Algorithms for Colourful Simplicial Depth and Median in the Plane

by

Olga Zasenko

B.Sc., Taras Shevchenko National University of Kiev, 2014

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
Department of Mathematics
Faculty of Science

© Olga Zasenko 2017
SIMON FRASER UNIVERSITY
Spring 2017

Approval

Name:	**Olga Zasenko**
Degree:	**Master of Science**
Title:	***Algorithms for Colourful Simplicial Depth and Median in the Plane***
Examining Committee:	**Chair:** Natalia Kouzniak Senior Lecturer

Tamon Stephen
Senior Supervisor
Associate Professor

Zhaosong Lu
Supervisor
Associate Professor

Michael Monagan
Internal Examiner
Professor

Date Defended: 7 April 2017

Abstract

The *colourful simplicial depth* (CSD) of a point $x \in \mathbb{R}^2$ relative to a configuration $P = (P^1, P^2, \ldots, P^k)$ of n points in k colour classes is exactly the number of closed simplices (triangles) with vertices from 3 different colour classes that contain x in their convex hull. We consider the problems of efficiently computing the colourful simplicial depth of a point x, and of finding a point in $\mathbb{R}^2$, called a *median*, that maximizes colourful simplicial depth.

For computing the colourful simplicial depth of x, our algorithm runs in time $O\left(n \log n + kn\right)$ in general, and $O(kn)$ if the points are sorted around x. For finding the colourful median, we get a time of $O(n^4)$. For comparison, the running times of the best known algorithm for the monochrome version of these problems are $O\left(n \log n\right)$ in general, improving to $O(n)$ if the points are sorted around x for monochrome depth, and $O(n^4)$ for finding a monochrome median.

Keywords: Computational geometry; data depth; colourful simplicial depth; bivariate medians; topological sweep

Dedication

To my grandmother, the strongest woman I know who achieved so much in her life, despite all the obstacles. You are my mentor, my strongest inspiration, and my most empowering supporter. I love you.

Acknowledgements

I would like to thank my supervisor Tamon Stephen for continuous support and guidance throughout my time as a student at SFU. Reaching this milestone would not be possible without having someone to look up to, to ask for advice or feedback. His expertise and knowledge were my source of truth when in doubt while writing a paper or this thesis. I also thank Dr. Stephen for supporting my decision to complete two co-op terms while doing my Master's at SFU.

My sincere thanks goes to my examiner Dr. Monagan for reading and editing this thesis, and for sharing great questions and suggestions. The small but important details that we have missed couldn't escape from his vigilant eye.

A very special gratitude goes out to Natalia Kouzniak who made moving to a foreign country and adjusting to the local university culture for me much easier. Also she was the only person I could speak to in my mother tongue, which I miss very much.

I express my gratitude to all the people who were by my side these last couple of years. Although most of my family is far away, I always feel their encouragement and confidence in my efforts, just as if they were here. It is truly astonishing that they never get tired of me talking about research and writing during our Skype calls. I am grateful for having such caring friends who are always there for me. They give me the strength to set goals and achieve them.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Colourful Simplicial Depth and Medians

The *simplicial depth* $D(x, P)$ of a point $x \in \mathbb{R}^2$ relative to a data set P is exactly the number of simplices (triangles) formed with the points from P that contain x in their convex hull (Figure 1.1). A *simplicial median* of the set P is any point in $\mathbb{R}^2$ which is contained in the most triangles formed by elements of P, i.e. has maximum simplicial depth with respect to P.

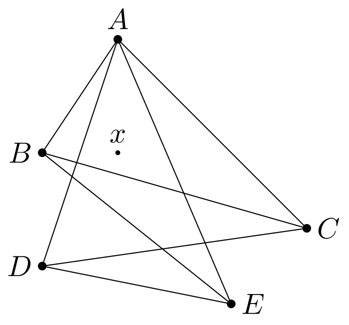


Figure 1.1: Point x is contained in exactly 4 triangles $\triangle ABC$, $\triangle ABE$, $\triangle ADC$, $\triangle ADE$, therefore, $D(x) = 4$.

In this thesis we consider a set P in $\mathbb{R}^2$ that consists of k colour classes $P^1, \ldots, P^k$, where $k \geq 3$. The *colourful simplicial depth* $\hat{D}(x, P)$ of $x \in \mathbb{R}^2$ with respect to configuration P is the number of triangles with vertices from 3 different colour classes that contain x (Figure 1.2). We call such triangles *colourful triangles*. A *colourful simplicial median* of a configuration $P = (P^1, P^2, \ldots, P^k)$ is any point in the convex hull of P with maximum colourful simplicial depth.

For the median, we illustrate an example in Figure 1.3, where a median is unique and lies at the point G_2. Another example is Figure 1.4, where median is not unique – it lies at the points E and F. Moreover, in Figure 1.5 a median is three points D, E, and F, whose

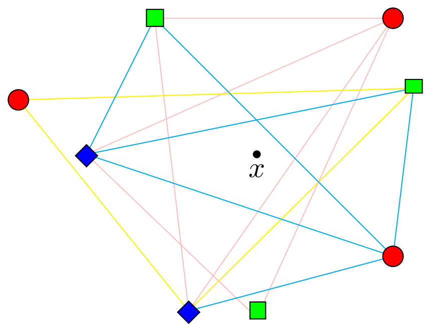


Figure 1.2: A configuration P of 8 points in $\mathbb{R}^2$ surrounding a point x with $\hat{D}(x, P) = 6$.

depth is also 5. In Figure 1.6 E and G are the deepest points with CSD of 7, but the line joining them contains 2 other vertices whose CSD is equal to 6.

