resp. x-axis), implying the swap (*flip*) of two consecutive elements and the appearance of a new critical direction θ. Thus, the maximum box that is oriented according to θ is computed by making the $O(r \log r + r \log b + b)$-time update in $\mathcal{D}$. The overall rotation angle is at most π radians and the following result is established.

Theorem 3.7 *Given a set of red points R and a set of blue points B on the plane, such that $|R| = r$ and $|B| = b$, the Arbitrarily Oriented Maximum Box Problem can be solved in $O((r+b)^2(r \log r + r \log b + b))$ time and $O(r^2 + rb)$ space.*

3.5 An approximation approach

In this section we study how to maintain an approximation of the maximum box (i.e. the number of blue points is approximated) by using sub-quadratic space. For a constant $c \in (0, 1)$, a *c-approximation of the Maximum Box* is a box containing no red points and at least cb^* blue points, where b^* is the number of blue points in the maximum box. An $O(n \log^2 n)$-time algorithm to compute a 1/2-approximation of the maximum box in the static setting is presented in [83]. The key idea the following: take the median point p of the x-order and compute the exact maximum box H_p that has p on either its left side or its right side. Then, return the best box among H_p and the boxes that are returned by the recursive calls to the sets $\{q \in R \cup B \mid \mathrm{x}(q) < \mathrm{x}(p)\}$ and $\{q \in R \cup B \mid \mathrm{x}(q) > \mathrm{x}(p)\}$.

We use a similar procedure to extend MBKDS to a new KDS named as Apx-MBKDS. First of all, we study in Subsection 3.5.1 the dynamic operations on $\mathcal{D}(p)$, that is, how to efficiently perform insertions and deletions of red

an blue points on it. Next, we consider a new data structure $\mathcal{D}^*(p)$ as an extension of $\mathcal{D}(p)$, and it is used with a binary tree that has the points of R sorted by y-coordinate.

3.5.1 Dynamic operations

In this subsection we show that inserting or removing a blue point in $\mathcal{D}(p)$ can be done in $O(r + \log b)$ time, and that inserting or removing a red point in $O(r + b)$ time. These dynamic operations are based on the following primitives.

GetPriorities: Given a $\mathcal{D}(p)$, computes in $O(r)$ time $|\,\mathrm{Blue}(\mathrm{H}(q))|$ for all the red points q in $\mathcal{D}(p)$. As will stated next. For each red point q in $\mathcal{D}(p)$, apply a top-bottom traversal in $\mathrm{T}_{left}(q)$ and in $\mathrm{T}_{right}(q)$ such that for each node u we take its α label u_α and increase the α labels of its children (if any) in u_α, and after that, reset u_α to zero. When the traversals finish, we have computed $|\,\mathrm{Blue}(\mathrm{H}_{left}(q))|$ and $|\,\mathrm{Blue}(\mathrm{H}_{right}(q))|$ for all q in $\mathcal{D}(p)$. Note that $|\,\mathrm{Blue}(\mathrm{H}(q))| = |\,\mathrm{Blue}(\mathrm{H}_{left}(q))| + |\,\mathrm{Blue}(\mathrm{H}_{right}(q))|$. The overall complexity of the traversals is $O(r)$ time.

IntersectLists(u): Given any $\mathcal{D}(p)$ and a point u, returns two lists L_{left} and L_{rigth} such that: (i) L_{left} (resp. L_{rigth}) contains the red points q of $\mathcal{D}(p)$ such that $\mathrm{I}(q)$ intersects $\mathrm{V}(u)$ and $\mathrm{x}(q) < \mathrm{x}(u)$ (resp. $\mathrm{x}(q) > \mathrm{x}(u)$) (ii) the elements in L_{left} and L_{rigth} are sorted in decreasing order of y-coordinate. This can be done in $O(r)$ time as follows: Set L_{left} and L_{rigth} to two empty lists. Iterate the red points q of $\mathcal{D}(p)$ by traversing the leaves of T_R from left to right. If $\mathrm{I}(q)$ intersects $\mathrm{V}(u)$ then insert q at the end of L_{left} if $\mathrm{x}(q) < \mathrm{x}(u)$, otherwise insert it at the beginning of L_{rigth}.

RebuildCandidates: This is the operation that given $\mathcal{D}(p)$ rebuilds in $O(r)$ time the set $\mathcal{Q}$ associated to $\mathcal{D}(p)$. First, we apply GetPriorities and obtain the candidates points by merging on a list L the two lists returned by the invocation of IntersectLists(p). After that, apply a bottom-top building of $\mathcal{Q}$ from L.

Inserting a blue point u in $\mathcal{D}(p)$ can be done as follows: We find the cell $\mathcal{C}$ of the partition $\mathcal{P}$ determined by $\mathcal{D}(p)$ that contains u, and insert u in $\mathrm{T}_B(\mathcal{C})$. Since we do not store explicitly the cells of $\mathcal{P}$, we have to identify $\mathcal{C}$. Note that either (i) $\mathcal{C}$ has a red point on its bottom boundary, or (ii) $\mathcal{C}$ has no bottom boundary and is bounded by $\mathrm{V}(q)$ and $\mathrm{V}(q')$, where q and q' are consecutive red points in the x-order. In the case (i) we first find left(q) and right(q) for all red points q in $\mathcal{D}(p)$ in overall $O(r)$ time, and after

that, consider, for each red point q in $\mathcal{D}(p)$, the cell $\mathcal{C}_q$ whose bottom side is $\mathrm{I}(q)$ and one vertex is the point of minimum y-coordinate among left(q) and right(q). If u is contained in $\mathcal{C}_q$ then insert u in $\mathrm{T}_B(q)$. In the case (ii) traverse the leaves of T_R from left to right and, if the two consecutive red points q and q' are such that u is contained in the cell of $\mathcal{P}$ having no bottom boundary and bounded by $\mathrm{V}(q)$ and $\mathrm{V}(q')$, then insert u in $\mathrm{T}_{B_{right}}(q)$. After the above operations, we apply GetPriorities and, for all red points q in $\mathcal{D}(p)$, increment $|\,\mathrm{Blue}(\mathrm{H}_{left}(q))|$ (resp. $|\,\mathrm{Blue}(\mathrm{H}_{right}(q))|$) in one if u is above $\mathrm{I}_{left}(q)$ (resp. $\mathrm{I}_{right}(q)$). Finally, we invoke RebuildCandidates. The total time complexity is $O(r + \log b)$.

For further information, we show now how to compute left(q) and right(q) for all red point q in $\mathcal{D}(p)$ in overall $O(r)$ time. Initialize an empty stack $\widehat{s}$ and denote as q_s the current top element of $\widehat{s}$. For each red point q in the x-order (i.e. the leaves of T_R from left to right) do the following. While $\mathrm{y}(q_s) < \mathrm{y}(q)$ set right$(q_s) = q$ and remove q_s from $\widehat{s}$. When the loop ends set left$(q) = q_s$ and push q in $\widehat{s}$. Since every red point is pushed once in $\widehat{s}$, and removed at most once from $\widehat{s}$, the overall time complexity is $O(r)$.

Inserting a red point u requires a different approach, that is described in the following seven steps.

(1) Apply GetPriorities to obtain $|\,\mathrm{Blue}(\mathrm{H}_{left}(q))|$ and $|\,\mathrm{Blue}(\mathrm{H}_{right}(q))|$ for all q in $\mathrm{Red}(S_p)$, and IntersectLists(u) to get the two lists of red points $L_{left} = \{p_1, p_2, \ldots, p_{k_1}\}$ and $L_{right} = \{q_1, q_2, \ldots, q_{k_2}\}$ where the p_i's $(1 \leq i \leq k_1)$ and the q_j's $(1 \leq j \leq k_2)$ are the left and the right red points, respectively, as depicted in Figure 3.11.

(2) Insert u in T_R. After that, compute left(u) and right(u) in $O(\log r)$ time, and compute $|\,\mathrm{Blue}(\mathrm{H}_{left}(u))|$ and $|\,\mathrm{Blue}(\mathrm{H}_{right}(u))|$ in $O(b)$ time. Finally, insert $\mathrm{I}_{left}(u)$ and $\mathrm{I}_{right}(u)$ in $\mathrm{T}_{right}(\mathrm{left}(u))$ and in $\mathrm{T}_{left}(\mathrm{right}(u))$ in $O(\log r)$, respectively.

(3) Suppose now w.l.o.g. that $\mathrm{y}(p_1) > \mathrm{y}(q_1)$. Determine for all red points p' in L_{left} or L_{rigth} $(p' \neq p_1)$ the lists of blue points $L_1(p')$ and $L_2(p')$ whose members are sorted in decreasing order of y-coordinate, and $L_1(p')$ (resp. $L_2(p')$) contains the blue points q' in $\mathrm{T}_B(p)$ such that $\mathrm{x}(q') < \mathrm{x}(u)$ (resp. $\mathrm{x}(q') > \mathrm{x}(u)$). For a given p', $L_1(p')$ and $L_2(p')$ can be obtained in $O(|\,\mathrm{T}_B(p')|)$ time by making an in-order traversal of $\mathrm{T}_B(p')$, where $|\,\mathrm{T}_B(p')|$ denotes the number of blue points stored in $\mathrm{T}_B(p')$. Thus the overall time complexity is $O(r + b)$.

(4) For the new cells $\mathcal{C}_u$, $\mathcal{C}_{p_1}$ and $\mathcal{C}_{q_1}$, whose bottom sides are respectively $\mathrm{I}(u)$, $\mathrm{I}(p_1)$ and $\mathrm{I}(q_1)$, compute in $O(b)$ time the corresponding trees $\mathrm{T}_B(u) = \mathrm{T}_B(\mathcal{C}_u)$, $\mathrm{T}_B(p_1) = \mathrm{T}_B(\mathcal{C}_{p_1})$ and $\mathrm{T}_B(q_1) = \mathrm{T}_B(\mathcal{C}_{q_1})$. Namely, let L_1 be the list of elements of $\mathrm{T}_B(p_1)$ that are above u, L_2 be the list of elements of $\mathrm{T}_B(p_1)$ that are below u and to the left of u, L_3 be the list of elements of $\mathrm{T}_B(p_1)$ that are below u and to the right of u, and L_4 be the list of elements of $\mathrm{T}_B(q_1)$ that are to the right of u. Note that L_1, L_2, L_3, and L_4 can be obtained in $O(b)$ time and sorted in decreasing order of y-coordinate by making in-order traversals in $\mathrm{T}_B(p_1)$ and $\mathrm{T}_B(q_1)$. After that, we build $\mathrm{T}_B(u)$ from L_1, and rebuild $\mathrm{T}_B(p_1)$ from L_2, and $\mathrm{T}_B(q_1)$ from the concatenation of L_3 and L_4.

(5) The elements of L_{left} are the left neighbors of $\mathrm{V}(u)$ and to build $\mathrm{T}_{left}(u)$ we have to compute $|\,\mathrm{Blue}(\mathrm{H}_{right}(p_i))|$ for all p_i in L_{left}. This can be done as follows: First, compute $|\,\mathrm{Blue}(\mathrm{H}_{right}(p_1))|$, and after that, for $i = 2 \ldots k_1$, build $\mathrm{T}_B(p_i)$ in $O(|\,\mathrm{T}_B(p_i)|)$ time from the concatenation of the lists $L_1(q_j)$, where $\mathrm{y}(p_{i-1}) < \mathrm{y}(q_j) < \mathrm{y}(p_i)$, then $|\,\mathrm{Blue}(\mathrm{H}_{right}(p_i))|$ is computed by using the following equation:

$$|\,\mathrm{Blue}(\mathrm{H}_{right}(p_i))| = |\,\mathrm{Blue}(\mathrm{H}_{right}(p_{i-1}))| - |\,\mathrm{Blue}(\mathrm{H}_{left}(p_i))| + |\,\mathrm{T}_B(p_i)|$$

Now proceed similarly to build $\mathrm{T}_{right}(u)$ from L_{right}. The building of $\mathrm{T}_{left}(u)$ and $\mathrm{T}_{right}(u)$ is done in $O(r)$ time.

(6) Compute $T_{B_{right}}(p_{k_1})$ and $T_{B_{right}}(u)$ in $O(b)$ time, similar as we did in Step (4).

(7) Finally, apply RebuildCandidates.

Removing a blue point (resp. a red point) can be done similarly by inserting a blue point (resp. a red point). Note that they are opposite operations. We obtain the following result:

Theorem 3.8 *Inserting or removing a blue point in a given $\mathcal{D}(p)$ can be done in $O(r + \log b)$ time, and inserting or removing a red point in $O(r + b)$ time.*

3.5.2 The approximated KDS

In this section we extend our MBKDS to support the maintenance of a 1/2-approximation of the maximum box. In order to do this, consider the extended version $\mathcal{D}^*(p)$ of $\mathcal{D}(p)$ that takes care of the points that are not only

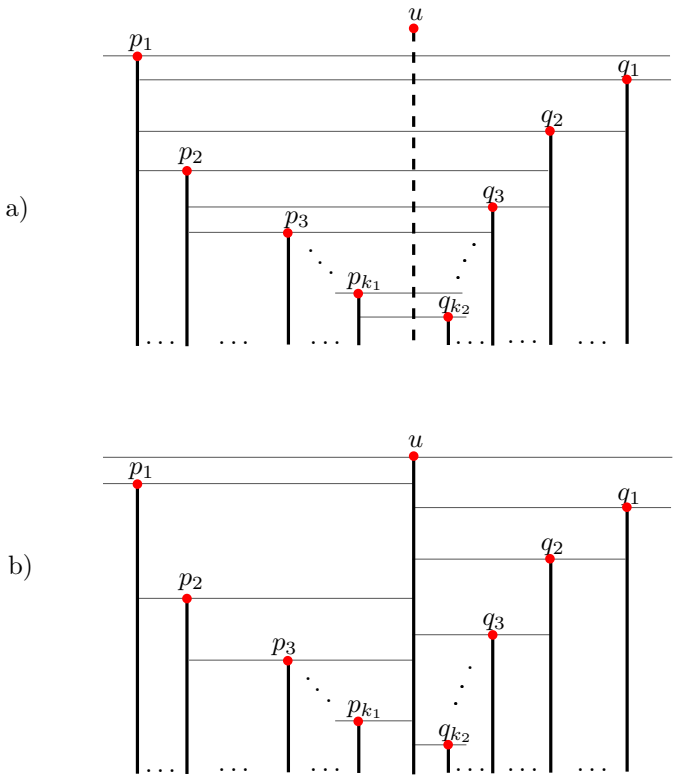


Figure 3.11: Inserting a red point u in $\mathcal{D}(p)$. a) Before insertion, b) after insertion.

below the line h_p but also above it. Those points above h_p are structured symmetrically to the points that lie below (Figure 3.12 a)). Now, suppose the use in the 1/2-approximation of the maximum box algorithm, presented in [83], the median point of the y-order instead of the x-order one's. Our Apx-MBKDS is a two-level data structure that is composed in the first level by a balanced binary search tree $T_{1/2}$ having the elements of R sorted by y-coordinate and in the second level by instances of $\mathcal{D}^*(.)$ (Figure 3.12 b)). The recursive definition of Apx-MBKDS is as follows:

The root u of $T_{1/2}$ is built from $R \cup B$, that is, u stores the median point in the y-order of the red points, and $\mathcal{D}^*(u)$ is built from $R \cup B$. The left and the right children of u are constructed from the sets $\{q \in R \cup B \mid \mathrm{y}(q) < \mathrm{y}(p)\}$ and $\{q \in R \cup B \mid \mathrm{y}(q) > \mathrm{y}(p)\}$, respectively.

Additionally, we associate to every node u of $T_{1/2}$ the best maximum box $H(u)$ of the elements of the subtree rooted at u, that is, the box containing the maximum number of blue points among the box provided by $\mathcal{D}^*(u)$, $H(lc_u)$, and $H(rc_u)$, where lc_u and (rc_u) denote the left and the right chil-

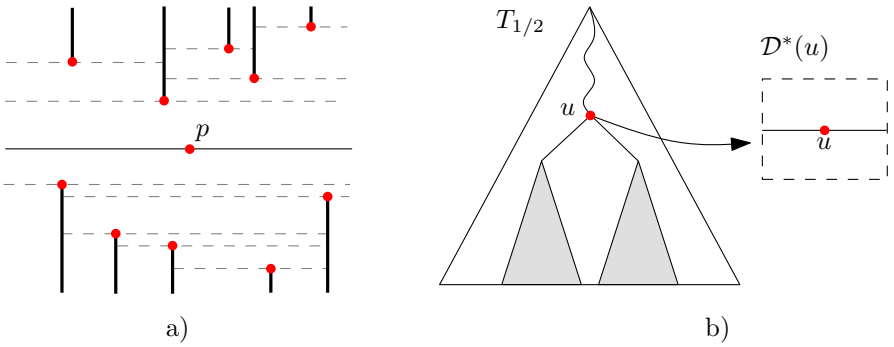


Figure 3.12: a) $\mathcal{D}^*(p)$, b) the two-level data structure Apx-MBKDS for maintaining a 1/2-approximation of the maximum box.

dren of u, respectively. The box $H(root)$ of the root node $root$ is a 1/2-approximation of the maximum box.

Let $u_1, u_2, \ldots, u_k$ be the nodes of a level ℓ of $T_{1/2}$, and denote as r_i and b_i $(1 \leq i \leq k)$ the number of red and blue points in $\mathcal{D}^*(u_i)$, respectively. The total memory used in ℓ is:

$$\begin{aligned} \sum_{i=1}^{k} O(r_i + b_i) &= O\left(\sum_{i=1}^{k} (r_i + b_i)\right) \\ &= O(r + b) \end{aligned}$$

and the overall time to build $\mathcal{D}^*(u_1), \mathcal{D}^*(u_2), \ldots, \mathcal{D}^*(u_k)$ is:

$$\begin{aligned} & \sum_{i=1}^{k} O(r_i \log r_i + r_i \log b_i + b_i \log b_i) \\ = & \sum_{i=1}^{k} O(r_i \log r + r_i \log b + b_i \log b) \\ = & O\left(\left(\sum_{i=1}^{k} r_i\right) \log r + \left(\sum_{i=1}^{k} r_i\right) \log b + \left(\sum_{i=1}^{k} b_i\right) \log b\right) \\ = & O(r \log r + (r + b) \log b) \end{aligned}$$

$T_{1/2}$ has $O(\log r)$ levels because it is a balanced tree. Thus the Apx-MBKDS uses $O((r + b) \log r)$ space and can be built in $O(r \log^2 r + (r + b) \log r \log b)$ time.

Now let us show how to process the flips. Firstly, we depict a common operation that we do at the beginning of the processing of each type of flip (except in the blue-blue x-flip in which we do nothing). Given two flipping

points p_1 and p_2, let u be the node in $T_{1/2}$ where the search paths of p_1 and p_2 split. Then apply the $O(\log r + \log b)$-time update in $\mathcal{D}^*(u')$ for every node u' in the path from the parent of u to the root.

In the blue-blue y-flip, and in the red-blue x-flip, we apply only the above operation.

In the red-blue y-flip, between a red point p_1 and a blue point p_2, we also do the following: Let u be the node in $T_{1/2}$ corresponding to p_1, first update $\mathcal{D}^*(u)$ by applying the insertion/deletion method of a blue point described in Subsection 3.5.1, and after that, update $\mathcal{D}^*(u')$ in every node u' in the path from u to the in-order predecessor (resp. successor) of u in the subtree rooted at u. This procedure spends $O(r + b)$ time.

In the red-red x-flip, between two red points p_1 and p_2, we add the following: Let u_1 and u_2 be the nodes in $T_{1/2}$ that correspond to p_1 and p_2 respectively. If no node among u_1 and u_2 is an ancestor of the other in $T_{1/2}$, then do nothing, otherwise suppose w.l.o.g. that u_1 is an ancestor u_2 and apply the $O(r + b)$-time update in $\mathcal{D}^*(u_1)$.

In the red-red y-flip, between two red points p_1 and p_2, we also do the following: Let u_1 and u_2 be the nodes in $T_{1/2}$ that correspond to p_1 and p_2 respectively, and suppose w.l.o.g. that u_2 belongs to the subtree rooted at u_1, then update $\mathcal{D}^*(u_1)$ by applying the insertion/deletion method of a red point in $\mathcal{D}(.)$ as described in Subsection 3.5.1.

Finally, we note that the complexity of the processing of flips in the worst case is $O(r + b)$ time, and thus we obtain the main result of this section.