Even though a colourful simplicial median is usually a discrete set of points, in some fairly trivial cases it can be a convex set. For instance, in Fig. 1.7a colourful simplicial median is a line, and in Fig. 1.7b it is a triangle.

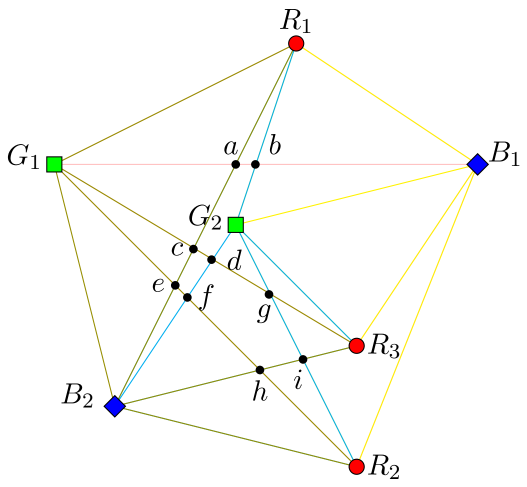


Figure 1.3: A configuration P of 7 points in $\mathbb{R}^2$, whose unique simplicial median is G_2 which has depth 8.

Our objective is to find efficient algorithms for finding both the colourful simplicial depth of a given point x with respect to a configuration P, and a colourful simplicial median of $P \in \mathbb{R}^2$.

1.2 Background

1.2.1 Facts about Simplicial Depth and Medians

We begin with a discussion of the *simplicial depth* introduced by Liu in 1990 [Liu90], [LPS99], which is one of several notions of data depth. Up to a constant, it can be interpreted as the

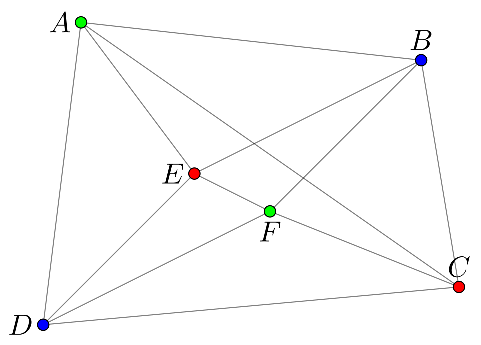


Figure 1.4: A configuration P of 6 points in 3 colours, where E and F have maximum CSD of 5.

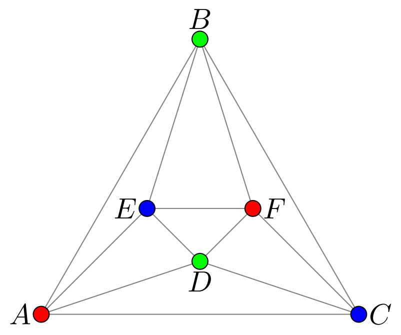


Figure 1.5: A configuration of 6 points in 3 colours, where D, E and F have maximum CSD of 5.

probability that a given point x lies inside the convex hull of a random simplex generated by the points in a data set P. Suppose X_1, X_2, and X_3 are three independent observations from P. Then for any point x in $\mathbb{R}^2$, the following function calculates the simplicial depth of x: $Prob(x \in \triangle(X_1, X_2, X_3)) = 6D(x)/|P|^3$. The simplicial depth measures the relative position of a point with respect to a distribution, and hence reflects its underlying probabilistic geometry. Liu calls it the "insideness" of a point relative to P, marking that points near the "centre" of the distribution should be more likely to lie within the random triangle $\triangle(X_1, X_2, X_3)$. Hence the centre is a point x that maximizes $D(x)$; it is a type of bivariate median.

The colourful version of the simplicial depth [DHST06], generalizes this notion to selecting points from k distributions. And the colourful version of the bivariate median in turn generalizes it to finding the deepest point among k distributions with respect to all of them. Suppose we have a colourful configuration $P = \{P_1, P_2, P_3\} \in \mathbb{R}^2$ and $X_1 \in P_1, X_2 \in P_2, X_3 \in P_3$ are independent uniformly distributed observations from the corresponding data sets, then colourful simplicial depth of a point x with respect to P can be written as $Prob(x \in \triangle(X_1, X_2, X_3)) = \hat{D}(x)/[|P_1||P_2||P_3|]$. All mentioned concepts extend to $\mathbb{R}^d$ and are natural objects of study in discrete geometry. The simplicial depth of x

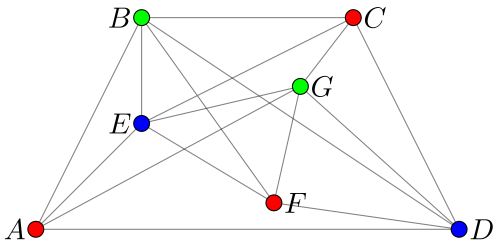
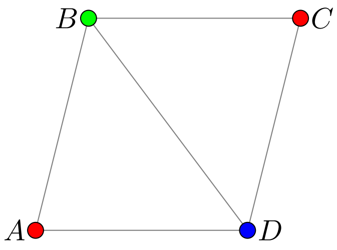
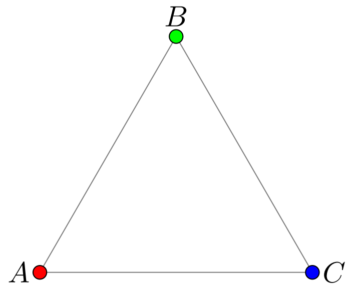


Figure 1.6: A configuration of 7 points in 3 colours, where E and G have maximum CSD of 7.



(a) A configuration of 4 points in 3 colours, where the whole line BD has CSD of 2.



(b) A configuration of 3 points in 3 colours, where the whole triangle $\triangle ABC$ has CSD of 1.