Theorem 3.9 *Given a set R of r moving red points and a set B of b moving blue points on the plane, the Apx-MBKDS is a compact and local kinetic data structure that maintains a $1/2$-approximation of the maximum box over $R \cup B$. It uses $O((r + b) \log r)$ space and can be built in $O(r \log^2 r + (r + b) \log r \log b))$ time. Each event can be processed in $O(r + b)$ time in the worst case.*

3.6 Conclusions and open problems

In this chapter we have studied the Maximum Box Problem for moving points on the plane. We have presented a compact and local KDS for maintaining an optimal solution over time. The KDS, named MBKDS, has $O(r^2 + rb)$ space and can be built in $O(r^2 \log r + r^2 \log b + rb \log b)$ time. The changes in the x-order and in y-order of the moving points, that can produce

a change of the maximum box, were named flips, and with the MBKDS the flips can be processed in $O(r \log r + r \log b + b)$ time.

We used the MBKDS to give the first non trivial solution to the problem of finding the maximum box for a static set of bicolored points on the plane, when the box can be oriented according to any direction. This problem was proposed in [10] as an open problem, and we gave here an $O((r+b)^2(r \log r + r \log b + b))$-time and $O(r^2 + rb)$-space algorithm.

Finally, we showed how to extend the MBKDS in order to maintain a 1/2-approximation of the maximum box.

We leave as an open problem the design of a different and more efficient KDS for the maximum box problem for moving points on the plane, as well as an improved algorithm for the arbitrarily oriented maximum box problem.

Chapter 4

The Class Cover Problem with Boxes

In previous chapters we dealt, in static and kinetic scenarios, with the basic problem of finding a box covering the maximum number of blue points and avoiding red points (the MB-problem). Such box does not always cover a reasonable amount of points with respect to the cardinality of the blue class. Consequently, here we consider a related problem which is to find a minimum set of boxes covering all blue points and no red point.

A classical problem in Data Mining and classification problems is the *Class Cover problem* [27, 41, 84]. It consists in, given a bichromatic set of points $S = R \cup B$, finding a minimum-cardinality set of R-empty balls which covers the blue class (i.e. every point in B is contained in at least one of the balls) and with the constraint that balls are centered at blue points. In [27], the authors showed that this problem is NP-hard in general. They also presented an $O(\log n + 1)$-approximation algorithm, and, in the case the points lie in a d-dimensional space with Euclidean norm, for some fixed constant d, they gave a polynomial-time approximation scheme. In this chapter we study a non-constrained version in the plane, named the *Boxes Class Cover problem*, in which axis-aligned rectangles (i.e. boxes) are considered as the covering objects. Our problem can be formulated as follows:

The Boxes Class Cover problem (BCC-problem): *Given $S = R \cup B$, find a minimum-cardinality set $\mathcal{H}$ of R-empty axis-aligned open rectangles such that every point in B is contained in at least one rectangle of $\mathcal{H}$.*

A solution to an instance of the BCC-problem is shown in Figure 4.1 a). A similar problem is the one of covering with disks, not necessarily centered

at blue points, instead of boxes (Figure 4.1 b)). This version of the class cover problem is NP-hard [15], and as we will see in Section 4.3, it admits a constant-factor approximation algorithm [24, 87] and is APX-hard [67].

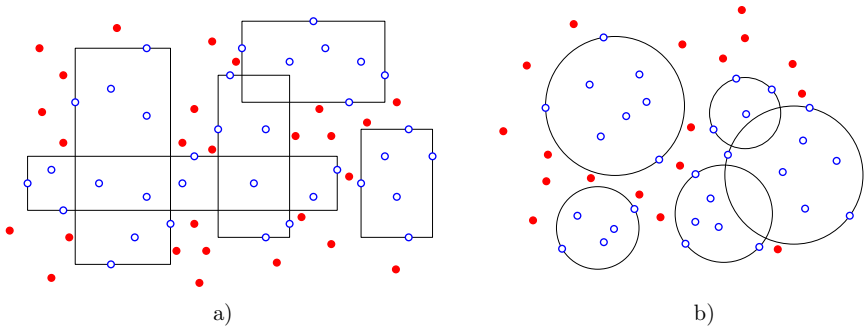


Figure 4.1: Covering the blue class: a) with boxes, b) with disks.

This chapter is outlined in the following way. In Section 4.1 we prove that the BCC-problem is NP-hard. In Section 4.2 we state a first approach to our problem. In Section 4.3 we review related results concerning range spaces and epsilon-nets, which give approximation algorithms to our problem. In Section 4.4 we study the BCC-problem when we restrict the boxes to strips or half-strips. In Section 4.5 we consider the version of the BCC-problem in which the boxes are axis-aligned squares, and we prove its NP-hardness. In Section 4.6 we show that the BCC-problem remains NP-hard if the covering boxes are pairwise disjoint, and show an $O(1)$-approximation algorithm. Finally, in Section 4.7, we state the conclusions and further research.

4.1 Hardness

In this section we prove that the BCC-problem is NP-hard. Our proof is based on a reduction from the *Rectilinear Polygon Covering problem* (RPC-problem), which is defined as follows: Given a rectilinear polygon P, find a minimum cardinality set of axis-aligned rectangles whose union is exactly P. See Figure 4.2 for clarity.

For a general class of rectilinear polygons with holes, Masek [85] proved that the RPC-problem is NP-hard. Culberson and Reckhow [38] used a clever reduction from the 3-SAT-problem [57] to show that this problem is also NP-hard for polygons without holes.

Theorem 4.1 *The BCC-problem is NP-hard.*

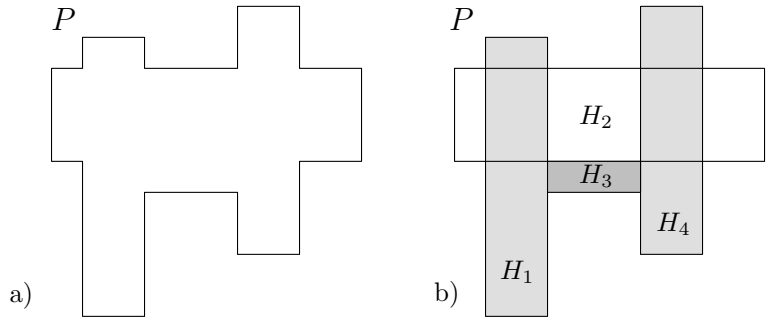


Figure 4.2: a) A rectilinear polygon P. b) An optimal covering of P with four rectangles: H_1, H_2, H_3, and H_4.

Proof. Let the rectilinear polygon P be an instance of the RPC-problem. Let A_1 be the set of all distinct axis-parallel lines that pass through an edge of P. For every two consecutive vertical (resp. horizontal) lines in A_1, draw a vertical (resp. horizontal) line in-between. Denote as A_2 these additional lines. Let G be the grid defined by $A_1 \cup A_2$. We put a red (resp. blue) point in every vertex of $G \setminus P$ (resp. $G \cap P$) (see Figure 4.3).

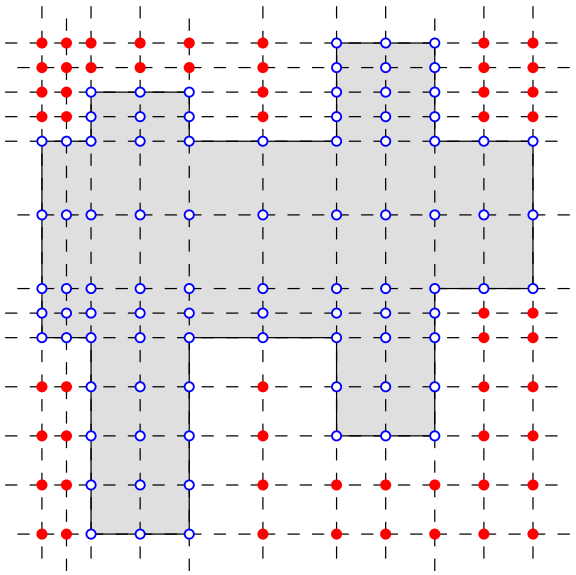


Figure 4.3: The reduction from an instance of the RPC-problem to an instance of the BCC-problem.

Let S be the above set of red and blue points. Clearly, any covering set of P corresponds with a solution to the BCC-problem on S with the same cardinality. Conversely, any solution $\mathcal{H}$ for the BCC-problem on S can be adjusted to be a covering set of P. Namely, let $\mathcal{H} = \{H_1, H_2, \ldots, H_k\}$ be

a solution to the BCC-problem on S. We can consider that each box in $\mathcal{H}$ is maximal, that is, it can not be enlarged in order to contain more blue points. Let H'_i $(1 \leq i \leq k)$ be the bounding box of $H_i \cap B$. We have that every H'_i $(1 \leq i \leq k)$ is fully contained in P. In fact, if some H'_i is not fully inside P, then H'_i must contain at least one cell of $G \setminus P$ which has at least one red vertex u, and thus H_i covers u which is a contradiction. Furthermore, $\bigcup_{i=1}^{k} H'_i$ covers P. This can be checked as follows. Let c be a cell of $G \cap P$. Notice that: (i) c has exactly two adjacent sides in lines of A_1 and two adjacent sides in lines of A_2, and (ii) any maximal box H_v of $\mathcal{H}$, covering the blue vertex v of c whose two edges are in lines of A_1, covers c. Hence, $\mathcal{H}' = \{H'_1, H'_2, \ldots, H'_k\}$ is a solution to the RPC-problem on P. ■

4.2 A simple approach

Any solution $\mathcal{H}$ to the BCC-problem is a covering of B, and we observe that we can extend each box $H \in \mathcal{H}$ until each side of H passes through a red point or reaches infinity. From this observation we will only consider the set $\mathcal{H}^*$ of all the R-empty open boxes whose sides pass through red points or are at infinity. Up to symmetry, such types of boxes are depicted in Figure 4.4.

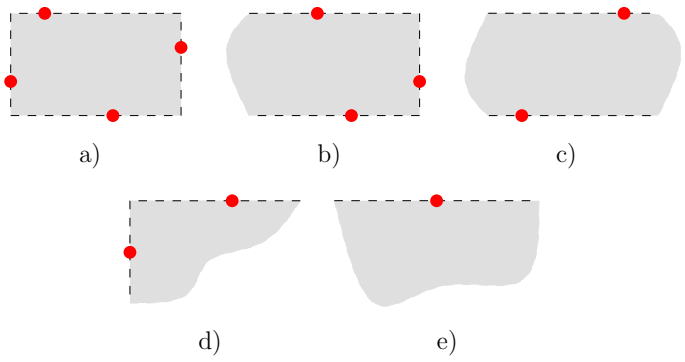


Figure 4.4: Types of boxes in $\mathcal{H}^*$. a) A rectangle, b) a half-strip, c) a strip, d) a quadrant, e) a half-plane.

We notice that the cardinality of $\mathcal{H}^*$ is $O(r^2)$. Namely, let p be a red point and let h_p be the horizontal line through p. For every red point q below h_p, if the box H_{pq}, with opposite vertices p and q, is R-empty, then it can be extended horizontally until its left and right sides touch red points or reach infinity, and then H_{pq} become a member of $\mathcal{H}^*$ (Figure 4.5 a)). We also consider that q is at infinity in which case H_{pq} is a vertical half-line with top-most point p. Then, there are at most $O(r)$ boxes in $\mathcal{H}^*$ whose top side contains p, and thus at most $O(r^2)$ boxes whose top side passes

through a red point. Similarly, there are $O(r^2)$ boxes in $\mathcal{H}^*$ whose left side (resp. right side, bottom side) contains a red point. Since every box in $\mathcal{H}^*$ has a red point on its boundary we conclude that $|\mathcal{H}^*|$ is $O(r^2)$. There are configurations in which $|\mathcal{H}^*|$ is $\Omega(r^2)$. For example, suppose that r is even and distribute two sets of red points S_1 and S_2 with $r/2$ points each (Figure 4.5 b)) and put the b blue points anywhere. Then, we have a box in $\mathcal{H}^*$ for every two consecutive red points in S_1 and two consecutive red points in S_2, thus $r^2/4 = \Omega(r^2)$ in total. We also notice that any solution $\mathcal{H}$ to our problem is such that $|\mathcal{H}| \leq \min\{r+1, b\}$ because either the $r+1$ R-empty vertical strips or b small enough boxes cover B (Figure 4.5 c) and d)).

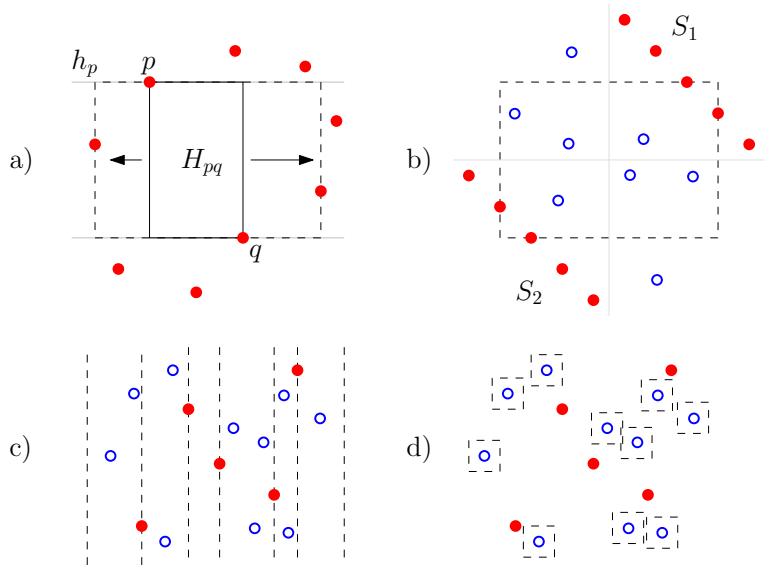


Figure 4.5: a) Finding boxes in $\mathcal{H}^*$ whose top sides contain the red point p. b) An example in which there are $\Omega(r^2)$ boxes in $\mathcal{H}^*$. c) The $r+1$ R-empty vertical strips are enough to cover B. d) A small box for each blue point is a covering of B.

We have seen that the BCC-problem is NP-hard, and to give an exact solution we can apply an exhaustive algorithm. First, compute in $O(r^2)$ time the set $\mathcal{H}^*$, and after that, test for $k = 1 \ldots \min\{r+1, b\}$ if there is a subset of cardinality k in $\mathcal{H}^*$ that covers B. The first covering found is a solution to our problem. Since in the worst case there are $(O(r^2))^{\min\{r+1,b\}}$ subsets of $\mathcal{H}^*$ to be tested, and every test can be done in $O(\min\{r+1, b\} \cdot b) = O(\min\{rb, b^2\})$ time, the overall time complexity is $O(r^{2\min\{r+1,b\}} \cdot \min\{rb, b^2\}) = O(\min\{r^{2r+3}b, r^{2b}b^2\})$. The time complexity of this simple algorithm is exponential in r and polynomial in b, and notice that if either r or b is small, e.g. $O(1)$, then it is polynomial.

4.3 Related results

In this section we relate our problem with ε-nets. A finite[1] range space $(X, \mathcal{R})$ is a pair consisting of an underlying finite set X of objects and a finite collection $\mathcal{R}$ of subsets of X called ranges. Given a range space $(X, \mathcal{R})$, the SET COVER problem [57] asks for the minimum-cardinality subset of $\mathcal{R}$ that covers X. Then, the BCC-problem is an instance of the SET COVER problem in the range space $(B, \mathcal{H}^*)$. The dual of the SET COVER problem is the HITTING SET problem [57], that is, to find a minimum subset $P \subseteq X$ such that P intersects with each range in $\mathcal{R}$. Both problems are NP-hard and the best known factor of approximation of a polynomial-time algorithm is $O(1 + \log |X|)$ [52, 57]. For the SET COVER problem, the algorithm follows the greedy approach [35], that is, while there are elements in X not covered, add to the solution the set of $\mathcal{R}$ covering the maximum number of non-covered elements in X. This approach gives the same logarithmic factor of approximation for the BCC-problem, even if we modify its definition by restricting the covering boxes to axis-aligned squares (Figure 4.6).

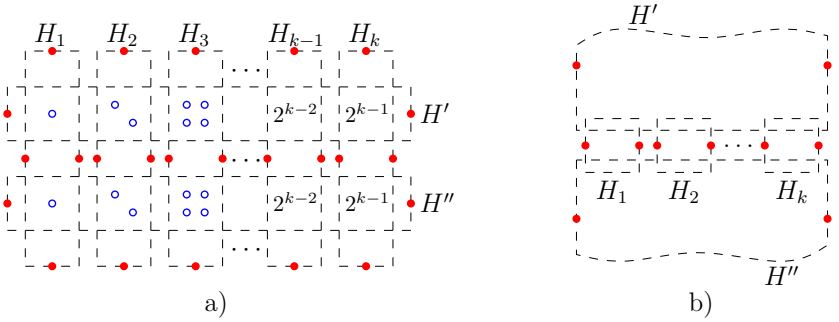


Figure 4.6: The greedy method gives a logarithmic factor of approximation for both boxes and squares. In a) (resp. b)), each of the intersections of the boxes (resp. squares) H' and H'' with the box (resp. square) H_i $(1 \leq i \leq k)$ contains 2^{i-1} blue points. The greedy method reports $\{H_1, H_2, \ldots, H_k\}$ instead of $\{H', H''\}$.

Given the (primal) range space $(X, \mathcal{R})$, its dual range space is $(\mathcal{R}, X^*)$ where $X^* = \{\mathcal{R}_x \mid x \in X\}$ and $\mathcal{R}_x$ is the set of all ranges in $\mathcal{R}$ that contains x [24]. A set cover in the primal range space is a hitting set in its dual, and vice versa. In [24], Brönnimann and Goodrich give a general approach in order to find an approximated hitting set for range spaces, in such a way that when it is solved in the dual range space, it gives a set cover in the primal one. Their method is based on finding small-size subsets called

[1] A range space can be infinite, but for the purpose of our problem we only define it as finite.

ε-nets, as candidate hitting sets, and it works for range spaces with finite VC-dimension [24, 68, 101]. ε-nets were introduced by Haussler and Welzl [68] and became a main concept in computational and combinatorial geometry. Given $0 < \varepsilon < 1$, a subset $N \subseteq X$ is an ε-net for X and $\mathcal{R}$ if any range $H \in \mathcal{R}$ with $|H \cap X| \geq \varepsilon|X|$ contains an element of N. In terms of our problem, an ε-net is a subset $B' \subset B$ such that any box in $\mathcal{H}^*$ that contains $\varepsilon|B|$ points covers an element of B'. In the dual range space, an ε-net is a subset $H \subset \mathcal{H}^*$ that covers all points p of B such that p is covered by at least $\varepsilon|\mathcal{H}^*|$ boxes of $\mathcal{H}^*$. The VC-dimension of $(X, \mathcal{R})$ is stated as the maximum cardinality of a subset $Y \subseteq X$ such that any subset of Y is the intersection of Y with some range in $\mathcal{R}$, and the VC-dimension of the dual range space is less than 2^{d+1}, where d is the VC-dimension of the primal space [24, 101]. Note that the VC-dimension of our range space $(B, \mathcal{H}^*)$ is at most four (i.e. in any subset $P \subseteq B$ with at least five points, there is subset $P' \subset P$ that can not be separated with a box in $\mathcal{H}^*$ from $P \setminus P'$).

For range spaces with constant VC-dimension, the method of Brönnimann and Goodrich [24] reports a hitting set of size at most a factor of $O(\log c)$ from the optimal size c. This result is based on the fact that, for every range space with finite VC-dimension d, there exists an ε-net of size $O(\frac{d}{\varepsilon}\log\frac{d}{\varepsilon})$ [68]. In fact, a random sample of X of this size is an ε-net with high probability [68]. In general, if the range space has constant VC-dimension, and there exists an ε-net of size $O(\frac{1}{\varepsilon}\varphi(\frac{1}{\varepsilon}))$, their method finds a hitting set of size $O(\varphi(c)c)$, where c is the size of an optimal set. Therefore, since our range space $(B, \mathcal{H}^*)$ has constant VC-dimension (also the dual), their technique can be applied to obtain in the dual range space a hitting set of size at most a factor of $O(\log c)$ from the optimal size c, which induces a solution $\mathcal{H}$ (a set cover) for the BCC-problem with the same size. Thus we have an $O(\log c)$-approximation algorithm for the BCC-problem. Recently, Aronov et al. [9] proved the existence of ε-nets of size $O(\frac{1}{\varepsilon}\log\log\frac{1}{\varepsilon})$ for range spaces of points and box ranges. They stated as further research to find, for the dual range space, ε-nets of size less than $O(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$. They also mentioned that in [25] authors claim, without proof, their existence. It would give an $O(\log\log c)$-approximation algorithm to the BCC-problem.