Figure 1.7: Trivial cases when colourful simplicial median is a convex set.

in $\mathbb{R}^d$ can be defined as $D(x) = Prob(x \in S[X_1, \dots, X_{d+1}])$, where $X_1, \dots, X_{d+1}$ are independent and identically distributed observations from P and $S[X_1, \dots, X_{d+1}]$ is the simplex with vertices $X_1, \dots, X_{d+1}$. Then colourful simplicial depth of a point x with respect to $P = \{P_1, P_2, \dots, P_{d+1}\} \in \mathbb{R}^d$ can be written as $\hat{D}(x) = Prob(x \in S[X_1, X_2, \dots, X_{d+1}])$, where $X_1 \in P_1, X_2 \in P_2, \dots, X_{d+1} \in P_{d+1}$ are independent and identically distributed observations from the corresponding data sets. The colourful simplicial depth represents the number of basic solutions to a colourful linear programming problem, see [BO97, DHST08]. Applications of colourful linear programming include computing Nash equilibria in a bimatrix game [MS17].

The medians are central points which are in some sense most representative of the distribution(s). They are also called *location estimators*, i.e. points that best describe or *estimate* the data.

For more background on simplicial depth (SD) and competing measures of data depth, see [Alo06] and [FR05]. Monochrome depth has seen a flurry of activity in the past few years, most notably relating to the *First Selection Lemma*, which is a lower bound for the depth of the median, see e.g. [MW14]. Among the recent work on colourful depth are proofs of the lower [Sar15] and upper [ABP$^+$16] bounds for depth of colourful medians conjectured by Deza et al. [DHST06].

1.2.1.1 Data Depth

Data depth was originally introduced as a way to generalize the concepts of median and quantiles to a multivariate framework. Many definitions of the data depth have been introduced in recent years, along with the algorithms to compute them. We present only some of these definitions below:

- *Tukey's depth* [Tuk75] also known as *half-space depth* of a point x is the smallest number of data points in a closed half-space with boundary through x.

- *Oja's depth* [Oja83] of a point x relative to a data set P in $\mathbb{R}^d$ is the cumulative volume of all simplices formed by x and a subset of d elements from P.

- *Convex hull peeling depth* [Bar76] of a point x is the *level of the convex layer x* belongs to. At every step the points on the border of the convex hull are removed from the data set and the convex hull is reconstructed.

- *Simplicial depth* [Liu90] of a point x relative to the data set P is the number of simplices formed with the points from P that contain x.

There are several properties that are desirable in a multivariate depth measure. Zuo and Serfling [ZS00] investigated various depth functions on the subject whether these properties hold for them or not. The following are desirable properties for a data depth function:

1. Affine invariance.

2. Maximality at centre.

3. Monotonicity relative to deepest point.

4. Vanishing at infinity.

5. Computability.

We observe that (1) and (4) always hold for simplicial depth, whereas (2) and (3) fail in some discrete cases. A similar issue was later addressed by Fukuda and Rosta [FR05], who pointed out that the *depth contours* (the sets of data points whose simplicial depth is equal to a certain value) are not necessarily nested in case of simplicial depth, although they always are for half-space depth. This means that we cannot use standard methods such as *convex hull peeling* for finding a simplicial median. Interestingly, Burr et al. [BRS06] considered an alternate definition of simplicial depth that behaves better in the counterexamples discovered by Zuo and Serfling [ZS00]. However, properties (2) and (3) still can fail in some cases. Liu et al. [LPS99] points out that when there is a unique maximum of the simplicial depth, the depth contours are indeed nested within one another.

Finally, property (5) is about how efficiently we can compute the depth function. For simplicial depth in $\mathbb{R}^2$, good algorithms exist and are reviewed in Section 1.2.3. For general dimension, tractability is a serious issue.

A number of statistical methodologies based on data depth has been developed. For example, depth vs. depth plot or *DD-plot* plots the depth values of the combined sample under the two corresponding empirical distributions F and G [LPS99]. It is a very useful generalization of the one dimensional quantile-quantile plot. Plots which deviate from diagonal line from $(0,0)$ to $(1,1)$ in $\mathbb{R}^2$ indicate differences between the two distributions (Figure 1.8). Different deviation patterns in the plot correspond to different types of variations between the distributions. Some of the examples of these differences are:

- Location differences
- Scale differences
- Skewness differences
- Kurtosis differences

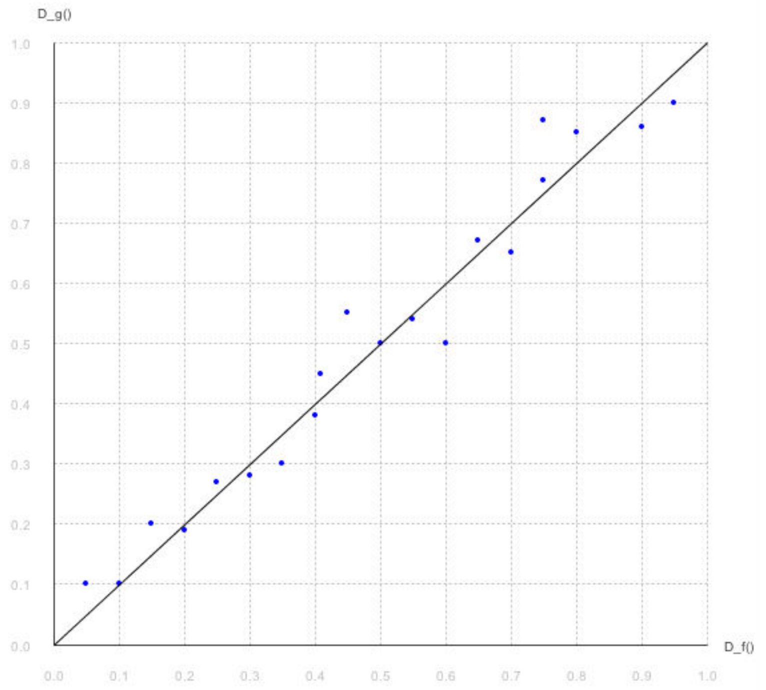


Figure 1.8: An example of a DD-plot of two distributions f and g.

To bring out some of these differences, certain properties of the distributions should be normalized first, such as centre, scale, location, etc. Liu, et al. [LPS99] also make a very important observation: “DD-plot with a common maximum point for both coordinates indicates a common centre (i.e., no location shift) for the two underlying samples.”

Data depth is motivated by multivariate analysis. For instance, a package "Nonparametric Depth Functions for Multivariate Analysis" [GMP08] is available for R – a language and environment for statistical computing and graphics. It is an open source package that allows calculation of depth values and depth-based location estimators, as well as drawing contour plots and perspective plots of depth functions. Depth functions covered are Tukey's, Liu's and Oja's. A package called "Statistical Depth Functions for Multivariate Analysis" [KBWZ16] offers a wider variety of depth functions: Euclidean depth, local depth, L^p depth, Mahalanobis depth, modified band depth, projection depth, Tukey depth; as well as drawing DD-plots, contour and perspective plots.

Another important factor is the *robustness* of the estimator, which is reflected by how much the estimator changes if some of the data is perturbed.

Definition 1.2.1. *The* breakdown point *is the proportion of data which must be moved to infinity so that the estimator will do the same.*

Donoho and Gasko present extensive research of the breakdown properties based on the half-space depth in [DG92]. Computational convenience for different definitions of a multidimensional median as well as common ideas of equivariance, symmetry and breakdown can be found in "A Survey of Multidimensional Medians" by Small [Sma90]. Mosler [Mos12] considers existing data depth definitions and suggests some approaches to define a multivariate depth statistic. Aloupis [Alo06] lists the main approaches of measuring the data depth along with their properties and algorithms for computing them.

1.2.2 Facts about Colourful Simplicial Depth and Medians

The colourful setting for simplicial depth is suggested by Bárány's approach [Bár82] to proving a colourful version of Carathéodory's theorem. Let us restate this famous theorem here:

Theorem 1.2.2. *Let $S_1, S_2, \dots S_{d+1}$ be $d+1$ sets in $\mathbb{R}^d$. Suppose that $x \in conv(S_1) \cap conv(S_2) \cap conv(S_3) \cap \dots \cap conv(S_{d+1})$. Then there are $x_1 \in S_1, x_2 \in S_2, \dots, x_{d+1} \in S_{d+1}$ such that $x \in conv(x_1, x_2, \dots, x_{d+1})$.*

Later Holmsen, Pach, and Tverberg [HPT08] and Arocha et. al [ABB$^+$09] independently discovered a generalization of this colourful theorem:

Theorem 1.2.3. *Let $S_1, S_2, \dots, S_{d+1}$ be $d+1$ sets in $\mathbb{R}^d$. Suppose that $x \in conv(S_i \cup S_j)$ for every $1 \le i < j \le d+1$. Then there are $x_1 \in S_1, x_2 \in S_2, \dots, x_{d+1} \in S_{d+1}$ such that $x \in conv(x_1, x_2, \dots, x_{d+1})$.*

Or, as Pach put it, "the right condition in the colored Carathéodory theorem is not that every color class contains the origin in its convex hull, but that the union of every pair of color classes contains the origin in its convex hull. This already guarantees that one can pick a point of each color so that the simplex induced by them contains the origin." Arocha, et. al [ABB$^+$09] called their generalization a *Very Colourful Carathéodory Theorem*.

Deza et al. [DHST06] formalized the notion of the colourful simplicial depth and considered its bounds of points in the intersection of the convex hulls of the colour sets or in a *core*. They conjectured that the minimum colourful simplicial depth of any point in the core in $\mathbb{R}^d$ is $d^2 + 1$ and that maximum is $d^{d+1} + 1$.

1.2.3 Known Algorithms for Simplicial Depth

The monochrome simplicial depth in $\mathbb{R}^d$ can be computed by enumerating simplices in time $O(n^{d+1})$, but in general dimension, it is quite challenging to compute it more efficiently [Alo06], [CO01], [FR05]. Several authors have considered the two-dimensional version of the problem, including Khuller and Mitchell [KM90], Gil, Steiger, and Wigderson [GSW92], and Rousseeuw and Ruts [RR96]. Each of these groups produced an algorithm that computes the monochrome depth in $O(n \log n)$ time, with sorting the input as the bottleneck. If the input points are sorted, these algorithms take linear time.

Gil, Steiger, and Wigderson [GSW92] proposed an algorithm that finds the simplicial depth of a point in $\mathbb{R}^3$ in $O(n^2)$ time. Later Cheng and Ouyang [CO01] discovered a slight flaw in it and corrected the algorithm with the running time still $O(n^2)$. They also suggested an adaptation of this approach in $\mathbb{R}^4$ and approximated the running time as $O(n^4)$.

1.2.4 Known Algorithms for Simplicial Medians

For simplicial medians, Khuller and Mitchell [KM90], and Gil et al. [GSW92] considered an *in-sample* version of a simplicial median, that is they looked for a point *from* P with maximum simplicial depth.