Matoušek et al. proved in [87] that ε-nets of size $O(\frac{1}{\varepsilon})$ exist for range spaces of points and disk ranges, and also for half-spaces in three dimensions. Due to this, techniques in [24] provide an $O(1)$-approximation for the class cover problem with disks. Very recently, Har-Peled [67] proved that the general problem of covering a set of points with the minimum number of disks from a given family of disks is APX-hard [57]. In [34], Clarkson and Varadarajan

showed that if the geometric range space[2] $(X, \mathcal{R})$ has the property that, given a random subset $R' \subset \mathcal{R}$ and a nondecreasing function $f(\cdot)$, there is a decomposition of the complement of the union of the elements of R' into an expected at most $f(|R'|)$ regions (each region of a particular form), then a cover of size $O(f(|C|))$ can be found in polynomial time, where C is the optimal covering set. This result is based on the fact that, with the above conditions, there are ε-nets of size $O(f(\frac{1}{\varepsilon}))$ for the dual range space [34, Theorem 2.2]. If $\mathcal{R}$ is a family of pseudo-disks[3], then the trapezoidization of the complement of any subset $R' \subset \mathcal{R}$ has complexity $O(|R'|)$ and thus the dual range space has ε-nets of size $O(\frac{1}{\varepsilon})$ [34]. Note that a set of axis-aligned squares is a family of pseudo-disks, thus we can give an $O(1)$-approximation algorithm for the BCC-problem (by using the techniques in [24, 34]) if we restrict the covering boxes to axis-aligned squares.

4.4 Solving particular cases

In this section we study the BCC-problem for some special cases. Namely, we consider only certain boxes of $\mathcal{H}^*$ having at most three points on their boundary. In Subsection 4.4.1 we give an $O(r \log r + b \log b + \sqrt{r}b)$-time exact algorithm when we use both horizontal and vertical strips as covering objects. In Subsection 4.4.2 we only consider one type of half-strip, say top-bottom half-strips, and give an $O((r+b) \log \min\{r, b\})$-time algorithm. In Subsection 4.4.3 we prove that the BCC-problem remains NP-hard if we cover with half-strips in the four possible directions, and that there exists a constant factor approximation algorithm based on results in [34]. Finally, in Subsection 4.4.4, we study the case of covering with opposite half-strips, say top-bottom and bottom-top half-strips. In this case we are unable to give either a polynomial-time exact algorithm or a hardness proof, and propose a 2-approximation algorithm.

4.4.1 Covering with horizontal and vertical strips

In this subsection we solve the BCC-problem by using only horizontal and vertical strips. A box in $\mathcal{H}^*$ is a strip if it does not contain red points in two consecutive sides (Figure 4.4 c) and e)). Thus we consider as the covering objects vertical or horizontal strips, and also axis-aligned half-planes.

[2] A range space $(X, \mathcal{R})$ is *geometric* if X is a set of geometric objects, generally points, and $\mathcal{R}$ is a set of geometric ranges such as half-spaces, boxes, convex polygons, balls, etc.

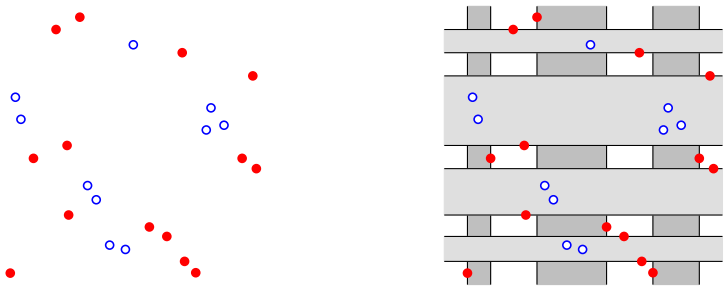
[3] A family of Jordan regions (i.e. regions bounded by closed Jordan curves) is a family of pseudo-disks if the boundaries of any pair of regions intersect at most twice.

Notice that a covering set exists if and only if every blue point can be covered by an axis-parallel line avoiding red points. Also, if a blue point and a red point lie on the same vertical (resp. horizontal) line then the blue point can only be covered by a horizontal (resp. vertical) strip.

Firstly, we sort S by x-coordinate and by y-coordinate (two orders) in $O(r \log r + b \log b)$ time. Preprocess in linear time the x-order to assign for each blue point p the references $r_{\mathrm{x}}^-(p)$ and $r_{\mathrm{x}}^+(p)$ to its previous and next red points, respectively. We do the same with the y-order to assign $r_{\mathrm{y}}^-(p)$ and $r_{\mathrm{y}}^+(p)$. A solution does not exist if and only if there is a blue point p such that $\mathrm{x}(p) = \mathrm{x}(r_{\mathrm{x}}^-(p))$ or $\mathrm{x}(p) = \mathrm{x}(r_{\mathrm{x}}^+(p))$, and $\mathrm{y}(p) = \mathrm{y}(r_{\mathrm{y}}^-(p))$ or $\mathrm{y}(p) = \mathrm{y}(r_{\mathrm{y}}^+(p))$. It can be checked in $O(r + b)$ time. Thus suppose now that a solution exists.

We can assume that there are no red and blue points on the same horizontal or vertical line. Otherwise we can apply the following linear-time preprocessing and after that, solve the same problem for the blue points not yet covered. Given a blue point p, let $H_v(p)$ be the vertical strip bounded by $r_{\mathrm{x}}^-(p)$ and $r_{\mathrm{x}}^+(p)$, and $H_h(p)$ be the horizontal strip bounded by $r_{\mathrm{y}}^-(p)$ and $r_{\mathrm{y}}^+(p)$. From each non-covered blue point p, if $\mathrm{x}(p) = \mathrm{x}(r_{\mathrm{x}}^-(p))$ or $\mathrm{x}(p) = \mathrm{x}(r_{\mathrm{x}}^+(p))$, then include in the solution the strip $H_h(p)$. Otherwise, if $\mathrm{y}(p) = \mathrm{y}(r_{\mathrm{y}}^-(p))$ or $\mathrm{y}(p) = \mathrm{y}(r_{\mathrm{y}}^+(p))$, then include $H_v(p)$.

Consider a graph G whose set of vertices V is the set of strips that cover at least one blue point (Figure 4.7), and whose set of edges E is the following: For each blue point p put an edge between the horizontal strip $H_h(p)$ and the vertical strip $H_v(p)$; different blue points may define the same edge. The graph G is bipartite, has $O(r)$ vertices and $O(b)$ edges, and can be constructed in $O(r + b)$ time.



a) b)

Figure 4.7: a) A set of red and blue points. b) Strips covering at least one blue point

Since each blue point is covered by exactly two strips, the problem is reduced to finding a *Minimum Vertex Cover* [57] in G. However, for bipartite graphs the *Vertex Cover Problem* is equivalent to the *Maximum Matching Problem* because of the König's theorem, and thus it can be solved in $O(\sqrt{|V|}|E|) = O(\sqrt{r}b)$ time [69].

Thus, the following result is obtained:

Theorem 4.2 *The BCC-problem can be solved in $O(r \log r + b \log b + \sqrt{r}b)$ time if we only use axis-aligned strips as covering objects.*

4.4.2 Covering with half-strips in one direction

In this subsection we solve our main problem by considering only half-strips oriented in a given direction, say top-bottom half-strips. A box of $\mathcal{H}^*$ is a half-strip if it contains at most three points on its boundary (Figure 4.4 b), c), d), and e)), and is top-bottom if either contains a red point on its top side or is a vertical strip. We give an optimal $O((r+b)\log \min\{r, b\})$-time algorithm.

Consider the structure of rays that is obtained by drawing a bottom-top red ray starting at each red point as depicted in Figure 4.8. For a given blue point p, let s_p be the maximum-length horizontal segment passing through p whose interior does not intersect any red ray. Let p_l (resp. p_r) be the red point such that the left (resp. right) endpoint of s_p is located in the ray corresponding to p_l (resp. p_r). We say that p_l and p_r are the left and the right red neighbors of p (Figure 4.8). Every time, we select the highest blue point p not yet covered, and include in the solution the top-bottom half-strip H_p whose top side is s_p translated upwards until it touches a red point or reaches the infinite. In other words, H_p is the top-bottom half-strip in $\mathcal{H}^*$ covering p and the maximum number of other blue points. We finish when all blue points are covered.

The correctness of the algorithm follows from the fact that, if p is a blue point not yet covered with maximum ordinate, then H_p is so that, for any other non-covered blue point p' which is not in H_p, p and p' can not be covered with the same top-bottom half-strip. This is because every top-bottom half-strip which covers both p and p', contains at least one of the two red neighbors of p.

In the algorithm, we first preprocess S to obtain the decreasing y-order of the elements of B, and build two balanced binary search trees T_B and T_R

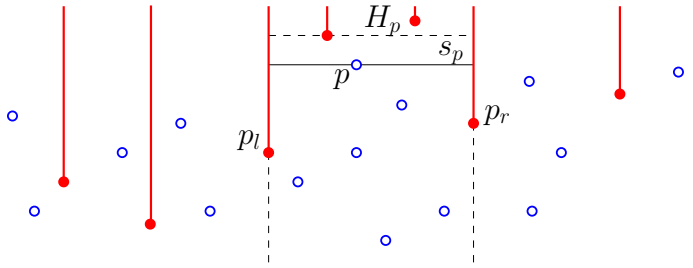


Figure 4.8: The ray structure. The algorithm selects every time the highest blue point p not yet covered and include in the solution the top-bottom half-strip $H_p \in \mathcal{H}^*$ which covers p and the maximum number of other blue points.

containing, respectively, the blue and the red points sorted lexicographically, that is, first by abscissa and after that by ordinate. The first one allows the deletion of elements. In the second one, each node v is labeled with the element of minimum ordinate in the subtree rooted at v. This labeling permits us to obtain the red neighbors for a given blue point p and to determine the top side of H_p, both in $O(\log r)$ time. See Section 3.3 for a similar operation. The preprocessing time is $O(r \log r + b \log b)$ in total. If there are a blue point p and a red point q such that $\mathrm{x}(p) = \mathrm{x}(q)$ and $\mathrm{y}(p) > \mathrm{y}(q)$, then there is no solution to our problem. It can be checked in $O(r + b)$ time by simultaneous in-order traversals on T_R and T_B. Now, we do the following for each blue point p in the decreasing y-order which is not still covered (i.e. p is in T_B): find the left and the right red neighbors p_l and p_r of p, determine H_p and include it in the solution, find in $O(\log b + k_p)$ time the k_p blue points in T_B covered by H_p (i.e. those points p' in T_B such that $\mathrm{x}(p_l) < \mathrm{x}(p') < \mathrm{x}(p_r)$) and remove them from T_B in $O(k_p \log b)$ time. The total time complexity is $O(r \log r + b \log b) + \sum_{p \in B} O(\log b + \log r + k_p \log b) = O(r \log r + b \log b + b \log r) = O(r \log r + b \log b)$.

Suppose now that r is much less than b. Then, before applying the above algorithm, the following can be done in order to reduce the asymptotic time complexity. For all red points p, define the sets of blue points $B_1(p)$ and $B_2(p)$ to group the elements of B, and they are filled up as follows. Sort the red points in lexicographic order to obtain the ordered sequence of red points X_R. Now, for each blue point q, find by the use of a binary search the two consecutive red points in X_R, p' and p, such that $p' < q < p$. If $\mathrm{x}(p') = \mathrm{x}(q)$, then there is no solution. Otherwise, insert q in $S_1(p)$ if $\mathrm{x}(q) < \mathrm{x}(p)$, or in $S_2(p)$ if $\mathrm{x}(q) = \mathrm{x}(p)$. Observe that, for a given red point p, if a top-bottom half-strip in $\mathcal{H}^*$ covers a point in $B_1(p)$ (resp. $B_2(p)$) having maximum ordinate, then it covers all the points of $B_1(p)$ (resp. $B_2(p)$). Therefore, we

can reduce our initial set S to a set S' with fewer points which consists of all the points in R and, for each p in R, of a point of maximum ordinate in $B_1(p)$ and a point of maximum ordinate in $B_2(p)$. The set S' is obtained in $O((r+b)\log r)$ time. Now, we apply the above algorithm to S', and it runs in $O(r \log r)$ time since the number of blue points in S' is at most $2r$. The overall time complexity is $O((r+b)\log r)$. We can proceed analogously in the case that b is much less than r in order to reduce the time complexity to $O((r+b)\log b)$.

In general, we can choose which variant to apply depending on the minority color, and finally obtain an algorithm running in $O(\min\{(r+b)\log r, (r+b)\log b\}) = O((r+b)\log \min\{r,b\})$ time. Thus, the following result is obtained:

Theorem 4.3 *The BCC-problem can be solved in $O((r+b)\log \min\{r,b\})$ time if we only use half-strips in one direction as covering objects.*

Now we show that the above algorithm is optimal in the algebraic computation tree model. Given a set $X = \{x_1, x_2, \ldots, x_n\}$ of n numbers, denote as $x_{\pi_1} \leq x_{\pi_2} \leq \cdots \leq x_{\pi_n}$ the sequence of this number once sorted. The maximum gap of X [8] is defined as MAX-GAP$(X) = \max_{1 \leq i < n}\{x_{\pi_{i+1}} - x_{\pi_i}\}$. It was proven in [8] that, given a set $X = \{x_1, x_2, \ldots, x_n\}$ of n numbers and a positive number ε, the problem of deciding whether or not

$$\text{MAX-GAP}\{x_1, \ldots, x_n, 0, \varepsilon, 2\varepsilon, \ldots, n\varepsilon\} < \varepsilon$$

has an $\Omega(n \log n)$ lower bound in the algebraic computation tree model. By a reduction from this new version of MAX-GAP, we show that our algorithm is optimal.

Theorem 4.4 *The version of the BCC-problem in which we only use to cover half-strips (or strips) in one direction, has an $\Omega(n \log n)$ lower bound in the algebraic computation tree model.*

Proof. Let the set $X = \{x_1, \ldots, x_n\}$ of n numbers and a positive number ε be an instance of the above MAX-GAP problem. Assume that $0 \leq x_i \leq n\varepsilon$, for $i = 1 \ldots n$, because in the contrary case the max gap is greater than or equal to ε. We do the following construction: Put red points in the coordinates $(0,0), (\varepsilon, 0), (2\varepsilon, 0), \ldots, (n\varepsilon, 0)$. Let R be the set of these $n+1$ red points. Put blue points in the coordinates $(x_1, 1), (x_2, 1), \ldots, (x_n, 1)$, and let B be the set of these n blue points. In order to have the max gap

smaller than ε, each of the open intervals $(0,\varepsilon),(\varepsilon,2\varepsilon),\ldots,((n-1)\varepsilon,n\varepsilon)$ has to be pierced by one of the x_i's. Now, solve the BCC-problem for R and B with half-strips in one direction, the top-bottom direction. It follows that

$$\text{MAX-GAP}\{x_1,\ldots,x_n,0,\varepsilon,2\varepsilon,\ldots,n\varepsilon\} < \varepsilon$$

if and only if the minimum number of covering half-strips is exactly n (Figure 4.9). In other words, if the minimum number of covering half-strips is less than n, then

$$\text{MAX-GAP}\{x_1,\ldots,x_n,0,\varepsilon,2\varepsilon,\ldots,n\varepsilon\} \geq \varepsilon$$

■

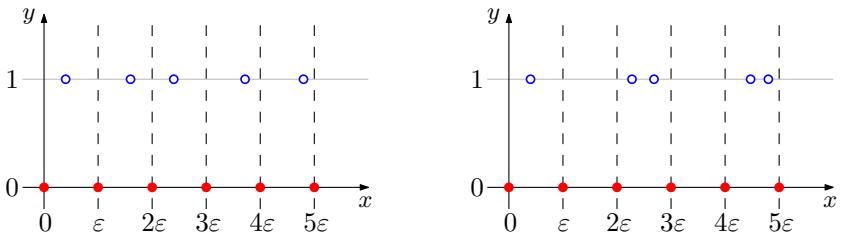


Figure 4.9: The reduction from the MAX-GAP$\{x_1,x_2,x_3,x_4,x_5,0,\varepsilon,2\varepsilon,3\varepsilon,4\varepsilon,5\varepsilon\}$ to our problem. If the solution to our problem consists of five top-bottom half-strips (i.e. strips in $\mathcal{H}^*$) then the MAX-GAP is less than ε (case a)), otherwise the MAX-GAP is exactly ε (case b))

Notice that we can solve the BCC-problem, by only using half-strips in one direction, in $O((r+b)\log\min\{r,b\})$ time, so in fact, if $\min\{r,b\} = \Omega(n)$, then the time complexity is $O(n\log n)$. Therefore, our algorithm is optimal.

4.4.3 Covering with half-strips

In this subsection we study the BCC-problem when the covering boxes are half-strips oriented in any of the four possible directions. First we show that this variant is also NP-hard, and after that we give a constant-factor approximation algorithm due to results in [24, 34]. We name this version as *The Half-Strip Class Cover problem* (HSCC-problem), and use a reduction from the 3-SAT-problem [57] to prove that it is NP-hard.

A solution to the HSCC-problem does not exist if and only if there are two segments with red endpoints, one vertical and one horizontal, such that their intersection is a blue point. This can be checked by using arguments

similar to that given in Subsections 4.4.1 and 4.4.2. We use a reduction from the 3-SAT-problem [57] to prove that the HSCC-problem is NP-hard. An instance of the 3-SAT-problem is a logic formula of t boolean variables $x_1, \ldots, x_t$ given by m conjunctive clauses $C_1, \ldots, C_m$, where each clause contains exactly three literals (i.e. a variable or its negation). The 3-SAT-problem asks for a value assignment to the variables which makes the formula satisfiable, and its NP-hardness is well known [57].

Theorem 4.5 *The HSCC-problem is NP-hard.*

Proof. Given an instance $\mathcal{F}$ of the 3-SAT-problem with t variables $x_1, \ldots, x_t$ and m clauses $C_1, \ldots, C_m$, an instance of the HSCC-problem is constructed in the following manner. Let α be a set of t pairwise disjoint vertical strips of equal width such that the i-th strip from left to right α_i represents the variable x_i. Similarly, let β be a set of $t + m$ pairwise disjoint horizontal strips of equal width. The clause C_j is represented by the $(t + j)$-th strip β_{t+j} from bottom to up. Consecutive strips in α and β are well separated. Let δ_i be the dividing line of α_i. We say that the part of the interior of α_i that is to the right (resp. to the left) of δ_i is the true (resp. false) part of α_i.

For each variable x_i $(1 \leq i \leq t)$ we put in $\alpha_i \cap \beta_i$ a set V_i of red and blue points as follows (Figure 4.10). We add red points in the intersections of δ_i and the boundary of β_i; a blue point p in the center of $\alpha_i \cap \beta_i$ (p is over δ_i); two red points q and q' in the interior of β_i such that q is over the left boundary of α_i and $\mathrm{y}(q) > \mathrm{y}(p)$, and q' is over the right boundary of α_i and $\mathrm{y}(q') < \mathrm{y}(p)$. Moreover, we add two blue points p' and p'' in the interior of $\alpha_i \cap \beta_i$ such that p' is in the false part of α_i and $\mathrm{y}(p') < \mathrm{y}(q')$, and p'' is in the true part of α_i and $\mathrm{y}(p'') > \mathrm{y}(q)$.

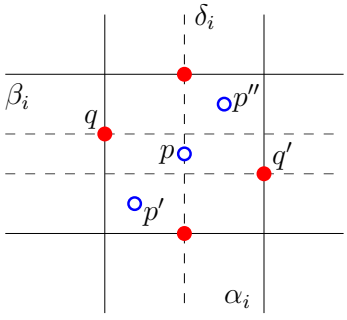


Figure 4.10: Reduction from the 3-SAT-problem. The set of bicolored points V_i for the variable x_i.

For each clause C_j $(1 \leq j \leq m)$ we add a set W_j of bicolored points in the following manner. Suppose that C_j involves the variables x_i, x_k, and x_l $(1 \leq i < k < l \leq t)$. Let ℓ_1 and ℓ'_1 (resp. ℓ_2 and ℓ'_2) be two horizontal lines that are close to the top (resp. bottom) boundary of β_{t+j} such that ℓ_1 (resp. ℓ_2) is outside β_{t+j} and ℓ'_1 (resp. ℓ'_2) is inside (Figure 4.11). Let ℓ_3 and ℓ'_3 be two vertical lines lying outside α_k and such that ℓ_3 and ℓ'_3 are close to the left and right boundaries of α_k, respectively. Put red points at the intersections of the lines ℓ_1 and ℓ_2 with δ_i, ℓ_3, δ_k, ℓ'_3, δ_l, and the boundaries of α_i, α_k, and α_l. Add three more red points, one over the top boundary of β_{t+j}, to the left of ℓ_3 and close to ℓ_3; another between ℓ_3 and the left boundary of β_k, above ℓ'_2 and close to ℓ'_2; and the last one over ℓ'_2 and between the right boundary of β_k and ℓ'_3. Now we add blue points. Put a blue point in the intersection of ℓ'_1 and ℓ_3, and another in the intersection of ℓ'_3 and the bottom boundary of β_{t+j}. If x_i is not negated in C_j, then put in the true part of α_i (otherwise in the false part) two blue points, the first one over ℓ'_1 and the second over the bottom boundary of β_{t+j}. If x_k is not negated in C_j, then put one blue point in the center of the intersection of β_{t+j} and the true part of α_k (otherwise in the false part). Finally, if x_l is not negated in C_j, then put in the true part of α_l (otherwise in the false part) two more blue points, one over the top boundary of β_{t+j} and another over the bottom boundary.