However, we consider a *simplicial median* to be *any* point $x \in \mathbb{R}^2$ maximizing the simplicial depth. Rousseeuw and Ruts [RR96] found an algorithm to compute a simplicial median in $O(n^5 \log n)$ time, Aloupis et al. [ALST03] improved this to $O(n^4)$. In their paper, Aloupis et al. [ALST03] compute the simplicial depth of all the intersection points of the segments formed by all possible pairs of points in the data set, then they also compute the simplicial depth of the data points in P. A point or a group of points with the maximum depth is called a simplicial median. They use a *topological sweep* [EG89] to discover all the intersection points, and several other techniques to improve the performance of their algorithm. The total running time is $O(n^4)$.

The problem of finding the pairwise intersection points of the lines from a given set is a well-studied problem. Bentley and Ottmann [BO79] proposed an algorithm that reports all k intersection points in $O(N \log N + k \log N)$ time, where N is the number of lines in the configuration, $N = O(n^2)$. Their approach was to sweep a straight line through an arrangement, tracking the order in which the segments intersect this line. If the order of two segments swaps, that means they've intersected. Edelsbrunner and Guibas came up with an idea of sweeping the arrangement with a topological line [EG89] which they did in $O(N^2)$ time and $O(N)$ space, where N is again the number of lines. A few years later Chazelle and Edelsbrunner presented "An Optimal Algorithm for Intersecting Line Segments in the Plane" [CE92]. It runs in $O(N \log N + k)$ time, where k is the number of intersection points and N is the number of segments. Their approach was to insert one segment at a time, checking for its intersection points with already inserted segments. The space required is $O(N+k)$ in the worst case, but they claim it to be much lower in practice. Furthermore, Balaban reduced the storage complexity to $O(N)$ [Bal95]. His strategy was to split the arrangement into strips and recursively apply an algorithm that finds intersection points within the strip. Moreover, Balaban's algorithm not only works on line segments, but also on curve segments.

1.2.5 Known Colourful Algorithms

The algorithm in [EG89], however, works with infinite lines, whereas we have bounded segments. Rafalin and Souvaine [RS08] came up with a novel algorithm to find all the intersection points in a complete graph. They have built upon the approach in [EG89], i.e. also used a topological line to sweep an arrangement. The difference is that this line could intersect a segment at most once, in contrast with an approach in [EG89], where the topological line intersects every line exactly once. The algorithm in [RS08] runs in $O(K + NM)$ time, where K is the complexity of the graph measured as the number of segments, N is the number of vertices in the graph, and M is the maximum number of edges cut by any vertical line. Here $N = |P| = n$ and $M \in O(n^2)$. K is the number of *graph segments* which are the graph edges that are delimited by two adjacent intersections or vertices along the graph edge (note these are different from segments of S). Therefore, $K \in O(n^4)$. We conclude that the running time of the topological sweep executed on our data is $O(n^4)$.

Moreover, it handles the degeneracies in the following way: dummy variables and additional information about the intersection points are added to the data structure. Rafalin and Souvaine [RS08] mention that the degeneracy improves the running time of their algorithm. However, such degeneracies as more than two lines intersect at one point increases the running time of our algorithm. Instead we use a technique called the *Simulation of Simplicity* described in [EM88]. It perturbs the data slightly by replacing each coordinate by a polynomial in ϵ, where ϵ is assumed to be positive and sufficiently small, which allows using it as an indeterminant and to handle primitive operations symbolically.

In addition, Rafalin and Souvaine [RS08] have demonstrated an application of their algorithm for computing the simplicial median in [ALST03]. Aloupis et al. [ALST03] used the topological sweep [EG89], but did not make it clear how to modify an algorithm that works with infinite lines to work with finite segments. Whereas Rafalin and Souvaine [RS08] made the change and suggested using their version of the topological sweep for finding the simplicial median.

1.3 Organization and Main Results

In Chapter 2, we develop an algorithm for computing colourful simplicial depth that runs in $O(n \log n + kn)$ time (Theorem 2.5.1), where k is the number of colours and n is the number of points. This retains the $O(n \log n)$ asymptotics of the monochrome algorithms when k is fixed. As in the monochrome case, sorting the initial input is a bottleneck, and the time drops to $O(kn)$ if the input is sorted around x. In this case, for fixed k, it is a linear time algorithm.

In Chapter 3, we turn our attention to computing a colourful simplicial median. We develop an algorithm that does this in $O(n^4)$ time using a topological sweep (Theorem 3.4.1). This is independent of k and matches the running time from the monotone case.

In Chapter 4 we give some thought to solving the same problems in $\mathbb{R}^3$. Chapter 5 contains conclusions and discussion about future directions.

An extended abstract containing preliminary versions of some of these results appeared in the conference proceedings of COCOA 2016 [ZS16].

Chapter 2

Computing Colourful Simplicial Depth

2.1 Preliminaries

We consider a family of sets P^1, P^2, ... , $P^k \subseteq \mathbb{R}^2$, $k \geq 3$, where each P^i consists of the points of some particular colour i. Refer to the j^{th} element of P^i as P^i_j. We use superscripts for colour classes, while subscripts indicate the position in the array.

We denote the union of all colour sets by P: $P = \bigcup\limits_{i=1}^{k} P^i$. The total number of points is n, where $|P^i| = n_i$, $\sum\limits_{i=1}^{k} n_i = n$. We assume that points of $P \bigcup \{x\}$ are in general position to avoid technicalities. Without loss of generality, we can take $x = \vec{0}$, the zero point. We will sometimes perform arithmetic operations on the subscripts, in which case the indices are taken modulo the size of the array i.e. (mod n_i).

Definition 2.1.1. *A* colourful triangle *is a triangle with one vertex of each colour, i.e. it is a triangle whose vertices v_1, v_2, v_3 are chosen from distinct sets P^{i_1}, P^{i_2}, P^{i_3}, where $i_i \neq i_2, i_3; i_2 \neq i_3$.*

Definition 2.1.2. *The* colourful simplicial depth *$\hat{D}(x, P)$ of a point x relative to the set P in $\mathbb{R}^2$ is the number of colourful triangles containing x in their convex hull. We reserve $D(x, P)$ for the (monochrome) simplicial depth, which counts all triangles from P regardless of the colours of their vertices.*

Remark 2.1.3. *We are checking containment in* closed *triangles. With our general position assumption, this will not affect the value of $\hat{D}(x, P)$. It is more natural to consider closed triangles than open triangles in defining colourful medians; the open triangles version of this question may also be interesting.*

Due to the fact that we consider closed triangular containments, when computing the CSD of a coloured point, we also add the number of colourful triangles it forms with the data points. In our case the point x is not assigned any colour, so we don't have to worry about those additional counts, but we will explain how to compute them in the end of Section 2.5.

Throughout the thesis we work with polar angles θ^i_j formed by the data points P^i_j and a fixed ray from x. We remark that simplicial containment does not change as points are moved on rays from x, see for example [ZS00]. Thus we can ignore the moduli of the P^i_j, and work entirely with the θ^i_j, which lie on the unit circle $\mathcal{C}$ with x as its origin. We will at times abuse notation, and not distinguish between P^i_j and θ^i_j.

Note that the ray taken to have angle 0 is arbitrary, and may be chosen based on an underlying coordinate system if available, or set to the direction of the first data point P_1 in P. We can sort the input by polar angles, in other words, we can order the points around x. In some cases, it is naturally presented this way. We reduce the θ^i_j to lie in the range $[0, 2\pi)$.

Definition 2.1.4. *The antipode of some point α on the unit circle is $\bar{\alpha} = (\alpha + \pi) \bmod 2\pi$.*

A key fact in computing CSD is that a triangle $\triangle abc$ does *not* contain x if and only if the corresponding polar angles of points a, b and c lie on a circular arc of less than π radians. This is illustrated in Fig. 2.1, and is equivalent to the following lemma, stated by Gil, Steiger and Wigderson [GSW92]:

Lemma 2.1.5. *Given points A, B, C on the unit circle $\mathcal{C}$ centred at x, let $\bar{A}$ be antipodal to A. Then $\triangle ABC$ contains x if and only if $\bar{A}$ is contained in the minor arc (i.e. of at most π radians) with endpoints B and C.*

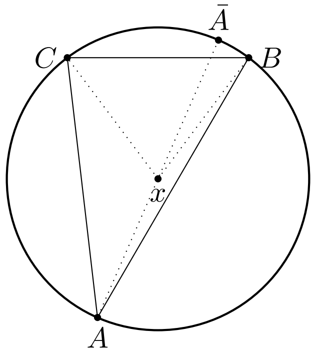


Figure 2.1: Antipode $\bar{A}$ falls in the minor arc between B and C and, therefore, the triangle $\triangle ABC$ contains x.

2.2 Outline of Strategy

Recall that we denote the ordinary and colourful simplicial depth by $D(x, P)$ and $\hat{D}(x, P)$ respectively. We start off by computing the simplicial depth of x with respect to all points in P, where no distinction between colours is made. From $D(x, P)$ we exclude the triangles that contain x and have two or three vertices of the same colour i, denoting this quantity by $D^i(x, P)$. When x and P are clear from the context, we will abbreviate these to D, $\hat{D}$ and D^i.

Since we can compute $D(x, P)$ efficiently using the algorithms mentioned in the introduction [GSW92], [KM90], [RR96], the challenge is to compute $D^i(x, P)$ for each $i = 1, 2, \ldots, k$. Then we conclude:

$$\hat{D}(x, P) = D(x, P) - \sum_{i=1}^{k} D^i(x, P) \ . \tag{2.1}$$

To compute D^i efficiently for each colour i, we walk around the unit circle tracking the minor arcs between pairs of points of colour i, and the number of antipodes between them. As shown in the Figure 2.2, an arc between A and B contains 3 antipodes. Note that we also include antipodes of the colour that matches the colour of the points that form the arc (red in this case). Having three points in the arc between A and B means that there are 3 triangles with base AB that contain x and have less than three distinctly coloured vertices. For each colour i, we consider all possible arcs formed by pairs of points of colour i and that are shorter than π. For simplicity, we will sometimes refer to these arcs as *intervals of colour i*.

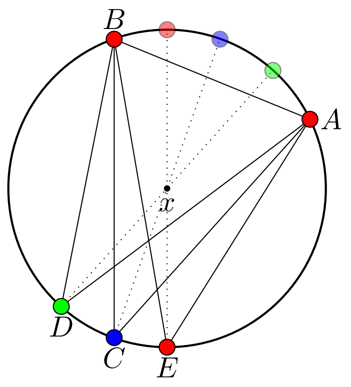


Figure 2.2: The antipodes of D, C, and E fall in the shorter arc between A and B, hence the triangles $\triangle ABE$, $\triangle ABC$, and $\triangle ABD$ contain x. All of these triangles have 2 or 3 red vertices, which is due to the fact that both A and B are red, and we are counting the antipodes of all possible colours between them.

Definition 2.2.1. *An* interval of colour i *is an arc shorter than π formed by a pair of points of colour i.*

However, tracking all such intervals would be expensive, so we came up with a formula that computes D^i in linear time in n_i. This builds on the approaches of Gil, Steiger, and Wigderson [GSW92], and Khuller and Mitchell [KM90] for monochrome depth.