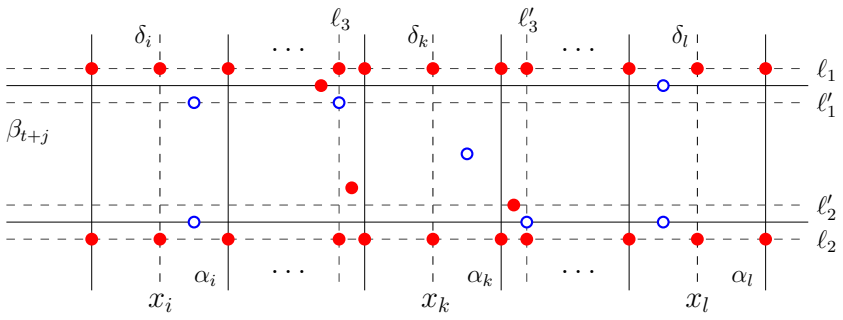


Figure 4.11: Reduction from the 3-SAT-problem. The set W_j of red and blue points for the clause $C_j = (x_i \vee x_k \vee \neg x_l)$.

Let $S = \bigcup_{i=1}^{t} V_i \cup \bigcup_{j=1}^{m} W_j$ be the instance of the HSCC-problem. We say that two blue points in S are independent if they can not be covered with the same half-strip. Notice that for each variable x_i the blue points in V_i are independent from the others blue points in S except with those that are in α_i, and also that at least two half-strips are needed to cover them. Moreover, blue points in the false part of α_i are independent from blue points in the true part. Then, blue points in V_i can be optimally covered in two ways.

The first one with a right-left half-strip covering the two lowest blue points in V_i and a vertical strip covering the true part of α_i (Figure 4.12 a)), and the second one with a vertical strip covering the false part of α_i and a left-right half-strip that covers the upper two blue points of V_i (Figure 4.12 b)). We say that the first way is a true covering of x_i (i.e. x_i is true), and that the second one is a false covering of x_i (i.e. x_i is false).

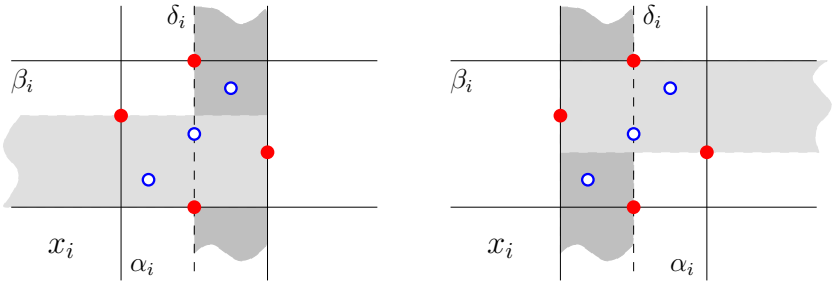


Figure 4.12: The two ways of optimally covering the blue points associated to a variable x_i. a) x_i is equal to true, b) x_i is equal to false.

For each clause C_j $(1 \leq j \leq m)$ that involves the variables x_i, x_k, and x_l $(1 \leq i < k < l \leq m)$ we observe that if at least one variable, say x_i, is such that the covering of V_i covers the blue points in $W_j \cap \alpha_i$ (i.e. the value of x_i makes C_j true), then at least two half-strips are needed to cover $W_j \setminus \alpha_i$. Otherwise, at least three half-strips are needed.

Due to the above observations we claim that $\mathcal{F}$ is satisfiable if and only if the blue points in S can be covered with $2t + 2m$ half-strips. In fact, if $\mathcal{F}$ is satisfiable, then for each variable x_i we cover V_i with a true covering if x_i is true, and otherwise with a false covering. Each clause C_j (with variables x_i, x_k, and x_l) is true, then with two half-strips we can cover the blue points in W_j not covered by the coverings of V_i, V_k, and V_l. We use $2t$ half-strips for the variables and $2m$ for the clauses, thus $2t + 2m$ in total. Inversely, we can not use less than $2t + 2m$ half-strips to cover blue points in S, thus if we use exactly $2t + 2m$, then we have to use two per each variable, and two for each clause, implying that $\mathcal{F}$ is satisfiable if we assign the value true to each variable x_i if V_i has a true covering, and the value false otherwise. Hence, the theorem follows. ■

Given the NP-hardness of the HSCC-problem, we are interested in approximation algorithms. Let $\mathcal{H}_S$ be the set of all half-strips in $\mathcal{H}^*$. By using results in [34] we prove that the dual of the range space $(B, \mathcal{H}_s)$ has ε-nets of size $O(\frac{1}{\varepsilon})$, implying an $O(1)$-approximation algorithm to the HSCC-problem.

Theorem 4.6 *There is a polynomial-time $O(1)$-approximation algorithm for the HSCC-problem.*

Proof. Let $\mathcal{H}_S$ be the set of all half-strips in $\mathcal{H}^*$, and partition $\mathcal{H}_S$ into the subsets $\mathcal{H}_{S_v}$ and $\mathcal{H}_{S_h}$ of all vertical and horizontal half-strips, respectively. Note that $\mathcal{H}_{S_v}$ and $\mathcal{H}_{S_h}$ are families of pseudo-disks. Given $\varepsilon > 0$, the dual of the range space $(B, \mathcal{H}_{S_v})$ has (by using [34]) an $(\frac{\varepsilon}{2})$-net N_v of size $O(\frac{2}{\varepsilon})$. Analogously, the dual of the range space $(B, \mathcal{H}_{S_h})$ has an $(\frac{\varepsilon}{2})$-net N_h of size $O(\frac{2}{\varepsilon})$. We claim that $N_v \cup N_h$ is an ε-net of size $O(\frac{4}{\varepsilon}) = O(\frac{1}{\varepsilon})$ for the dual of $(B, \mathcal{H}_S)$. In fact, if p is a blue point covered by $\varepsilon|\mathcal{H}_S|$ half-strips, then at least $\frac{\varepsilon}{2}|\mathcal{H}_S|$ of them are either vertical or horizontal. Thus p is covered by a half-strip in $N_v \cup N_h$ since N_v and N_h are $(\frac{\varepsilon}{2})$-nets. Hence, there exists, by [24] and also by [34, Theorem 3.2], a polynomial-time $O(1)$-approximation algorithm for the HSCC-problem. ■

4.4.4 Covering with vertical half-strips

In this subsection we study the BCC-problem when the covering boxes are open vertical half-strips. At this moment we are unable to give either a polynomial-time exact algorithm or a hardness proof. Nevertheless, we propose a 2-approximation algorithm. We also observe that the greedy approach gives a logarithmic factor of approximation in the worst case (e.g. consider the set of points in Figure 4.6 b)). There are two types of vertical half-strips, top-bottom and bottom-top. Those that have a red point in its top side are top-bottom, and those with a red point in its bottom side are bottom-top. Vertical strips are simultaneously top-bottom and bottom-top half-strips. The following lemma gives a 2-approximation to our problem.

Lemma 4.7 *Let $\mathcal{H}$ be an optimal solution for the BCC-problem on R and B by using vertical half-strips as covering rectangles. Then $\mathcal{H}$ can be decomposed into sets $\mathcal{H}_1$ and $\mathcal{H}_2$ of open top-bottom and bottom-top half strips, respectively, whose elements are not necessarily in $\mathcal{H}^*$, such that: (i) $\mathcal{H}_1 \cup \mathcal{H}_2$ covers B, (ii) $\mathcal{H}_1 \cup \mathcal{H}_2$ covers no red point, (iii) the elements in $\mathcal{H}_1$ and $\mathcal{H}_2$ have pairwise disjoint interiors, respectively, and (iv) $|\mathcal{H}_1| + |\mathcal{H}_2| < 2|\mathcal{H}|$.*

Proof. Let $H_1, H_2, \ldots, H_m$ be the m top-bottom half-strips plus vertical strips in $\mathcal{H}$. Let $x_1 < x_2 < x_3 < \cdots < x_{2m}$ be the x-coordinates of the sides of the H_i's, $i = 1, \ldots, m$, and define $E = H_1 \cup H_2 \cup \cdots \cup H_m$. Then, $\mathcal{H}_1$ can be defined as $\mathcal{H}_1 = \{([x_j, x_{j+1}] \times (-\infty, +\infty)) \cap E \mid 1 \leq j < 2m\}$. Note that $|\mathcal{H}_1| \leq 2m - 1$. Analogously, $\mathcal{H}_2$ can be defined by considering the m'

bottom-top half-strips in $\mathcal{H}$. Then $|\mathcal{H}_2| \leq 2m' - 1$ and thus $|\mathcal{H}_1| + |\mathcal{H}_2| < 2(m + m') = 2|\mathcal{H}|$. The elements of $\mathcal{H}_1$ and $\mathcal{H}_2$ have pairwise disjoint interiors, respectively; and $\mathcal{H}_1 \cup \mathcal{H}_2$ covers all blue points and no red point since this is done by $\mathcal{H}$. ■

An algorithm that finds optimal sets $\mathcal{H}_1$ and $\mathcal{H}_2$ with the same conditions as in Lemma 4.7 is a 2-approximation algorithm. By using dynamic programming [35], a polynomial time algorithm can be designed. In fact, $\mathcal{H}_1 \cup \mathcal{H}_2$ has optimal substructure because if we take a vertical line ℓ, then ℓ intersects at most two elements of $\mathcal{H}_1 \cup \mathcal{H}_2$, and the half-strips of $\mathcal{H}_1 \cup \mathcal{H}_2$, that are completely to the left (resp. to the right) of ℓ, form an optimal solution for the blue points covered by them.

4.5 Covering with squares

In this section we study the variant of the BCC-problem in which we cover with axis-aligned squares instead of general boxes (rectangles). We name this version *The Square Class Cover problem (QCC-problem)*. In [12] authors studied the problem of covering a binary image with the minimum number of squares, that is: *Given an image, represented by an array of $\sqrt{n} \times \sqrt{n}$ black-and-white pixels, cover the black pixels with a minimum set of (possibly overlapping) squares.* It was proved that obtaining a minimum square covering for a polygonal binary image with holes is NP-hard. By a reduction from it, we show that the QCC-problem is NP-hard. A small variation in our proof shows that the the QCC-problem is also NP-hard if the squares are centered at blue points.

Theorem 4.8 *The QCC-problem is NP-hard.*

Proof. Let P be a polygonal binary image with holes represented by an array of $\sqrt{n} \times \sqrt{n}$ black-and-white pixels. We reduce P to an instance S of the QCC-problem as follows. Let $N = \sqrt{n}$. We can see P as a rectilinear polygon with lattice vertices in $[0, N] \times [0, N]$ (Figure 4.13 a)). We subdivide the square $[0, N] \times [0, N]$ into a regular grid G of cell size $\frac{1}{N}$, and put a blue (resp. red) point in every vertex of G that is in the interior (resp. boundary) of P (Figure 4.13 b)). Let S be the added set of red and blue points.

Any covering of P is a covering of S, and conversely, the squares of a covering in S can be enlarged/shifted to be a covering of P. Namely, let $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_k\}$ be a covering of S. We can assume that Q_i $(1 \leq i \leq k)$ is closed and also maximal, that is, Q_i can not be enlarged without contain-

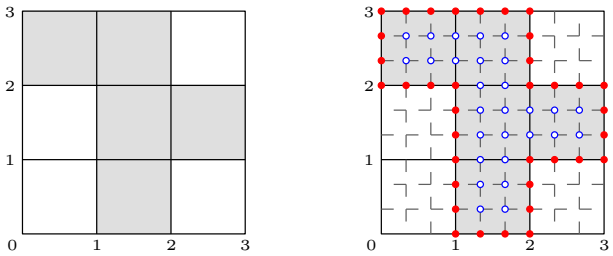


Figure 4.13: The reduction from the problem of covering a binary image with the minimum number of squares to the QCC-problem. a) A binary image represented by an array of 3×3 black-and-white pixels that is seen as a rectilinear polygon with lattice vertices in $[0,3] \times [0,3]$. b) The set of red and blue points generated from the image.

ing red points in its interior; thus the side length of Q_i is an integer number. Now we show that we can shift the elements of $\mathcal{Q}$ in order to obtain a covering of P. Let ℓ be a horizontal line passing through points of S, and $\mathcal{Q}_\ell$ be the set of squares of $\mathcal{Q}$ intersected by ℓ. We have that $\mathcal{Q}_\ell$ covers $P \cap \ell$ but maybe in a set I_ℓ of segments having each length at most $\frac{1}{N}$. For a Q_i in $\mathcal{Q}_\ell$, denote as $\mathrm{x}(Q_i)$ the abscissa of the left side of Q_i. In order to cover all $P \cap \ell$, we shift elements of $\mathcal{Q}_\ell$ as follows. For $j = 0, 1, \ldots, N-1$, if there is no segment in I_ℓ so that the abscissa of its right endpoint is j then: if there is Q_i in $\mathcal{Q}_\ell$ such that $j - 1 < \mathrm{x}(Q_i) < j$, then shift Q_i to the right so that $\mathrm{x}(Q_i)$ is equal to j, otherwise there is a Q_i in $\mathcal{Q}_\ell$ with $j < \mathrm{x}(Q_i) < j + 1$ and we shift Q_i to the left until $\mathrm{x}(Q_i)$ is equal to j. The last condition holds since for the current j the total shift to the left of squares is at most $\frac{j}{N} \leq \frac{N-1}{N} < 1$. By repeating the above process for every horizontal line ℓ passing through points of S, and analogously when ℓ is vertical, the final set $\mathcal{Q}$ covers P. ■

Notice that we can prove that the QCC-problem remains NP-hard if we restrict the squares to be centered at blue points. In fact, we can use the above reduction and only adding blue points over every lattice vertex of the interior of P and at the centers of the pixels of P, and red points over the lattice points of the boundary of P. We have seen in Section 4.3 that there exists an $O(1)$-aproximation algorithm for the QCC-problem since a set of squares is a set of pseudo-disks [34].

4.6 The disjoint version

Another version of the BCC-problem is when the boxes of the solution are pairwise disjoint. This version is also NP-hard as we will prove. We will reduce from the Planar 3-SAT-problem which is strongly NP-complete [81]. In an instance of the Planar 3-SAT-problem we are given a planar bipartite graph such that: the vertices in one part of the bipartition are the variables, the vertices in the another part are the clauses, and the edges connect each clause with the three variables it contains. Moreover, the variables can be arranged as axis-aligned rectangles on a horizontal line, and the three-legged clauses drawn so that all edges are either above or below this line. All the graph can be embedded on a rectangular grid of polynomial size [77] (see Figure 4.14)

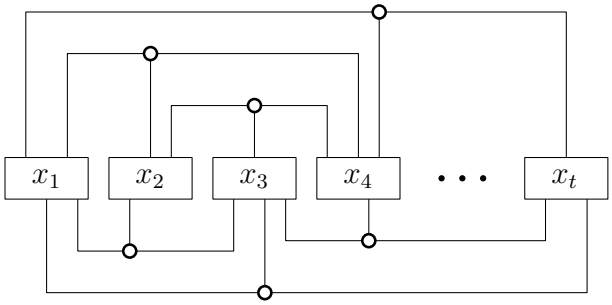


Figure 4.14: Example of a planar 3-SAT instance. The variables are axis-aligned rectangles arranged on a horizontal line, and the clauses are rounded vertices with three orthogonal edges connecting to the variables.

Theorem 4.9 *The version of the BCC-problem in which the covering boxes are pairwise disjoint is also NP-hard.*

Proof. Given an instance $\mathcal{F}$ of the Planar 3-SAT-problem, consisting of t variables $x_1, \ldots, x_t$ and m clauses $C_1, \ldots, C_m$, we construct a bicolored set of point as follows. We dispose the variables and the clauses as in Figure 4.14. We replace each variable x_i $(1 \leq i \leq t)$ by a set V_i of $4K+2$ small axis-aligned boxes arranged in a ring-like fashion, where K is greater than the maximum number of clauses in which a variable appears, as shown in Figure 4.15 a). We say that an axis-aligned box is oriented horizontally if its width is greater than its height, and oriented vertically otherwise. Note that two boxes in V_i, the leftmost and the rightmost, are oriented vertically and the rest $4K$ boxes are oriented horizontally. In each set V_i $(1 \leq i \leq t)$ the boxes are enumerated circularly from 1 to $4K + 2$. Each clause C_j $(1 \leq j \leq m)$, that

involves the variables x_i, x_k, and x_l $(1 \leq i < k < l \leq t)$, is replaced by a set W_j of nine thin boxes as depicted in Figure 4.15 b).

The first (resp. second, third) vertically-oriented box in W_j, from left to right, intersects exactly one horizontally-oriented box H of V_i (resp. V_k, V_l) so that: H is enumerated with an odd number if and only if x_i (resp. x_k, x_l) is not negated in C_j, and V_i (resp. V_k, V_l) contains the center point of H on its interior. Moreover, every box of V_i $(1 \leq i \leq t)$ intersects at most one box of the W_j's $(1 \leq j \leq m)$.

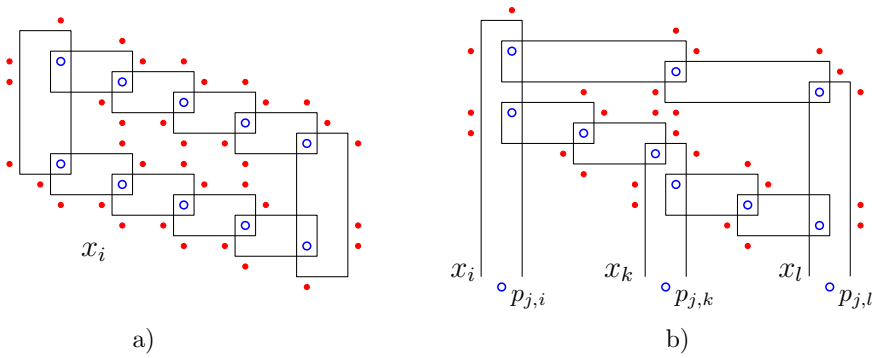


Figure 4.15: Reduction from the Planar 3-SAT-problem. a) The set of boxes V_i corresponding to the variable x_i. b) The set of nine boxes W_j for the clause C_j that involves the variables x_i, x_k, and x_l.

Let $\mathcal{G} = \{V_1, V_2, \ldots, V_t\} \cup \{W_1, W_2, \ldots, W_m\}$. We put a blue point in the intersection of every two boxes of $\mathcal{G}$ ensuring that: if the two boxes are so that one of them is oriented vertically and the other horizontally, the blue point is put closer to either the top-left or the bottom-right corner of the horizontally-oriented box; and if the two boxes are one of some V_i $(1 \leq i \leq t)$, and the other of some W_j $(1 \leq j \leq m)$, put the blue point $p_{j,i}$ in the center of the rectangle of V_i (see Figure 4.15). Let $U_{\mathcal{G}}$ be the region corresponding to the union of all the boxes of $\mathcal{G}$. For each blue point p we add four red points as follows. Let h_p (resp. v_p) be the maximal length horizontal (resp. vertical) segment containing p and fully inside $U_{\mathcal{G}}$. Put two red points, one to the left of p and other to the right, such that they are in the same horizontal line that p, outside $U_{\mathcal{G}}$, and close enough to the endpoints of h_p (see Figure 4.15). Analogously, put two red points, one above p and other below p, such that they are in the same vertical line that p, outside $U_{\mathcal{G}}$, and close enough to the endpoints of v_p. The red points ensure that if two blue points can be covered by a box that does not cover any red point, then both blue points belong to the same box in $\mathcal{G}$.