Remark 2.2.2. *When computing D^i, we count antipodes of all k colours; the triangles with three vertices of colour i will be counted three times: $\triangle abc$, $\triangle bca$ and $\triangle cab$. Thus the quantity obtained by this count is in fact $D^i_* := D^i + 2\sum\limits_{j=1}^{k} D(x, P^j)$. We separately compute $\sum\limits_{j=1}^{k} D(x, P^j)$, allowing us to correct for the overcounting at the end.*

2.3 Data Structures and Preprocessing

We begin with the arrays θ^i of polar angles, which we sort. All elements in $\bigcup\limits_{i=1}^{k} \theta^i$ are distinct due to the general position requirement. By construction we have:

$$0 \le \theta^i_0 < \theta^i_1 < \ldots < \theta^i_{n_i-1} < 2\pi, \quad \text{for all } 1 \le i \le k \ . \tag{2.2}$$

Let $\bar{\theta}^i$ be the array of antipodes of θ^i for colour i, also sorted in ascending order. We generate $\bar{\theta}^i$ by finding the first $\theta^i_j \ge \pi$, moving the part of the array that begins with that element to the front, and hence the front of the original array to the back; π is subtracted from the elements moved to the front and added to those moved to the back. This takes linear time.

We merge all $\bar{\theta}^i$ into a common sorted array denoted by A which takes $O(n \log k)$. Now we have all antipodes ordered as if we were scanning them in counter-clockwise order around the circle $\mathcal{C}$ with origin x. Let us index the n elements of A starting from 0. Then, for each colour $i = 1, \ldots, k$, we merge A and θ^i into a sorted array A^i. Once again, this corresponds to a counter-clockwise ordering of data points of colour i plus the antipodes of all k colours around $\mathcal{C}$. Merging two sorted arrays of sizes n and n_i correspondingly takes linear time in their cumulative size: $O(n + n_i) = O(n)$.

While building A^i, we associate pointers from the elements of array θ^i to the corresponding position (index) in A^i. This is done by updating the pointers whenever a swap occurs during the process of merging the arrays. Denote the index of some θ^i_j in A^i by $p(\theta^i_j)$. Then the number of the antipodes that fall in the minor arc between two consecutive points θ^i_j and θ^i_{j+1} on $\mathcal{C}$ is

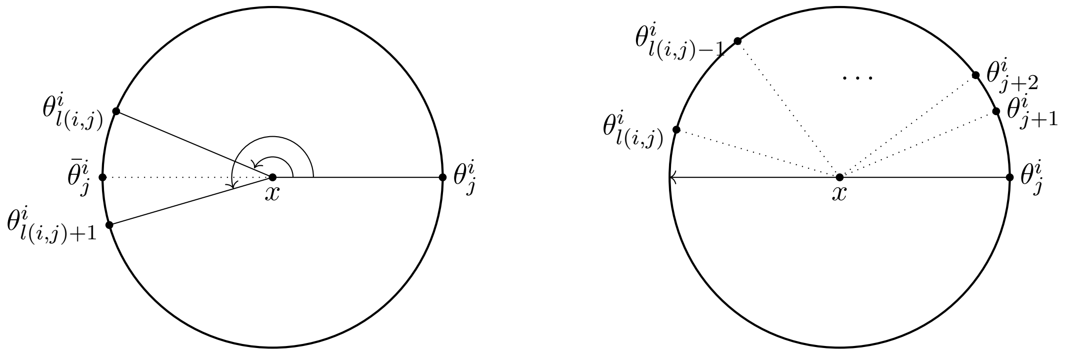
$$\begin{cases} p\left(\theta^i_{j+1}\right) - p\left(\theta^i_j\right) - 1, & \text{if } p\left(\theta^i_j\right) < p\left(\theta^i_{j+1}\right) \ , \\ n + n_i - p\left(\theta^i_j\right) + p\left(\theta^i_{j+1}\right) - 1, & \text{if } p\left(\theta^i_j\right) > p\left(\theta^i_{j+1}\right) \ . \end{cases} \tag{2.3}$$

Remark 2.3.1. *Note that $p\left(\theta^i_j\right)$ is never equal to $p\left(\theta^i_{j+1}\right)$.*

Now, for each point θ^i_j, we find the index $l(i,j)$ in the corresponding array θ^i such that $\angle\theta^i_j, x, \theta^i_{l(i,j)} < \pi$ and $\angle\theta^i_j, x, \theta^i_{l(i,j)+1} > \pi$ (Fig. 2.3a). Thus looking 180 degrees in the counterclockwise direction from θ^i_j on the circle $\mathcal{C}$, we will see a following sequence of points: $\theta^i_j, \theta^i_{j+1}, \ldots, \theta^i_{l(i,j)}$, where $\theta^i_{l(i,j)}$ is the last one (Fig. 2.3b).

Viewing the minor arc between two points as an interval (Def. 2.2.1), the intervals formed with the points from this sequence overlap and can be split into small disjoint intervals as follows:

$$\left[\theta^i_j, \theta^i_t\right) = \bigcup_{h=j+1}^{t} \left[\theta^i_{h-1}, \theta^i_h\right), \text{ where } t = j+1, \ldots, l(i,j) \ . \tag{2.4}$$



(a) Index $l(i,j)$ and index $(l(i,j)+1)$

(b) Arcs $\left[\theta^i_j, \theta^i_{j+1}\right), \left[\theta^i_j, \theta^i_{j+2}\right), \ldots, \left[\theta^i_j, \theta^i_{l(i,j)}\right)$ along the circle

Figure 2.3: Illustrations of the index $l(i,j)$.