Let S be the above set of red and blue points. Notice that the set of blue points B_i that are contained in the boxes of a given V_i $(1 \leq i \leq t)$ can be optimally covered by $2K+1$ disjoint boxes which can be boxes of V_i with the same parity in their enumeration number. If the boxes are enumerated with an odd number then we have a true covering of B_i (i.e. x_i is true), otherwise we have a false covering (i.e. x_i is false). For the clause C_j $(1 \leq j \leq m)$, that involves the variables x_i, x_k, and x_l $(1 \leq i < k < l \leq t)$, we have that if at least one of the blue points $p_{j,i}$, $p_{j,k}$, and $p_{j,l}$ is covered by the coverings of B_i, B_k, and B_l, then we need exactly five disjoint boxes to cover the rest of the blue points contained in the boxes of W_j, otherwise we need exactly six. The first case is equivalent to C_j is true, and the second one to C_j is false. From this observations we conclude that $\mathcal{F}$ is satisfiable if and only if the number of disjoint boxes in the solution to the BCC-problem for S is exactly $(2K+1)t + 5m$. Hence, the result follows. ■

In [88] the following problem is studied: Given a set of points P in the plane and a family $\mathcal{F}$ of axis-parallel boxes that covers P, find a smallest set F of pairwise disjoint axis-parallel boxes such that F covers P and every box in F is fully contained within a box of $\mathcal{F}$. This problem is NP-hard and a polynomial-time $O(1)$-approximation algorithm is given in [88]. The algorithm is based on that if we have k disjoint boxes on the plane and construct a binary space partition, it results in $O(k)$ disjoint new boxes such that there exists either a horizontal or a vertical line which separates the set of boxes into two nonempty sets [88, 89]. Due to this, the authors apply dynamic programming. We can apply their approach to obtain a constant-factor approximated solution to our problem. Notice that every box in the solution is fully contained within a box in $\mathcal{H}^*$. Thus we have to solve an instance of the problem in [88] in which P is B and $\mathcal{F}$ is $\mathcal{H}^*$.

4.7 Conclusions and further research

In this chapter we have addressed the class cover problem with boxes, which consists in covering the blue points with a minimum-cardinality set of axis-aligned boxes, and without covering any red point. We proved the NP-hardness by a reduction from the *Rectilinear Polygon Covering Problem* [38], and showed that there is an $O(\log c)$-approximation algorithm due to known results on ε-nets [24, 68, 101], where c is the size of an optimal covering.

Given the hardness of the general problem, we explored some variants by restricting the covering boxes to have special shapes. If the covering boxes are vertical or horizontal strips, we presented an efficient algorithm that runs

in $O(r \log r + b \log b + \sqrt{r}b)$ time. When the covering rectangles are half-strips oriented in one direction, say top-bottom, an $O((r + b) \log \min\{r, b\})$-time algorithm was proposed. If the boxes are half-strips oriented in the four directions, the problem remains NP-hard. We proved the hardness by a reduction from the 3-SAT-problem [57], and showed the existence of an $O(1)$-approximation algorithm from results in [24, 34]. Another variant we considered is when the boxes are vertical half-strips. It was impossible for us to give either a hardness proof or an exact polynomial-time algorithm, but we proved that there exists a 2-approximation algorithm.

The version in which we cover with squares, instead of boxes, is also NP-hard. We proved it by using a reduction from the problem of covering an image with a minimum number of squares [12], which is NP-hard when the image has holes. Due to results in [24, 34], there exists a constant-factor approximation algorithm. Another variant is when the boxes are pairwise disjoint, and we showed its NP-hardness by using the Planar 3-SAT-problem [57, 81]. In that case, an $O(1)$-approximation algorithm exists because of results in [88, 89].

The main results of this chapter are the NP-hardness proofs and the exact algorithms when we cover with strips and top-bottom half-strips, respectively (see Subsections 4.4.1 and 4.4.2). All the approximation algorithms for the NP-hard problems come from results on ε-nets, which were stated for a more general problem, and the factors of approximation given are asymptotic. The major open problem of this chapter is to develop approximation algorithms whose approximation factors are either better than or equal, but not asymptotic, to the ones stated in this chapter. A minor open problem is to give a hardness proof, or an exact polynomial-time algorithm, for the problem of covering with vertical and horizontal half-strips (see Subsection 4.4.4).

Chapter 5

Bichromatic Discrepancy via Convex Partitions

In database management systems, clustering is often an initial stage for data classification [65]. A typical problem in clustering analysis is that of splitting S into disjoint sets $S_1, \ldots, S_k$ such that the elements of each S_i are all red or blue, and their convex hulls are pairwise disjoint. In general we would like each S_i to have a large number of points. However, such partitions do not necessarily exist. For example, let S contain $2n$ elements placed on a circle, and alternating in color. It is easy to see that any partition $S_1, \ldots, S_k$ of S into monochromatic subsets with disjoint convex hulls contains at least $n+1$ elements, n of which contain at most two elements. In this chapter we study a parameter that measures how *blended* two point sets are. Intuitively speaking, R and B are *well blended* if for any convex region C of the plane the proportion of red elements of S is approximately $\frac{r}{b+r}$, e.g. if $r = 2b$, we would expect C to contain twice as many red points as blue.

It is clear that we must be careful in defining well-blended point sets. For example if R and B have n points each, we can always find n disjoint line segments whose endpoints have different colors, and thus we could conclude that $S = R \cup B$ is always well blended. In this chapter we introduce a parameter that seems like a good candidate to measure how well blended a bicolored point set is, we call this parameter the *discrepancy* of S.

If P is a point set, $CH(P)$ will denote the convex hull of P. We say that a point set P is in *convex position* if the elements of P are the vertices of a convex polygon.

Let $S = R \cup B$, and $X \subseteq S$. Let $\nabla(X) = ||X \cap R| - |X \cap B||$. We say

that a partition $\Pi = \{S_1, S_2, \ldots, S_k\}$ of S is a *convex partition* if $CH(S_i) \cap CH(S_j) = \emptyset$ for all $1 \leq i < j \leq k$. The *discrepancy of S with respect to* Π is defined as $d(S, \Pi) = \min_{i=1,\ldots,k} \nabla(S_i)$. The *discrepancy* $d(S)$ of S, is defined as the largest $d(S, \Pi)$ over all the convex partitions Π of S.

If $d(S) = 1$ *any convex partitioning* of S has at least one element S_i such that $\nabla(S_i)$ is one or zero. Such is the case when S contains $2n$ points on a circle such that their colors alternate. On the other hand, if $S = R \cup B$ is separable, i.e. there is a line ℓ that leaves all the elements of R on one of the half-planes it determines, and all the elements of B on the other, the discrepancy of S is at least the minimum of r and b.

If we restrict ourselves to convex partitions of S with exactly k elements, we obtain what we call *the k-discrepancy* of S, denoted as $d_k(S)$. When $k = 1$ then the partition Π has only one element, and thus $d_1(S) = \nabla(S) = |r - b|$. If $k = 2$ then we have what we call *linear discrepancy*, that is, the discrepancy obtained by partitions of S induced by lines that split S into two subsets.

Our concept of discrepancy can be applied in Data Analysis and Clustering for bicolored point sets. We say that $S = R \cup B$ is not good for clustering when its discrepancy is low. The extreme case is when $d(S) = 1$, in this case we say that S is *locally balanced*. Some results of this chapter focus on the hardness of deciding if a bicolored point set is locally balanced or not.

In [2, 86] a parameter known as the *combinatorial discrepancy* of hypergraphs is studied. The problem is that of assigning to each vertex of a hipergraph weight $+1$ or -1 in such a way the maximum weight over all the edges of the hypergraph is minimized, where the weight of a hyperedge is the absolute value of the sum of the weights of its vertices. Another concept of discrepancy is considered in [86], in which the authors study the problem of finding *the most uniform* way of distributing n points in the unit square according to some criteria. In *Geometric discrepancy theory* [30] they study problems such as how to color n points on the plane in such a way that we minimize the difference between the number of red points and blue points within any disk. The papers [14, 42, 44] consider the problem of computing a convex set Q such as a box, triangle, strip, convex polygon, etc. such that the discrepancy of the subset of S contained in Q is maximized. Combinatorial and algorithmic results concerning convex partitions with monochromatic elements can be found in [47]. Other results about subdivisions of the plane into convex regions, containing each specified numbers of red and blue points and inducing a convex partition of the point set, can be seen in [73, 74].

In Section 5.1 we state and prove some properties of the discrepancy of point sets. In Section 5.2 we study the discrepancy of point sets in convex position, and in Section 5.3 of point sets in general position. In Section 5.4 we focus on the *linear discrepancy* of point sets. Finally, in Section 5.5, we give a slightly different definition of discrepancy and state further research.

5.1 Basic properties

Let $X \subset R \cup B$. We denote by $\nabla'(X) = |X \cap R| - |X \cap B|$. Observe that $\nabla(X) = |\nabla'(X)|$. We say that X is *m-red* (resp. *m-blue*) if $\nabla'(X) > 0$ (resp. $\nabla'(X) < 0$). Let $\Pi = \{S_1, S_2, \ldots, S_k\}$ be a convex partition of S. We say that Π is *optimal* if $d(S) = d(S, \Pi)$. Let $r_i = |S_i \cap R|$ and $b_i = |S_i \cap B|$, for $i = 1 \ldots k$.

The following lemmas list some basic properties of the discrepancy of point sets.

Lemma 5.1 *$d(S) \geq 1$. If $d(S) = 1$, then $|r - b| \leq 1$.*

Proof. Suppose that $S = \{p_1, p_2, \ldots, p_{r+b}\}$. Let $\Pi = \{\{p_1\}, \{p_2\}, \ldots, \{p_{r+b}\}\}$. We have $d(S) \geq d(S, \Pi) = 1$. Moreover, if $d(S) = 1$ then $|r - b| = d_1(S) \leq d(S) = 1$. ∎

Lemma 5.2 *If $\Pi = \{S_1, \ldots, S_k\}$ is an optimal convex partition of S and $k \geq 2$, then there are S_i and S_j such that S_i is m-red and S_j is m-blue.*

Proof. Suppose that S_i is m-blue for every index i. Then $b - r = \sum_{i=1}^{k}(b_i - r_i) = \sum_{i=1}^{k} \nabla(S_i) > \min_{i=1\ldots k} \nabla(S_i) = d(S, \Pi) = d(S)$, which contradicts that $d_1(S) \leq d(S)$. Therefore, it follows that Π contains m-blue and m-red elements. ∎

Lemma 5.3 *If a convex partition $\Pi = \{S_1, \ldots, S_k\}$ of S contains an m-red and an m-blue element, then $d(S, \Pi) \leq \min\{r, b\}$.*

Proof. Suppose w.l.o.g. that S_1 is m-red and S_2 is m-blue. Then

$$d(S, \Pi) \leq \nabla(S_1) = r_1 - b_1 \leq r_1 \leq r$$
$$d(S, \Pi) \leq \nabla(S_2) = b_2 - r_2 \leq b_2 \leq b.$$

Hence $d(S, \Pi) \leq \min\{r, b\}$. ∎

Lemma 5.4 *If $r \geq 2b$ or $b \geq 2r$, then $d(S) = d_1(S) = |r - b|$.*

Proof. Assume w.l.o.g. that $r \geq 2b$. We have that $d_1(S) \leq d(S)$. Suppose now that Π is an optimal convex partition of S with cardinality bigger than one. By Lemma 5.2, Π contains m-red and m-blue elements, thus $d(S, \Pi) \leq \min\{r, b\}$ by Lemma 5.3. Then $d(S) = d(S, \Pi) \leq \min\{r, b\} = b \leq r - b = d_1(S)$. This implies that $d(S) = d_1(S)$. ■

Lemma 5.5 *If R and B are linearly separable, and $b \leq r < 2b$ or $r \leq b < 2r$, then $d(S) = \min\{r, b\}$.*

Proof. Suppose w.l.o.g. that $b \leq r < 2b$ and let Π be an optimal convex partition of S. Π can not have cardinality one because $d_1(S) = r - b < b = d(S, \{R, B\})$. Therefore, by Lemma 5.2, Π has m-red and m-blue elements implying, by Lemma 5.3, that $d(S) = d(S, \Pi) \leq \min\{r, b\} = b$. Since $d(S, \{R, B\}) = b$ then $d(S) = b$. ■

It follows easily from Lemmas 5.4 and 5.5 that if R and B are linearly separable, and $2r \leq b$ or $2b \leq r$, then $d(S) = \min\{r, b\}$, otherwise $d(S) = |r - b|$.

The following lemma establishes a relation between the cardinality of the convex partitions and the value of discrepancy.

Lemma 5.6 *Let $\Pi = \{S_1, \ldots, S_k\}$ be a convex partition of S. Then we have that $d(S, \Pi) \leq \frac{r+b}{k}$.*

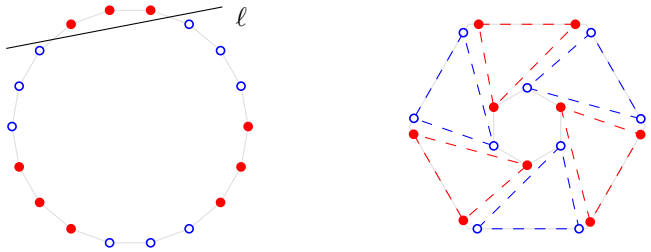
Proof.

$$d(S, \Pi) = \min_{i=1}^{k} \nabla(S_i) \leq \frac{1}{k} \sum_{i=1}^{k} \nabla(S_i) = \frac{1}{k} \sum_{i=1}^{k} |r_i - b_i| \leq \frac{1}{k} \sum_{i=1}^{k} (r_i + b_i) = \frac{r + b}{k}$$

■

It is worth noticing from Lemma 5.6 that if $d(S)$ is large with respect to the cardinality of S, then there exists an optimal convex partition of S with few elements. For point sets with small discrepancy, the minimum cardinality of an optimal convex partition can be small or large (see Figure 5.1).

The following lemma states a relation between the members of a minimum-cardinality optimal convex partition. It shows that in an optimal convex partition with minimum cardinality, one can not join two or more members



a) b)

Figure 5.1: Two point sets with discrepancy 3. In a) the discrepancy is determined by the convex partition induced by the line ℓ. In b) the discrepancy is obtained by using the triangles shown in dashed lines. These examples can be generalized to point sets with $n = tm$ points and discrepancy t.

having the same majority color (i.e. m-red or m-blue) because the convex hull of the resulting member contains points of other members of the partition.

Lemma 5.7 *If Π is a minimum-size optimal convex partition and $S_i, S_j \in \Pi$ are both m-red (or m-blue), then $k \geq 3$. Moreover $CH(S_i \cup S_j) \cap S_l \neq \emptyset$ for every $l \neq i, j$.*

Proof. Suppose that $CH(S_i \cup S_j) \cap S_l = \emptyset$ for every $l = 1 \ldots k$, $l \neq i, j$. Then, the partition $\Pi' = (\Pi \setminus \{S_i, S_j\}) \cup \{S_i \cup S_j\}$ is a convex partition of S so that $d(S, \Pi) \leq d(S, \Pi')$ because $\min\{\nabla(S_i), \nabla(S_j)\} < \nabla(S_i) + \nabla(S_j) = \nabla(S_i \cup S_j)$. This is a contradiction since the cardinality of Π' is $k - 1$. ■

5.2 Point sets in convex position

Let P be a point set in convex position. A subset of P is called *P-consecutive* if it is empty, or its elements are consecutive vertices of $CH(P)$.

Lemma 5.8 *If S is in convex position, then any convex partition $\Pi = \{S_1, \ldots, S_k\}$ of S with $k > 1$, has at least two elements S_i and S_j which are S-consecutive.*

Proof. We assume that $S_i \neq \emptyset$ for every i. The proof is by induction on $r + b$. If $r + b = 1$ then it is trivial. Now suppose that $r + b > 1$. Let $\Pi = \{S_1, S_2, \ldots, S_k\}$ be any convex partition of S, and suppose that the elements of S_1 are not S-consecutive. Then $S \setminus S_1$ is composed by at least

two maximal S-consecutive chains. Let C be one of these chains. If C is not an element of Π then C is partitioned into at least two sets (induced by Π) and the claim follows by induction. Otherwise, each chain in $S \setminus S_1$ is an element of Π and the lemma follows. ■

A point set S in convex position is called an *alternating convex chain* if we can label its elements $p_1, p_2, \ldots, p_{r+b}$ counterclockwise along $CH(S)$ such that for every $1 \leq i < r+b$, p_i and p_{i+1} no have the same color (Figure 5.2).

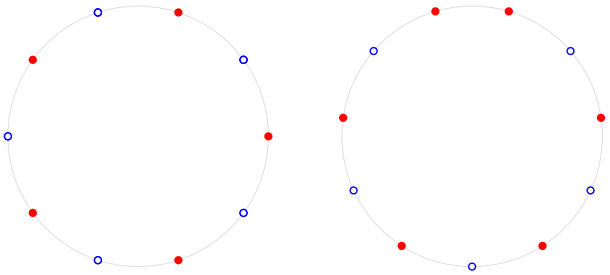


Figure 5.2: Alternating convex chains. a) 5 red points and 5 blue points, b) 6 red points and 5 blue points.

Lemma 5.9 *If S is in convex position then $d(S) = 1$ if and only if S is an alternating convex chain.*

Proof. Suppose that $d(S) = 1$ and that S is not an alternating convex chain. By Lemma 5.1, $|r - b| \leq 1$. If $r = b$ and for some i we have that p_i and p_{i+1} have the same color, then the partition $\Pi = \{\{p_i, p_{i+1}\}, S \setminus \{p_i, p_{i+1}\}\}$ has discrepancy two. If $r = b + 1$ and there is an $1 \leq i < r + b$ such that p_i and p_{i+1} are blue points, then if $S_1 = \{p_i, p_{i+1}\}$, we have that $\nabla(S_1) = 2$, $\nabla(S \setminus S_1) = 3$, and then $d(S) \geq d(S, \Pi) = 2$. Thus S is an alternating convex chain.

Suppose now that S is an alternating convex chain. By Lemma 5.8, any convex partition $\Pi = \{S_1, S_2, \ldots, S_k\}$ of S has at least two S-consecutive elements, and thus for at least one of then, say S_i, $\nabla(S_i)$ is 1 or 0. Then $d(S, \Pi) \leq 1$. ■

Theorem 5.10 *If S is in convex position then, $d(S) = \max_{k=1,2,3} d_k(S)$.*

Proof. Let $d = d(S)$, and observe that $0 \leq \nabla(S) \leq d$. Assume w.l.o.g. that $0 \leq \nabla'(S) \leq d$. Let $\Pi = \{S_1, S_2, \ldots, S_k\}$ be an optimal convex partition of S

of minimum cardinality. By definition, $\nabla(S_i) \geq d$ $(1 \leq i \leq k)$. Suppose that $k > 3$. By Lemma 5.8, S has at least two S-consecutive elements, say S_1 and S_2. If any of S_1 or S_2, say S_1, is such that $\nabla'(S_1) \leq -d$ then $\nabla'(S \setminus S_1) = \nabla'(S) - \nabla'(S_1) \geq 0 + d = d$, and thus $\nabla(S \setminus S_1) = |\nabla'(S \setminus S_1)| \geq d$. This is a contradiction since the convex partition $\Pi' = \{S_1, S \setminus S_1\}$ has cardinality 2 and $d(S, \Pi') \geq d$. Suppose then that $\nabla'(S_1) \geq d$ and $\nabla'(S_2) \geq d$. Observe that $\nabla'((S \setminus S_1) \setminus S_2) = \nabla'(S) - \nabla'(S_1) - \nabla'(S_2) \leq d - d - d = -d$, and thus $\nabla((S \setminus S_1) \setminus S_2) = |\nabla'((S \setminus S_1) \setminus S_2)| \geq d$. This is a contradiction because $\Pi'' = \{S_1, S_2, S \setminus (S_1 \cup S_2)\}$ has cardinality 3 and $d(S, \Pi'') \geq d$. ∎

Our objective now is to prove that the discrepancy of point sets in convex position can be computed in $O(n \log n)$ time. We show first how to solve two problems on circular sequences of real values.

5.2.1 Two maximum weight problems on circular sequences

Consider a set X of n points on a circle denoted clockwise as $x_0, \ldots, x_{n-1}$. An *interval* $[x_i, x_j]$ of X is the subset containing the points $x_i, x_{i+1}, \ldots, x_j$ with addition taken *mod* n. The weight of every point x_i $(0 \leq i < n)$ is the real number $w(x_i)$. The weight $w[x_i, x_j]$ of $[x_i, x_j]$ is defined as $w(x_i) + w(x_{i+1}) + w(\cdots + x_j)$. In this section we solve the following problems:

The Maximum Weight Interval of a Circular Sequence problem (MWI-problem): *Find the interval of X with maximum weight.*

The Max-Min Two Interval problem (MM2I-problem): *Find two disjoint intervals $[x_i, x_j]$ and $[x_k, x_\ell]$ of X such that the minimum of $w[p_i, p_j]$ and $w[p_k, p_\ell]$ is maximized.*

We give an outline of how to solve the MWI-problem, and a more detailed solution to the MM2I-problem.

We notice first that MWI-problem is a small variation on Bentley's well-known maximum weight interval problem. See [17] and the MCS-problem in Section 2.1. More specifically, let $X = (x_0, \ldots, x_{n-1})$ be a *linear* sequence of n elements and weight $w(x_i)$ for $i = 0, \ldots, n-1$. An interval $[x_i, x_j]$ of X contains the elements $x_i, x_{i+1}, \ldots, x_j$, $i \leq j$. Bentley's problem is that of finding the interval of X with maximum weight. Observe that in Bentley's problem, $i \leq j$, whereas in the MWI-problem this is not necessarily the case. It is well known that Bentley's problem can be solved in linear time [17].

Observe that if a solution $[x_i, x_j]$ to the MWI-problem is such that $0 \leq i \leq j \leq n-1$, then the solution obtained by solving Bentley's problem

on $(x_0, \ldots, x_{n-1})$ is $[x_i, x_j]$. Otherwise, $[x_i, x_j]$ is the union of two disjoint intervals $[x_0, x_i]$ and $[x_j, x_{n-1}]$ maximizing $w[x_0, x_i] + w[x_j, x_{n-1}]$. This case can be solved in linear time, see Section 2.1 for more details.

Another linear-time solution to the MWI-problem is the following. In [32], the authors solve in $O(n)$-time the Bentley's problem satisfying length constraints, that is, given a lower bound L, and a upper bound U, find the maximum-weight interval with length at least L and at most U. Observe that if we duplicate our sequence X and find the solution to the length-constrained Bentley's problem with $L = 1$ and $U = n$, then we will obtain a solution to the MWI-problem on X.

We show now how to solve the MM2I-problem in $O(n \log n)$ time. In [32], the authors solve the following problem, which they call the *Range Maximum-Sum Segment Query Problem with Two Query Intervals*:

RMSQ2-problem: *Given a linear sequence $X = (x_0, \ldots, x_{n-1})$, preprocess X in linear time so that, for any $i \leq j \leq k \leq \ell$, the following query can be answered in constant time: Find the interval $[x_s, x_t]$ of maximum weight such that $i \leq s \leq j$ and $k \leq t \leq \ell$.*

Observation 5.11 *If $i = j = k$ in the above problem, $[x_s, x_t]$ will be the interval of maximum weight contained in $[x_i, x_\ell]$ starting at x_i.*

Let $X' = (x_0, \ldots, x_{n-1}, x_n, \ldots, x_{2n-1})$, where $x_{n+i} = x_i$, $i = 0, \ldots, n-1$. Preprocess X' as in [32] to solve the RMSQ2-problem.

Let $I_1 = [x_i, x_j]$ and $I_2 = [x_k, x_\ell]$ form an optimal solution to the MM2I-problem, and suppose w.l.o.g. that $i \leq j < k$. We solve now the MM2I-problem in $O(\log n)$ time for a fixed value of i, $0 \leq i \leq n-1$.

For simplicity assume that $i = 0$. Let $I_1 = [x_0, x_j]$ and $I_2 = [x_k, x_\ell]$ be an optimal solution to our problem. Notice that there exists an index t such that $j \leq t < k$. Let $I_1(t)$ be an interval of maximum weight contained in $[x_0, x_t]$ that starts at x_0. By Observation 5.11, $I_1(t)$ can be found in constant time. Let $I_2(t)$ be the interval of maximum weight contained in $[x_{t+1}, x_{n-1}]$ that can be also found in constant time. Observe that, by the way we choose I_1 and I_2, we can assume that I_1 and I_2 are $I_1(t)$ and $I_2(t)$, respectively. Then, the MM2I-problem consists in finding a value of t so that $\min\{w(I_1(t)), w(I_2(t))\}$ is maximum.

Suppose now that for a given t, $w(I_1(t)) \leq w(I_2(t))$. Then we can discard all indexes $t' < t$ since $\min\{w(I_1(t')), w(I_2(t'))\} \leq w(I_1(t')) \leq w(I_1(t)) = \min\{w(I_1(t)), w(I_2(t))\}$. The case when $w(I_1(t)) > w(I_2(t))$ is analogous

and all indexes $t' > t$ are discarded. It now follows that we can search for t in a logarithmic number of steps. Since the RMSQ2-problem can be used to obtain both $I_1(t)$ and $I_2(t)$ in constant time, t can be found in logarithmic time. We repeat this procedure for $i = 1, \ldots, n-1$ by using the preprocessing done in X' and choosing I_1 and I_2 in the interval $[x_i, x_{i+n-1}]$ of X'.

Thus we have proved:

Theorem 5.12 *The MM2I-problem can be solved in $O(n \log n)$ time.*

5.2.2 Computing the discrepancy of point sets in convex position

Theorem 5.13 *The discrepancy of a point set S in convex position can be computed in $O(n \log n)$ time and $O(n)$ space.*

Proof. Suppose w.l.o.g. that $b \leq r$. By Lemma 5.4, if $r \geq 2b$, then $d(S) = d_1(S) = r - b$. Suppose then that this is not the case. By Theorem 5.10, we have to compute the maximum among $d_1(S)$, $d_2(S)$, and $d_3(S)$. Assign weights to the elements of S as follows: red points are weighted $+1$, and blue points -1. We can now consider S as a *weighted circular sequence*.

Computing $d_2(S)$: By Lemma 5.2, any optimal convex partition contains an m-red element and an m-blue element. Let $\Pi = \{S_1, S_2\}$ be a convex partition of S such that S_1 is m-blue and S_2 is m-red, see Figure 5.3 a). We have that $\nabla(S_2) = r_2 - b_2 = (r - r_1) - (b - b_1) = r - b + b_1 - r_1 = r - b + \nabla(S_1) \geq \nabla(S_1)$, and thus $d(S, \Pi) = \nabla(S_1)$. Then $d(S, \Pi)$ is maximum if and only if $\nabla(S_1)$ is maximum. In this case, S_1 corresponds to an interval of S with minimum weight (i.e. S_2 has maximum weight). This an instance of the MWI-problem and can be solved in linear time.

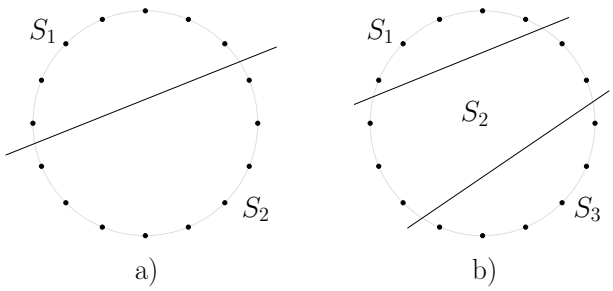


Figure 5.3: Points in convex position. a) Computing $d_2(S)$, b) computing $d_3(S)$.

Computing $d_3(S)$: Assume that $d(S) = d_3(S)$. Let $\Pi = \{S_1, S_2, S_3\}$ be an optimal convex partition of S, see Figure 5.3 b). It is easy to see, from Lemma 5.7, that if S_i is m-blue (resp. m-red) then S_{i+1} is m-red (resp. m-blue), $i = 1, 2$. Moreover $d(S, \Pi)$ is $\nabla(S_1)$ or $\nabla(S_3)$, otherwise $d(S, \Pi) \leq d(S, \{S_1 \cup S_2 \cup S_3\}) = d_1(S)$. If S_1 and S_3 are m-blue, then $d(S, \Pi)$ is at most $d_2(S)$. Then S_1 and S_3 are m-red, and thus computing $d_3(S)$ reduces to the problem of finding in S two disjoint intervals such that the minimum weight of both of them is maximized. By Theorem 5.12, we can solve this problem in $O(n \log n)$ time. ■

5.3 Point sets in general position

The problem of determining the discrepancy of point sets in general position seems to be non-trivial. At this point, we are unable even to characterize point sets with discrepancy one. In this section we study some particular families of point sets.

Proposition 5.14 *For all $n \geq 4$, there are bichromatic point sets of size n, not in convex position, with discrepancy one.*

Proof. Let S be a point set consisting of the vertices of a regular polygon $\mathcal{P}$ with $2n$ vertices together with an extra point p close to the center of $\mathcal{P}$. Color the vertices of $\mathcal{P}$ red or blue in such a way that adjacent vertices receive different color, and color p red (Figure 5.4 a)). Let $\Pi = \{S_1, S_2, \ldots, S_k\}$ be any convex partition of S. If $k = 1$ then $d(S, \Pi) = 1$. Suppose that $k > 1$, then there is some $S_i \in \Pi$ $(1 \leq i \leq k)$ such that $p \notin S_i$ and S_i contains a set of consecutive vertices of $\mathcal{P}$. Then, $\nabla(S_i) \leq 1$ and therefore $d(S, \Pi) \leq 1$.

For $n = 2m + 2$, let S consist of the vertices of $\mathcal{P}$, colored as before plus two points p and q in the interior of $\mathcal{P}$ close enough to the middle of an edge e of $\mathcal{P}$, so that the line joining them is almost parallel to e (Figure 5.4 b)). Observe that $d_2(S) = 1$. Let $\Pi = \{S_1, \ldots, S_k\}$ be any convex partition of S. If p and q are in the same set of Π, then $d(S, \Pi) = 1$ since $S \setminus \{p, q\}$ is an alternating convex chain. Otherwise, $k = 2$ or there is an element $S_i \in \Pi$, not containing both p and q, so that S_i is composed by consecutive vertices of $\mathcal{P}$ and thus $d(S, \Pi) \leq \nabla(S_i) \leq 1$. It follows now that $d(S) = 1$. ■

Proposition 5.15 *Let $b \leq r < 2b$, and d an integer such that $\max\{1, |r - b|\} < d \leq b$. Then there exists a set of S with r red points and b blue points, not in convex position, such that $d(S) = d$.*

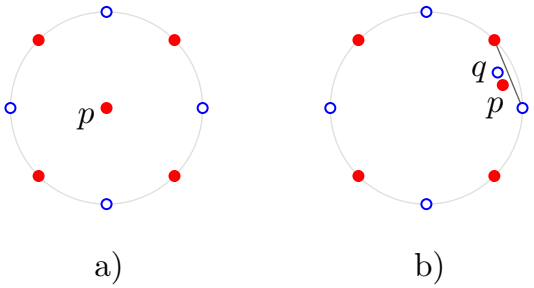


Figure 5.4: In a) (resp. b)), we show a point set with discrepancy one and an odd (resp. even) number of points.

Proof. We construct first a point set S as follows. Let c be a circle centered at the origin O, and let α be an arc of c with length $\frac{\pi}{2}$. Let m be the point such that the midpoint of the segment joining O to m is the midpoint of α. On a small circle centered at O place a point set X_1 of $r - b + d - 2$ red points uniformly spaced. In a similar way, place a point set X_2 of $d-2$ blue points uniformly spaced on a small circle whose center is m. Finally, place a set X_3 of $b-d+2$ pairs of points p_i, q_i close enough to α such that each pair contains a red and a blue point as shown in Figure 5.5, $i = 1, \ldots, b-d+2$. Let $S = X_1 \cup X_2 \cup X_3$.

If a convex set Q is such that $S \cap Q$ is m-blue, then $\nabla(S \cap Q) \leq d$. In fact, suppose that Q contains exactly h blue points of X_3, then Q has at least $h-2$ red points of X_3 and hence $\nabla(S \cap Q) \leq h - (h-2) + (d-2) = d$.

Since $d_1(S) = d > r - b$, it follows that $d(S)$ is not $d_1(S)$. Let $\Pi = \{S_1, \ldots, S_k\}$ a convex partition of S such that $k \geq 2$. By Lemma 5.2, Π has an m-blue element S_i and then $d(S, \Pi) \leq \nabla(S_i) = \nabla(S \cap CH(S_i)) \leq d$.

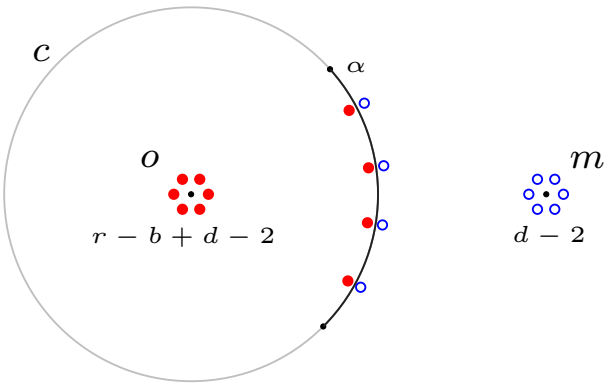


Figure 5.5: A point set in general position with r red points and b blue points. Its discrepancy is equal to d, where $\max\{1, |r-b|\} < d \leq b$.

Choose now two pairs of points p_i, q_i and p_j, q_j in X_3, and let ℓ be the line that passes trough the midpoints of the segments determined by p_i, q_i and p_j, q_j. Let S' and S'' be the subsets of S determined by ℓ. Then $d(S, \{S', S''\}) = \min\{r - b + d, d\} = d$. Hence $d(S) = d$. ∎

Let $C_1, \ldots, C_t$ be a family of sets of points such that each C_i is in convex position. We say that $C_1, \ldots, C_t$ is *nested* if the elements of C_{i+1} belong to the interior of $CH(C_i)$, $i = 1, \ldots, t-1$. The following results deal with families of nested even alternating convex chains, that is, alternating convex chains containing an even number of points, see Figure 5.6 a). This special configuration allows us to give examples with any value of discrepancy and whose points are "well blended".

Lemma 5.16 *Let $C_1, \ldots, C_t$ be a nested family of point sets in convex position, $S = C_1 \cup \cdots \cup C_t$, and $\Pi = \{S_1, \ldots, S_k\}$ a convex partition of S. Then there is an element S_i such that $S_i \cap C_j$ is C_j-consecutive for $j = 1, \ldots, t$.*

Proof. The proof is by induction on t. For $t = 1$ the result follows from Lemma 5.8. Suppose that $t > 1$ and let $\Pi = \{S_1, \ldots, S_k\}$ be any convex partition of S. If $S_i \subset C_1$ for some $i \in \{1, \ldots, k\}$, then the result follows again from Lemma 5.8. Let $\Pi_1 = \{S_1 \setminus C_1, \ldots, S_k \setminus C_1\}$. Π_1 is a convex partition of $S \setminus C_1$, and by induction there is a subset $S_i \in \Pi$ such that $S_i \setminus C_1 \in \Pi_1$ and $C_j \cap (S_i \setminus C_t)$ is C_j-consecutive for $j = 2, \ldots, t$. If $S_i \cap C_1$ is C_1-consecutive our result follows. Otherwise, it is easy to see that there is another element $S_l \in \Pi$ such that $S_l \subset C_1$. Our result follows. ∎

Proposition 5.17 *Let $C_1, \ldots, C_t$ be a family of nested even alternating convex chains, and $S = C_1 \cup \cdots \cup C_t$. Then $d(S) \leq t$. In some cases, $d(S) = t$.*

Proof. Let $\Pi = \{S_1, \ldots, S_k\}$ be a convex partition of S. By Lemma 5.16, there is at least one $S_i \in \Pi$ such that $S_i \cap C_j$ is C_j-consecutive for $j = 1, \ldots, t$. Then $\nabla(S_i \cap C_j) \leq 1$ for $j = 1, \ldots, t$. But $d(S, \Pi) \leq \nabla(S_i) \leq \sum_{j=1}^{t} \nabla(S_i \cap C_j) \leq t$ and $d(S) \leq t$.

Let W_m be the set of $4m^2$ points with integer coordinates (i, j), $1 \leq i, j, \leq 2m$, and such that if $i + j$ is even the point is colored blue, otherwise it is colored red. We call such a point set an m-chessboard. Note that W_t is the union of t nested even alternating convex chains, and thus $d(S) \leq t$. Let ℓ be a line with slope $\frac{\pi}{2}$ that leaves $1 + \cdots + 2t - 1$ elements of W_t below it, see Figure 5.6 b). Then the partition of W_t induced by ℓ has discrepancy

t. It is clear that W_t can be perturbed a bit so that all of its points are in general position without changing our results. ■

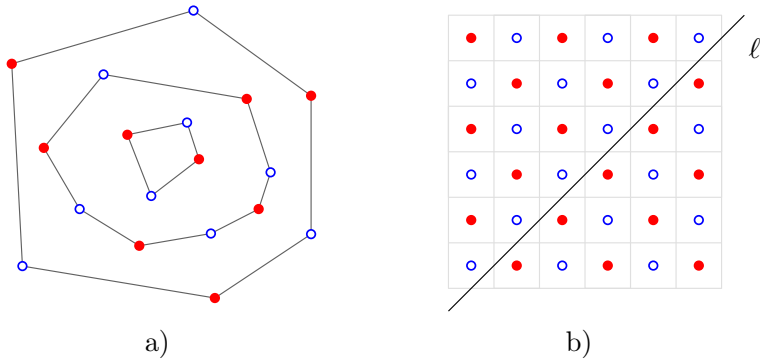


Figure 5.6: a) A configuration of 3 nested even alternating convex chains. b) a 3-chessboard and a line ℓ giving a partition with discrepancy 3.

The idea of the t-chessboard can be generalized as in Figure 5.7, in which there is a line ℓ so that, in each of the half-planes defined by ℓ, all the t chains have the same majority color. It results in interesting cases depending of the value of t. If S is formed by t even alternating convex chains with $4t$ points each, then $d(S) = t = \frac{\sqrt{4t^2}}{2} = \frac{\sqrt{n}}{2}$. If S is a set of $n = 2^{2^m}$ $(m \geq 1)$ points distributed in $t = 2^m = \log_2 n$ even alternating convex chains of length $2^{2^m - m} = \frac{n}{\log_2 n}$ each, then $d(S) = t = \log_2 n$.

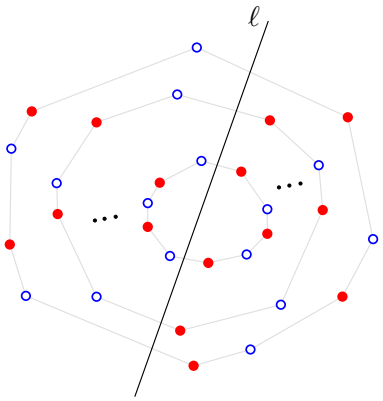


Figure 5.7: A generalization of the m-chessboard.

5.4 Partitions with a line

In this section we characterize sets with linear discrepancy one and show how to decide if the linear discrepancy of a bicolored point set is equal to a

given d. We introduce the following notation.

Let Π_{ℓ^+} and Π_{ℓ^-} be the open half-planes bounded below and above respectively by a non vertical line ℓ. Let $S_{\ell^+} = S \cap \Pi_{\ell^+}$, $S_{\ell^-} = S \cap \Pi_{\ell^-}$, and $\Pi_\ell = \{S_{\ell^+}, S_{\ell^-}\}$. The *linear discrepancy of* S is $d_2(S) = \max_\ell d(S, \Pi_\ell)$, where the lines ℓ contain no point in S.

Proposition 5.18 *Let $S = R \cup B$ such that $r = b$ and $d_2(S) = 1$. Then the following properties hold:*

1. *The convex hull of S is an alternating chain.*
2. *When projected on any line, the points of S form a sequence such that no three consecutive points have the same color.*
3. *For every $p \in S$ on the convex hull of S, the angular ordering of the elements of $S \setminus \{p\}$ with respect to p is a sequence with alternating colors.*
4. *For every line ℓ passing through two points of the same color, say red, the number of red points in each of S_{ℓ^+} and S_{ℓ^-} is exactly one less than the number of blue points in S_{ℓ^+} and S_{ℓ^-}, respectively.*

Property 2 in Proposition 5.18 is not sufficient to guarantee that $d(S) = 1$, e.g. see Figure 5.8 (a). If $r \neq b$ properties 3 and 4 in the same proposition are not necessarily true, see Fig 5.8 b) and c). We now show that if $r = b$, property *4* is sufficient.

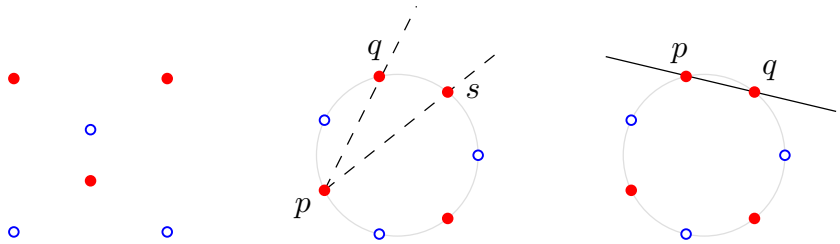


Figure 5.8: a) There are no three consecutive points of the same color in the projection on any line and $d_2(S) = 2$, b) the red points q and s are consecutive in the angular sorting of $S \setminus \{p\}$ with respect to p and $d_2(S) = 1$, c) the number of red and blue points in the half-plane above the line through p and q is zero and $d_2(S) = 1$ because S is an alternating convex chain.

The next result, proven in [39], will be useful:

Theorem 5.19 *Let P and Q be two disjoint convex polygons on the plane. Then there is at least one edge e of P or Q such that the line ℓ_e containing e separates the interior of P from the interior of Q (Figure 5.9).*

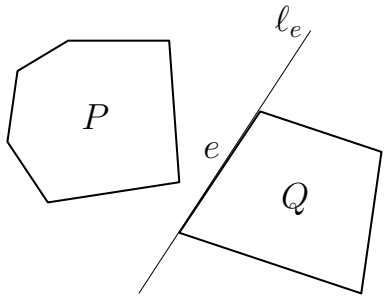


Figure 5.9: Two convex polygons P and Q and a separating line ℓ_e containing the edge e of Q.

Lemma 5.20 *If $r = b$ then the following two conditions are equivalent: (a) $d_2(S) = 1$; (b) for every line ℓ passing through two points of S with the same color, $\nabla(S_{\ell^+}) = \nabla(S_{\ell^-}) = 1$.*

Proof. It is easy to prove that (a) implies (b). We show here that (b) implies (a). Suppose that $d_2(S) = d \geq 2$. We now show that there exists a line ℓ containing two points of the same color of S such that $\{\nabla(S_{\ell^-}), \nabla(S_{\ell^+})\} = \{d, d-2\}$.

Let ℓ_0 be a line containing no elements of S such that $d_2(S) = d(S, \Pi_{\ell_0}) = d$. Assume w.l.o.g. that ℓ_0 is horizontal. Since $r = b$ we have that $d_2(S) = d(S, \Pi_{\ell_0}) = \nabla(S_{\ell_0^+}) = \nabla(S_{\ell_0^-}) = d$ and $\nabla'(S_{\ell_0^+}) = -\nabla'(S_{\ell_0^-})$. Assume w.l.o.g. that $\nabla'(S_{\ell_0^+}) > 0$ (i.e. $S_{\ell_0^+}$ is m-red and $S_{\ell_0^-}$ is m-blue).

Let P and Q be the polygons induced by the convex hulls of $S_{\ell_0^+}$ and $S_{\ell_0^-}$ respectively. Let p be a vertex of P such that there is a line ℓ' passing trough p that separates P from Q. Then p must be a red point, for otherwise by translating ℓ' up by a small distance, we obtain a partitioning Π' of S with discrepancy $d+1$. Similarly any point q in Q such that there is a line trough q that separates P from Q must be blue.

By Theorem 5.19, there is an edge e of P or Q, with vertices p and q such that the line ℓ_e containing e separates P from Q. If e is an edge of P, then it can be shown by using the above observation that p and q are red. Thus $\{\nabla(S_{\ell_e^+}), \nabla(S_{\ell_e^-})\} = \{d, d-2\}$. A symmetric argument works when e belongs to Q. ■

The next proposition gives lower and upper bounds of the linear discrepancy. The proof is in the appendix.

Proposition 5.21 $\max\left\{1, \lfloor\frac{|r-b|}{2}\rfloor\right\} \leq d_2(S) \leq \max\left\{\lfloor\frac{|r-b|}{2}\rfloor, \min\{r,b\}\right\}$. *Furthermore, both bounds are tight.*

Proof. Suppose w.l.o.g. that $r \geq b$. By the *Ham Sandwich Cut Theorem* [58] there exists a line ℓ, passing through at most one red point and at most one blue point, such that $|S_{\ell^+} \cap R| = |S_{\ell^-} \cap R| = \lfloor\frac{r}{2}\rfloor$ and $|S_{\ell^+} \cap B| = |S_{\ell^-} \cap B| = \lfloor\frac{b}{2}\rfloor$. Four cases arise depending on the parities of r and b.

1. If $r = 2a$ and $b = 2c$ then ℓ contains no point of S, and $\nabla(S_{\ell^+}) = \nabla(S_{\ell^-}) = a - c = \lfloor\frac{r-b}{2}\rfloor$. Thus $\lfloor\frac{r-b}{2}\rfloor = d(S, \Pi_\ell) \leq d_2(S)$.

2. If $r = 2a+1$ and $b = 2c+1$ then ℓ passes through one red p point and one blue point q, and $|\nabla(S_{\ell^+})| = |\nabla(S_{\ell^-})| = b-c = \lfloor\frac{r-b}{2}\rfloor$. By moving slightly ℓ, both p and q pass to be either in S_{ℓ^+} or in S_{ℓ^-} thus $|\nabla(S_{\ell^+})|$ and $|\nabla(S_{\ell^-})|$ do not change. Hence $\lfloor\frac{r-b}{2}\rfloor = d(S, \Pi_\ell) \leq d_2(S)$.

3. If $r = 2a + 1$ and $b = 2c$ then ℓ passes through only one red point p and no blue point. $|\nabla(S_{\ell^+})| = |\nabla(S_{\ell^-})| = a - c = \frac{r-b-1}{2} = \lfloor\frac{n-m}{2}\rfloor$. By moving slightly ℓ, the point p passes to be either in S_{ℓ^+} or in S_{ℓ^-}. Then $\min\{|\nabla(S_{\ell^+})|, |\nabla(S_{\ell^-})|\}$ does not change and thus $\lfloor\frac{r-b}{2}\rfloor = d(S, \Pi_\ell) \leq d_2(S)$.

4. If $r = 2a$ and $b = 2c + 1$ then ℓ passes through only one blue point q and no red point. $|\nabla(S_{\ell^+})| = |\nabla(S_{\ell^-})| = a - b = \frac{r-b+1}{2}$. By moving slightly ℓ, the point q passes to be either in S_{ℓ^+} or in S_{ℓ^-} thus $\min\{|\nabla(S_{\ell^+})|, |\nabla(S_{\ell^-})|\} = \frac{r-b+1}{2} - 1 = \frac{r-b-1}{2} = \lfloor\frac{r-b}{2}\rfloor$. Hence $\lfloor\frac{r-b}{2}\rfloor = d(S, \Pi_\ell) \leq d_2(S)$.

If $|r - b| \geq 2$ then $d_2(S) \geq \lfloor\frac{|r-b|}{2}\rfloor \geq 1$ thus it is missing to prove that $d_2(S) \geq 1$ when $|r - b| \leq 1$. Suppose w.l.o.g. that $b \leq r \leq b + 1$. If there is a blue point p in the convex hull of S take a line ℓ separating p from $S \setminus \{p\}$ and suppose that $p \in S_{\ell^+}$, then $\nabla(S_{\ell^+}) = 1$, $\nabla(S_{\ell^-}) = r - b + 1 \geq 1$ and $d_2(S) \geq d(S, \Pi_l) = 1$. If no such p exists, there are two consecutive red points p and q in the convex hull of S, then take a line ℓ separating p and q from $S \setminus \{p, q\}$ and suppose that $p, q \in S_{\ell^+}$, then $\nabla(S_{\ell^+}) = 2$, $\nabla(S_{\ell^-}) = b - r + 2 \geq 1$ and $d_2(S) \geq d(S, \Pi_l) = 1$. This proves the lower bound.

We show now that this lower bound is tight. Suppose w.l.o.g. that $r > b$ and let X be a set composed by r red points and r blue points, and let Y be a set of $r-b$ red points. Put the elements of X on an alternating convex chain and the elements of Y in the interior of the convex hull of X in such a way there is a line ℓ_e such that $d(X, \Pi_{\ell_e}) = 0$ and ℓ_e splits Y into two subsets of cardinality $\lfloor \frac{|Y|}{2} \rfloor$ and $\lceil \frac{|Y|}{2} \rceil$ respectively. Let $S = X \cup Y$ and observe that $d(S, \Pi_{\ell_e}) = \lfloor \frac{|Y|}{2} \rfloor = \lfloor \frac{r-b}{2} \rfloor$. For any line ℓ we have that $d(X, \Pi_\ell) \in \{0, 1\}$ (by Lemma 5.9). If $d(X, \Pi_\ell) = 0$ then $d(S, \Pi_\ell) = d(X \cup Y, \Pi_\ell) = d(Y, \Pi_\ell) \leq \lfloor \frac{|Y|}{2} \rfloor = \lfloor \frac{r-b}{2} \rfloor$. If $d(X, \Pi_\ell) = 1$ then $d(X \cup Y, \Pi_\ell) = \min\{x-1, (r-b)-x+1\}$ where x is such that l splits Y into x and $(r-b)-x$ points, respectively. It is easy to prove that $\min\{x-1, (r-b)-x+1\} \leq \lfloor \frac{r-b}{2} \rfloor$. Then $d_2(S) = d(S, \Pi_{\ell_e}) = \lfloor \frac{r-b}{2} \rfloor$.

To prove the upper bound suppose w.l.o.g. that $r \geq b$ (i.e. $b = \min\{r, b\}$). We have to show that $d(S, \Pi_\ell) > b \Rightarrow d(S, \Pi_\ell) \leq \lfloor \frac{r-b}{2} \rfloor$ for every line ℓ. Let ℓ be a line such that $d(S, \Pi_\ell) > b$. Then we have that $\nabla'(S_{\ell^+}) > 0$ and $\nabla'(S_{\ell^-}) > 0$. In fact, suppose that $\nabla'(S_{\ell^+}) < 0$, then $\nabla(S_{\ell^+}) = |S_{\ell^+} \cap B| - |S_{\ell^+} \cap R| \leq |S_{\ell^+} \cap B| \leq b$ thus $d(S, \Pi_\ell) \leq b$, a contradiction. Now, $\nabla'(S_{\ell^+}) > 0$ and $\nabla'(S_{\ell^-}) > 0$ imply that $d(S, \Pi_\ell) \leq \lfloor \frac{r-b}{2} \rfloor$. In fact, suppose the contrary, $\nabla(S_{\ell^+}) \geq \lfloor \frac{r-b}{2} \rfloor + 1$ and $\nabla(S_{\ell^-}) = (r-b) - \nabla(S_{\ell^+}) \geq \lfloor \frac{r-b}{2} \rfloor + 1$, thus $r - b \geq 2\lfloor \frac{r-b}{2} \rfloor + 2$, a contradiction. If $\lfloor \frac{r-b}{2} \rfloor \leq b$ the upper bound is tight if we take separable sets R and B. If $\lfloor \frac{r-b}{2} \rfloor > b$ we have shown above how to build a set of points S with $d_2(S) = \lfloor \frac{r-b}{2} \rfloor$. ∎

Corollary 5.22 *Let $|r-b| \geq 2$. If $r \geq 3b$ or $b \geq 3r$ then $d_2(S) = \lfloor \frac{|r-b|}{2} \rfloor$.*

Proof. Suppose that $r \geq 3b$, then $r - b \geq 2b \Rightarrow \frac{r-b}{2} \geq b \Rightarrow \lfloor \frac{r-b}{2} \rfloor \geq b$. Thus the upper and lower bounds of $d_2(S)$ in Proposition 5.21 are equal. ∎

5.4.1 Hardness

We start with a technical lemma that is proven in the appendix of this chapter, Section 5.6.

Lemma 5.23 *Let a, b and c be three distinct integers and $M = \max\{|a|, |b|, |c|\}$. Let ε be a real positive value such that $\varepsilon < \frac{1}{6M^2}$. Then there is no line that simultaneously intersects the horizontal segments $[a-\varepsilon, a+\varepsilon] \times a^3$, $[b-\varepsilon, b+\varepsilon] \times b^3$ and $[c-\varepsilon, c+\varepsilon] \times c^3$ unless the points (a, a^3), (b, b^3) and (c, c^3) are collinear.*

Theorem 5.24 *Given $d \geq 1$, it is 3SUM-hard to decide if $d_2(S) = d$.*

Proof. We will use a reduction from the 3SUM-problem similar to the 3SUM-hardness proof of the 3-POINTS-ON-LINE-problem [55]. Consider the set $X = \{x_1, \ldots, x_n\}$ of n integer numbers (positive and negative), an instance of the 3SUM-problem, and assume w.l.o.g. that $x_1 < \cdots < x_j < 0 < x_{j+1} < \cdots < x_n$ $(1 \leq j < n)$. Let $M = \max\{|x_1|, |x_n|\}$. If $d = 1$, put a blue point in $(-2M, 0)$ and a red point in $(2M, 0)$. If $d > 2$, then for each $1 \leq i \leq d-2$ put a red point in $(-2M - i + 1, 0)$ and a blue point in $(2M + i - 1, 0)$. Let ε be a small real positive number such that $\varepsilon < \frac{1}{6M^2}$. For each $1 \leq i \leq n$ put a red point p_i in $(x_i - \varepsilon, x_i^3)$ and a blue point q_i in $(x_i + \varepsilon, x_i^3)$; see Figure 5.10. Since $\varepsilon < \frac{1}{6M^2}$ we obtain by Lemma 5.23 that there is a line separating three distinct pairs (p_i, q_i), (p_j, q_j), and (p_k, q_k) if and only if (x_i, x_i^3), (x_j, x_j^3), and (x_k, x_k^3) are collinear (i.e., $x_i + x_j + x_k = 0$). Let S be the set of red an blue points as above. We have that $d_2(S) \geq d$ because $d_2(S, \Pi_\ell) = d$ for every line ℓ separating exactly two distinct pairs (p_i, q_i) and (p_j, q_j). If $d_2(S) > d$, then there is a line separating more than two pairs, implying that three elements in X sum to zero. Therefore, three elements in X sum to zero if and only if $d_2(S) \neq d$. ■

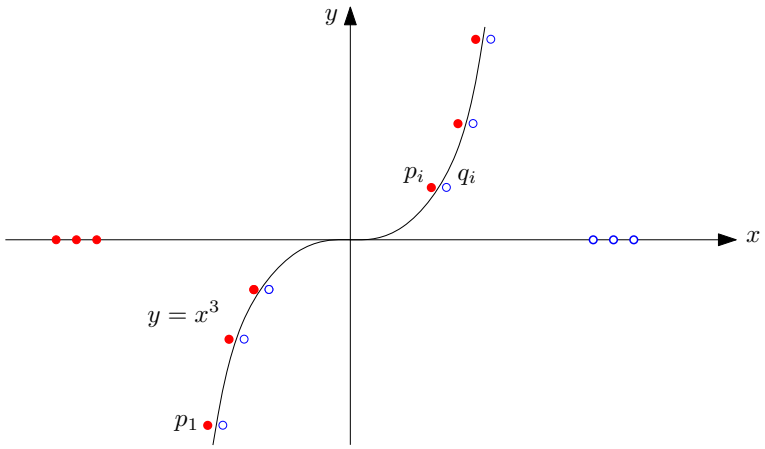


Figure 5.10: Reduction from 3SUM-problem when $d = 5$.

Theorem 5.25 *Computing the linear discrepancy of a bichromatic point set is 3SUM-hard and it can be done in $O(n^2)$ time.*

Proof. The hardness is due to Theorem 5.24, and duality can be used to find $d_2(S)$. ■

5.4.2 The Weak Separator problem

Given a bichromatic set of points in the plane, the *Weak Separator Problem* (WS-problem) looks for a line that maximizes the sum of the number of blue points on one side of it and the number of the red points on the other. The WS-problem can be solved in $O(n^2)$ [70] or in $O(nk \log k + n \log n)$ time [50], where k is the number of misclassified points. An $O((n+k^2) \log n)$ expected-time algorithm was presented recently in [29]. We prove that the WS-problem is 3SUM-hard.

Lemma 5.26 *Let $S = R \cup B$ such that $r = b$. Solving the WS-problem for S is equivalent to finding a line ℓ such that $d(S, \Pi_\ell) = d_2(S)$.*

Proof. Let ℓ be any line such that $d(S, \Pi_\ell) = d_2(S)$. Since $r = b$ then $\nabla'(S_{\ell^+}) = -\nabla'(S_{\ell^-})$. Suppose w.l.o.g. that $d_2(S, \Pi_\ell) = \nabla'(S_{\ell^+}) = |S_{\ell^+} \cap R| - |S_{\ell^+} \cap B| > 0$. We have that $|S_{\ell^+} \cap R| + |S_{\ell^-} \cap B| = |S_{\ell^+} \cap R| + |B| - |S_{\ell^+} \cap B| = b + |S_{\ell^+} \cap R| - |S_{\ell^+} \cap B|$. Hence $|S_{\ell^+} \cap R| + |S_{\ell^-} \cap B|$ is maximum if and only if $|S_{\ell^+} \cap R| - |S_{\ell^+} \cap B| = d_2(S)$ is maximum. ∎

Theorem 5.27 *The WS-problem is 3SUM-hard.*

5.5 Conclusions and further research

In this chapter we have presented a new parameter to measure how blended a bichromatic set of points is. Basically, we introduced a new concept of discrepancy for bicolored point sets that uses convex partitions of the points to determine if the set is good or not for clustering.

We proved combinatorial properties of the discrepancy, and provided a complete characterization if R and B are linearly separable. As an interesting result, it was shown that the discrepancy of points in convex position can be computed in $O(n \log n)$ time by using a reduction to instances of problems on circular sequences of weighted elements. The case in which the discrepancy is induced by partitions with a straight line was also studied, and we gave exact combinatorial lower and upper bounds of the value of discrepancy. Furthermore, we showed that computing this type of discrepancy is 3SUM-hard [55]. Additionally and as a consequence, we proved that the well-known *Weak Separator problem* [29, 50, 70] is also 3SUM-hard.

An important open problem of the present chapter is to give an exact algorithm or a hardness proof to the problem of computing the discrepancy of

a bichromatic point set. If the problem is hard, as we believe, it would be interesting to obtain approximation algorithms.

A similar definition of discrepancy of S, denoted as $d'(S)$, can be established by considering partitions Π of the plane into convex cells $C_1, C_2, \ldots, C_k$, instead of convex partitions of S. We can define $d'(S,\Pi) = \min_{i=1..k} \nabla(S \cap C_i)$ and $d'(S) = \max_{\Pi} d'(S,\Pi)$. It is easy to see that d' is a restricted version of d since we can induce a convex partition of S from any convex partition of the plane, and then $1 \leq d'(S) \leq d(S)$. In some sense, the optimal partition giving discrepancy $d'(S)$ can be viewed as a specific tessellation in which every cell has a dominant color. This version of discrepancy could be useful, for example, for coloring the plane. Furthermore, most of the results we have presented here are also valid for this new version of discrepancy. It is worthy to study the hardness of computing both versions of discrepancy for general configurations of bicolored point sets.

We leave also to further research the study of some specific types of discrepancy. For example, if we have a set of k parallel lines dividing the point set S into $k+1$ groups, then those groups form a convex partition of S (Figure 5.11 a)). Then, we can measure the discrepancy of S by only considering the convex partitions generated by parallel lines. Other variants of discrepancy can be studied if we only use the convex partitions generated by: a binary partition of the plane (Figure 5.11 b)), a straight binary partition of the plane (Figure 5.11 c)), a rectangular grid (Figure 5.11 d)), and a set of of pairwise disjoint boxes (Figure 5.11 e)).

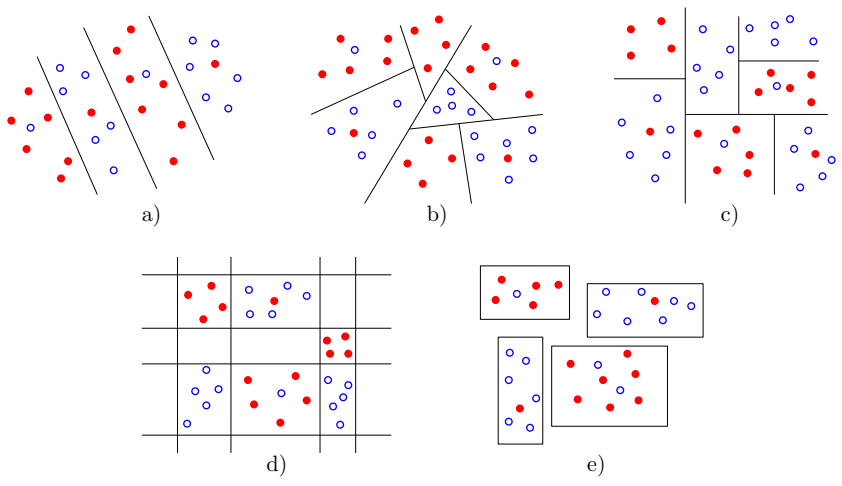


Figure 5.11: Different types of discrepancy of a point set induced by: a) a set of parallel lines, b) a binary partition of the plane, c) a straight binary partition of the plane, d) a rectangular grid, and e) a set of pairwise disjoint boxes.

5.6 Appendix

Proof of Lemma 5.23

Suppose w.l.o.g. that $a < b < c$. For a given $\varepsilon > 0$ denote as s_a, s_b, and s_c the horizontal segments $[a-\varepsilon, a+\varepsilon]\times a^3$, $[b-\varepsilon, b+\varepsilon]\times b^3$, and $[c-\varepsilon, c+\varepsilon]\times c^3$, respectively. Let $\delta(b, \overline{ac})$ be the horizontal distance from (b, b^3) to the line through (a, a^3) and (c, c^3), then:

$$\begin{aligned}
\delta(b,\overline{ac}) &= \left| b - \left((b^3 - a^3)\frac{c-a}{c^3-a^3} + a \right) \right| \\
&= \left| b - a - \frac{(b-a)(b^2+ab+a^2)}{c^2+ac+a^2} \right| \\
&= \left| (b-a)\left(1 - \frac{b^2+ab+a^2}{c^2+ac+a^2}\right) \right| \\
&= \left| (b-a)\left(\frac{c^2-b^2+ac-ab}{c^2+ac+a^2}\right) \right| \\
&= \left| \frac{(b-a)(c-b)(a+b+c)}{c^2+ac+a^2} \right| \\
&= (b-a)(c-b)\frac{|a+b+c|}{|c^2+ac+a^2|}
\end{aligned}$$

If $a+b+c = 0$ then $\delta(b, \overline{ac}) = 0$, and thus (a, a^3), (b, b^3), and (c, c^3) are collinear, and for all $\varepsilon > 0$ the line through them intersects the segments s_a, s_b, and s_c.

Now, suppose that $a+b+c \neq 0$ (i.e. $|a+b+c| \geq 1$). Since $a < b < c$ we have that $b-a \geq 1$ and that $c-b \geq 1$. Therefore:

$$\delta(b,\overline{ac}) \geq \frac{1}{|c^2+ac+a^2|} \geq \frac{1}{|c|^2+|a||c|+|a|^2} \geq \frac{1}{3M^2}$$

Note that for a given $\varepsilon > 0$ there is no line that intersects s_a, s_b, and s_c if and only if $2\varepsilon < \delta(b, \overline{ac})$. This can be ensured if $\varepsilon < \frac{1}{6M^2}$. Hence, the result follows.

Bibliography

[1] P.K. Agarwal, B. Aronov, and V. Koltun. Efficient algorithms for bichromatic separability. *ACM Transactions on Algorithms* (*TALG*), Vol. 2, No. 2, pp. 209–227, 2006.

[2] P.K. Agarwal and J. Pach. Combinatorial Geometry. *Wiley-Interscience Series in Discrete Mathematics and Optimization*, Vol. 16, pp. 267–290, 1995.

[3] P.K. Agarwal and M. Sharir. Efficient Algorithms for Geometric Optimization. *ACM Computing Surveys,* Vol. 30, No. 4, 1998.

[4] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Paghavan. Automatic subspace clustering of high dimensional data for data mining applications. *ACM SIGMOD International Conference on Management of Data*, 1998.

[5] M. Ali Abam. *New Data Structures and Algorithms for Mobile Data.* Ph. Thesis. Advisor: M. de Berg. Technische Universiteit Eindhoven, Eindhoven, 2007.

[6] L. Allison. Longest Biased Interval and Longest Non-Negative Sum Interval. *Bioinformatics*, Vol. 19, pp. 1294—1295, 2003.

[7] E. M. Arkin, G. Barequet, and J. S. B. Mitchell. Algorithms for Two-Box Covering. *Proceedings SoGC'06*, Arizona, USA, 2006.

[8] E.M. Arkin, F. Hurtado, J.S.B. Mitchell, C. Seara, and S.S. Skiena. Some lower bounds on geometric separability problems. International Journal of Computational Geometry and Applications. Vol. 16, No. 1, pp. 1-26, 2006.

[9] B. Aronov, E. Ezra, and M. Shair. Small-size ε-nets for axis-parallel rectangles and boxes. *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, Bethesda, USA, 2009.

[10] B. Aronov and S. Har-Peled. On approximating the depth and related problems. SIAM J. Comput. Vol. 38, pp. 899–921, 2008.

[11] M.J. Atallah. Some dynamic computational geometry problems. *Comput. Math. Appl.*, Vol. 11, No. 12, pp. 1171—1181, 1985.

[12] L.J. Aupperle, H.E. Corm, J.M. Keil, and J. O'Rourke. Covering Orthogonal Polygons with Squares. *In Proc. 26th Annu. Allerton Conf. on Communications, Control and Computing*, pp. 97–106, 1988.

[13] J. Basch, L. J. Guibas, and J. Hershberger. Data structures for mobile data. *Journal of Algorithms*, Vol. 31, pp. 1—28, 1999.

[14] C. Bautista-Santiago, J.M. Díaz-Báñez, D. Lara, P. Pérez-Lantero, J. Urrutia, and I. Ventura. Computing Maximal Islands. *Proceedings of the 25th European Workshop on Computational Geometry - EWCG'09*, Brusseles, Belgium, 2009.

[15] C. Bautista-Santiago, J.M. Díaz-Báñez, D. Lara, C. Peláez, and J. Urrutia. On Covering a Class with Arbitrary Disks. *Technical Report.*

[16] K.P. Bennett and E. Bredensteiner. Geometry in Learning. *In Geometry at Work.* Mathematical Association of America. Washington, D.C., 1998.

[17] J. L. Bentley. Programming pearls: algorithm design techniques. *Comm. ACM*, Vol. 27, No. 9, pp. 865–873, 1984.

[18] J. L. Bentley and M. I. Shamos. A problem in multivariate statistics: Algorithms, data structure and applications. *Proceedings of the 15th annual Allerton Conference on Communications, Control, and Computing*, pp. 193–201, 1977.

[19] S. Bespamyatnikh and M. Segal. Covering a set of points by two axis-parallel boxes. *Information Processing Letters*, Vol. 75, pp. 95–100, 2000.

[20] B.K. Bhathacharya. Circular Separability of Planar Point Sets. *Computatioanl Morphology*, G.T. Toussaint ,editor. North Holland, 1988.

[21] P. Bonnet, J. Gehrke, P. Seshadri. Towards Sensor Database Systems. *Proceedings of the Second International Conference on Mobile Data Management, Hong Kong.* Lecture Notes Comp. Sci., Vol. 1987, pp. 3–14, 2001.

[22] E. Boros, P.L. Hammer, T. Ibaraki, and A. Kogan. Logical analysis of numerical data. *Mathematical Programming*, Vol. 79, pp. 163—190, 1997.

[23] S. Brakatsoulas, D. Pfoser, and N. Tryfona. Modeling, storing, and mining moving object databases. *In Proc. International Database Engineering and Applications Symposium (IDEAS)*, pp. 68—77, 2004.

[24] H. Brönnimann and M.T. Goodrich. Almost optimal set covers in finite VC-dimension. *Discrete and Computational Geometry*, Vol. 14, pp. 463–479, 1995.

[25] H. Brönnimann and J. Lenchner. Fast almost-linear-sized nets for boxes in the plane. *In Proc. 14th Annu. Fall Workshop Comput. Geom.*, pp. 36—38, 2004.

[26] S. Cabello, J.M. Díaz-Báñez, C. Seara, J.A. Sellarès, J. Urrutia and I. Ventura. Covering point sets with two disjoint disks or squares. *Computational Geometry: Theory and Applications*, Vol. 40, No. 3, pp. 195–206, 2008.

[27] A.H. Cannon and L.J. Cowen. Approximation algorithms for the class cover problem. *Annals of Mathematics and Artificial Intelligence*, Vol. 40, pp. 215–223, 2004.

[28] E. Carrizosa y B. Martín. Problemas de clasificación: una mirada desde la localización. *Avances en Localización de Servicios y sus aplicaciones*, Universidad de Murcia, pp. 249–275, 2004.

[29] T. M. Chan. Low-Dimensional Linear Programming with Violations. *SIAM Journal on Computing*, Vol. 34, No. 4, pp. 879–893, 2005.

[30] B. Chazelle. The Discrepancy Method in Computational Geometry. *Handbook of Discrete and Computational Geometry*, Second Edition. J. E. Goodman and J. O'Rourke, editors. CRC Press 44, pp. 983–996, 2004.

[31] B. Chazelle and L. J. Guibas. Fractional cascading: A data structuring technique. *Algorithmica*, No. 1, pp. 133–162, 1986.

[32] K.-Y. Chen and K.-M. Chao. On the range maximum-sum segment query problem. *R. Fleischer and G. Trippen Eds., ISAAC 2004, LNCS 3341*, pp. 294–305, 2004.

[33] A. Civilis, C. S. Jensen, S. Pakalnis. Techniques for Efficient Road-Network-Based Tracking of Moving Objects. *IEEE Transactions On Knowledge And Data Engineering*, Vol. 17, No. 5, pp. 698–712, 2005.

[34] K.L. Clarkson and K. Varadarajan. Improved Approximation Algorithms for Geometric Set Cover. *Discrete and Computational Geometry*, Vol. 37, pp. 43–58, 2007.

[35] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*, Second Edition. MIT Press, McGraw-Hill, 2001.

[36] C. Cortés, J. M. Díaz-Báñez, and J. Urrutia. Finding enclosing boxes with empty intersection. *Proceedings of the 23rd. European Workshop on Computational Geometry*, Delphi (Greece), pp. 185–188, 2006.

[37] Y. Crama, P.L. Hammer, and T. Ibaraki. Cause-effect relationships and partially defined Boolean functions. *Annals of Operations Research*, Vol. 16, pp. 299—325, 1988.

[38] J. C. Culberson and R. A.Reckhow. Covering polygons is hard. *J. Algorithms*, Vol. 17, pp. 2-–44, 1994.

[39] J. Czyzowicz, E. Rivera Campo, J. Urrutia, and J. Zaks. Separating convex sets in the plane. *Discrete and Computational Geometry*, Vol. 7, pp. 189–195, 1992.

[40] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, Third Edition, Springer - Verlag, 2008.

[41] J.G. Devinney and C. Priebe. The class cover problem and its applications in pattern recognition. *The Johns Hopkins University*, 2003.

[42] D.P. Dobkin and D. Gunopulos. Concept learning with geometric hypotheses. *Proceedings of the 8th Annu. Conference on Computational Learning Theory*, ACM Press, pp. 329–336, 1995.

[43] D.P. Dobkin and D. Gunopulos. Geometric Problems in Machine Learning. *Selected papers from the Workshop on Applied Computational Geormetry, Towards Geometric Engineering*, Vol. 1148, pp. 121–132, 1996.

[44] D.P. Dobkin, D. Gunopulos, and W. Maass. Computing the maximum bichromatic discrepancy, with applications to computer graphics and

machine learning. *J. Computer and Systems Sciences*, Vol. 52, No. 3, pp. 453–470, 1996.

[45] D.P. Dobkin, D. Eppstein, and D.P. Mitchell. Computing the discrepancy with applications to supersampling patterns. *ACM Transactions on Graphics* (*TOG*), Vol. 15, No.4, pp. 354–376, 1996.

[46] R. Duda, P. Hart, and D. Stork. *Pattern classification. John Wiley and Sons, Inc.*, New York, 2001.

[47] A. Dumitrescu and J. Pach. Partitioning Colored Point Sets into Monochromatic Parts. *F. Dehne, J.-R. Sack, and R. Tamassia (Eds.): WADS 2001, LNCS 2125*, pp. 264—275, 2001.

[48] J. Eckstein, P. L. Hammer, Y. Liu, M. Nediak, and B. Simeone. The maximum box problem and its applications to data analysis. *Comput. Optim. Appl.*, Vo. 23, pp. 285–298, 2002.

[49] H. Edelsbrunner and F. P. Preparata. Minimum polygonal separation. *Inform. Comput.*, Vol. 77, pp. 218–232, 1988.

[50] H. Everett, J.M. Robert, M. Kreveld. An Optimal Algorithm for Computing ($\leq k$)-Levels, with Applications to Separation and Transversal Problems. *Int. J. Comput. Geom. Appl.*, Vol. 6, pp. 247–261, 1996.

[51] T.H. Fan, S. Lee, H.I. Lu, T.S. Tsou, T-C. Wang, and A. Yao. An Optimal Algorithm for Maximum-Sum Segment and Its Application in Bioinformatics. *CIAA, LNCS 2759*, pp. 251–257, 2003.

[52] U. Feige. A threshold of ln n for approximating set cover. Journal of ACM, Vol. 45, No. 4, pp. 634-–652, 1998.

[53] S. P. Fekete. On the complexity of min-link red-blue separation. *Manuscript*, Department of Applied Mathematics, SUNY Stony Brook, NY, 1992.

[54] M.L. Fredman. On computing the length of longest increasing subsequences. *Discrete Mathematics*, Vol. 11, pp. 29–35, 1975.

[55] A. Gajentaan and M. H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Computational Geometry: Theory and Applications*, Vol. 5, pp. 165–185, 1995.

[56] V. García, J.S. Garreta, R.M. Cardenas, R. Alejo, and J.M. Sotoca. The class imbalance problem in pattern classification and learning. *CEDI 2007, II Congreso Español de Informatica*. Zaragoza, 2007.

[57] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, 1979.

[58] J. E. Goodman and J. O'Rourke, editors. *Handbook of Discrete and Computational Geometry*, CRC Press LLC, Boca Raton, FL, 1997.

[59] Q. Gu, Z. Cai, L. Zhu, and B. Huang. Data Mining on Imbalanced Data Sets. *2008 International Conference on Advanced Computer Theory and Engineering*. ICACTE, pp. 1020–1024, 2008

[60] L.J. Guibas. Kinetic Data Structures. *Handbook Of Data Structures And Applications (Chapman & Hall/Crc Computer and Information Science Series.)*. D.P. Mehta and S. Sahni, editors. Chapman & Hall/CRC, 2004.

[61] L.J. Guibas. Kinetic Data Structures: A State of the Art Report. *In Proceedings of the Third Workshop on the Algorithmic Foundations of Robotics on Robotics: the algorithmic perspective*. Houston, Texas, United States, pp. 191–209, 1998.

[62] P. Gupta, R. Janardan, and M. Smid. Fast algorithms for collision and proximity problems involving moving geometric objects. *Computational Geometry, Theory and Applications*, Vol. 6, pp. 371—391, 1996.

[63] P. L. Hammer and T. Bonates. Logical analysis of data: from combinatorial optimization to medical applicatios. *Rutcor Research Report, RRR 10-2005*, Rutger University, New Jersey, USA, 2005.

[64] P.L. Hammer, A. Kogan, B. Simeone, and S. Szedmak. Pareto-optimal patterns in logical analysis of data. *RUTCOR Research Report 7-2001*, 2001.

[65] J. Han and M. Kamber. Data mining: concepts and techniques, Morgan Kaufmann Publishers, USA, 2006.

[66] H. Hand, H. Mannila, and P. Smyth. *Principles of data mining*. The MIT Press, 2001.

[67] S. Har-Peled. Being Fat and Friendly is Not Enough. CoRR, 2009.

[68] D. Haussler and E. Welzl. ε-nets and simplex range queries. *Discrete Computational Geometry*, Vol. 2, pp. 127–151, 1987.

[69] J.E. Hopcroft and R.M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SICOMP*, Vol. 2, pp. 225—231, 1973.

[70] M. E. Houle. Algorithms for weak and wide separation of sets. *Proceedings of the International Workshop on Discrete Algorithms and Computation*, pp. 61—68, 1989.

[71] X. Huang. An Algorithm for Identifying Regions of a DNA Sequence that Satisfy a Content Requirement. *CABIOS*, Vol. 10, pp. 219–225, 1994.

[72] P. Indyk. *High-dimensional computational geometry.* Ph. Thesis. Advisor: R. Motwani. Stanford University Stanford, CA, USA, 2001.

[73] A. Kaneko and M. Kano, Discrete geometry on red and blue points in the plane–a survey. *In Discrete and Computational Geometry, The Goodman-Pollack Festschrift, Vol. 25 of Algorithms and Combinatorics, Springer-Verlag*, pp. 551–570, 2003.

[74] M. Kano and M. Uno. General Balanced Subdivision of Two Sets of Points in the Plane. *J. Akiyama et al. (Eds.): CJCDGCGT 2005, LNCS 4381*, pp. 79–87, 2007.

[75] S. Kotsiantis. Educational data mining: a case study for predicting dropout-prone students. *International Journal of Knowledge Engineering and Soft Data Paradigms.* Vol. 1, No. 2, pp. 101–111, 2009

[76] D. E. Knuth. *Sorting and searching. The art of computer programming.* Addison-Wesley, 2000.

[77] D. E. Knuth and A. Raghunathan. The problem of compatible representatives. *SIAM Journal of Discrete Mathematics.* Vol. 5, No. 3, pp. 422–427, 1992.

[78] S.R. Kosaraju, N. Megiddo, and J. O'Rourke. Computing Circular Separability. Discrete Computatioanl Geometry. Vol. 1, pp. 105–113, 1986.

[79] L.V.S. Lakshmanan, R.T. Ng, C.X. Wang, X. Zhou, and T.J. Johnson. The generalized MDL approach for summarization. *International Conference on Very Large Data Bases*, 2002.

[80] P.F. Lambert. Designing Pattern Categorizers with Extremal Paradigm Information. *Proceedings of the International Conference on Mehtodologies of Pattern Recognition*, pp. 359–391, 1969.

[81] D. Lichtenstein. Planar formulae and their uses. *SIAM Journal of Computing.* Vol. 11, No. 2, pp. 329–343, 1982.

[82] Y.L. Lin, T. Jiang, and K.M. Chao. Efficient Algorithms for Locating the Length-constrained Heaviest Segments with Applications to Biomolecular Sequence Analysis. *Journal of Computer and System Sciences*, Vol. 65, pp. 570–586, 2002.

[83] Y. Liu and M. Nediak. Planar Case of the Maximum Box and Related Problems. *Proceedings of the Canadian Conference on Computational Geometry*, Halifax, Nova Scotia, 2003.

[84] D. Marchette. Class cover catch digraphs. *Wiley Interdisciplinary Reviews: Computational Statistics*, Vol. 2, No. 2, pp. 171–177, 2010.

[85] W. J. Masek. Some NP-complete set covering problems. Manuscript, MIT, Cambridge, MA, 1979.

[86] J. Matoušek. *Geometric Discrepancy: An Illustrated Guide.* Springer-Verlag, 1999.

[87] J. Matoušek, R. Seidel, and E. Welzl. How to Net a Lot with Little: Small ε-Nets for Disks and Halfspaces. *Proceedings of the 6th Annual ACM Symposium on Computational Geometry*, pp. 16–22, 2000.

[88] J.S.B. Mitchell. Approximation algorithms for geometric separation problems. Technical report, Dept. of Applied Math. and Statistics, State U. of New York at Stony Brook, July 1993.

[89] M. S. Paterson and F. F. Yao. Optimal binary space partitions for orthogonal objects. *Journal of Algorithms*, Vol. 13, pp. 99–113, 1992.

[90] S. Har-Peled and and V. Koltun. Separability with Outliers. *Lecture Notes in Computer Science*, Vol. 3827, pp. 28–39, 2005.

[91] F.P. Preparata and M. I. Shamos. *Computacional Geometry, An Introduction*, Springer-Verlag, 1988.

[92] J.R. Sack and J. Urrutia, editors. *Handbook of Computational Geometry*, North-Holland, Elsevier, 2000.

[93] E. Schömer and C. Thiel. Efficient collision detection for moving polyhedra. *In Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pp. 51—60, 1995.

[94] C. Seara. *On geometric separability*. Ph. Thesis. Advisor: F. Hurtado. Universitat Politècnica de Catalunya, Barcelona, 2002.

[95] M. Segal. Planar Maximum Box Problem. Journal of Mathematical Modeling and Algortihms. Vol. 3, pp. 31–38, 2004.

[96] A.P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and querying moving objects. *In Proc. International Conference on Data Engineering (ICDE)*, pp. 422—432, 1997.

[97] A.P. Sistla, O. Wolfson, S. Chamberlain, and Y. Yesha. Updating and querying databases that track mobile units. *Distributed and Parallel Databases*, Vol. 7, No. 3, pp. 257—387, 1999.

[98] T. Takaoka. Efficient Algorithms for the Maximum Subarray Problem by Distance Matrix Multiplication. *Electronic Notes in Theoretical Computer Science*, Vol. 61, pp. 191–200, 2002.

[99] H. Tamaki and T. Tokuyama. Algorithms for the Maximum Subarray Problem Based on Matrix Multiplication. *Interdisciplinary Information Sciences*, Vol. 6, No. 2, pp. 99–104, 2000.

[100] V.V. Vazirani. *Approximation algorithms*. Springer-Verlag New York, Inc., New York, NY, 2001.

[101] V. N. Vapnik and A. Ya. Červonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory Probab. Appl.*, Vol. 16, pp. 264–280, 1971.