2.4 Computing D^i_*

Let us the denote the count of the antipodes of all k colours within the minor arc between a and b by $c(a,b)$. Then Equation 2.3 expresses what we denote by $c(\theta^i_j, \theta^i_{j+1})$. Recall that D^i_* stands for D^i with overcounting the triangles that have all three vertices of the same colour. Using the newly introduced notation, D^i_* can be written as follows:

$$D^i_* = \sum_{j=0}^{n_i-1} \sum_{t=j+1}^{l(i,j)} c\left(\theta^i_j, \theta^i_t\right) \ . \tag{2.5}$$

Note that index t is taken modulo n_i. From (2.4) we have:

$$c\left(\theta^i_j, \theta^i_t\right) = \sum_{h=j+1}^{t} c\left(\theta^i_{h-1}, \theta^i_h\right), \quad \text{for } t = j+1, \ldots, l(i,j) \ . \tag{2.6}$$

Due to (2.5) and (2.6), we have:

$$D^i_* = \sum_{j=0}^{n_i-1} \sum_{t=j+1}^{l(i,j)} \sum_{h=j+1}^{t} c\left(\theta^i_{h-1}, \theta^i_h\right) \ . \tag{2.7}$$

Let us store the counts of antipodes between two consecutive points of colour i in the array C^i, so that $C^i_h = c\left(\theta^i_{h-1}, \theta^i_h\right)$, $|C^i| = n_i$. Then (2.7) can be rewritten as:

$$D^i_* = \sum_{j=0}^{n_i-1} \sum_{t=j+1}^{l(i,j)} \sum_{h=j+1}^{t} C^i_h \ . \tag{2.8}$$

We create an array of prefix sums: S^i, where $S^i_t = \sum\limits_{h=0}^{t} C^i_h$, $|S^i| = n_i$. This array can be filled in $O(n_i)$ time and proves to be very useful when we need to calculate a sum of the elements of C^i between two certain indices. In fact, this sum can be obtained in constant time using the elements of array S^i:

$$\sum_{h=j+1}^{t} C^i_h = \begin{cases} S^i_t - S^i_j, & \text{if } t \geq j+1, j \neq n_i - 1 \ , \\ S^i_{n_i-1} + S^i_t - S^i_j, & \text{if } t < j+1, j \neq n_i - 1 \ , \\ S^i_t, & \text{if } j = n_i - 1 \ . \end{cases} \tag{2.9}$$

We process the case $j = n_i - 1$ separately as follows:

$$D^i_* = \sum_{j=0}^{n_i-2} \sum_{t=j+1}^{l(i,j)} \left((S^i_t - S^i_j) + \begin{cases} 0, & \text{if } t \geq j+1 \\ S^i_{n_i-1}, & \text{if } t < j+1 \end{cases} \right) + \sum_{t=0}^{l(i,n_i-1)} S^i_t \ . \tag{2.10}$$

Noting that S^i_j is not dependent on t, we get:

$$\begin{aligned} D^i_* = &\sum_{j=0}^{n_i-2} \left(\sum_{t=j+1}^{l(i,j)} S^i_t - ((l(i,j) - j) \bmod n_i) \cdot S^i_j \right) + \sum_{t=0}^{l(i,n_i-1)} S^i_t \\ &+ \sum_{j=0}^{n_i-2} \begin{cases} \sum\limits_{t=j+1}^{l(i,j)} S^i_{n_i-1}, & \text{when } t < j+1 \ , \\ 0, & \text{otherwise} \ . \end{cases} \end{aligned} \tag{2.11}$$

Note that the index t runs from $j+1$ to $l(i,j)$. So $t < j+1$ in (2.10) and (2.11) only occurs where $j+1 > l(i,j)$ and we have wrapped around the array. In other words, $t < j+1$ is equivalent to $j+1 > l(i,j)$ and $t = 0, \ldots, l(i,j)$. Then:

$$\begin{aligned} D^i_* = &\sum_{j=0}^{n_i-2} \left(\sum_{t=j+1}^{l(i,j)} S^i_t - ((l(i,j) - j) \bmod n_i) \cdot S^i_j \right) + \sum_{t=0}^{l(i,n_i-1)} S^i_t \\ &+ \sum_{j=0}^{n_i-2} \begin{cases} \sum\limits_{t=0}^{l(i,j)} S^i_{n_i-1}, & \text{when } l(i,j) < j+1 \ , \\ 0, & \text{otherwise} \ . \end{cases} \end{aligned} \tag{2.12}$$

Following the previous logic, we create another array of prefix sums T^i, where $T^i_j = \sum\limits_{t=0}^{j} S^i_t$, $|T^i| = n_i$. This array is used to store the sum of elements of S^i between two certain indices in $O(1)$ time. We write:

$$\sum_{t=j+1}^{l(i,j)} S^i_t = \begin{cases} T^i_{l(i,j)} - T^i_j, & \text{if } l(i,j) \geq j+1 \ , \\ T^i_{n_i-1} + T^i_{l(i,j)} - T^i_j, & \text{if } l(i,j) < j+1 \ . \end{cases} \tag{2.13}$$

And

$$\sum_{t=0}^{l(i,n_i-1)} S^i_t = T^i_{l(i,n_i-1)} \ . \tag{2.14}$$

Combining (2.12) with (2.13) and (2.14), we have:

$$\begin{aligned} D^i_* = &\sum_{j=0}^{n_i-2} \left(T^i_{l(i,j)} - T^i_j - ((l(i,j) - j) \bmod n_i) \cdot S^i_j \right) + T^i_{l(i,n_i-1)} \\ &+ \sum_{j=0}^{n_i-2} \begin{cases} T^i_{n_i-1} + ((l(i,j)+1) \bmod n_i) \cdot S^i_{n_i-1}, & \text{when } l(i,j) < j+1 \ , \\ 0, & \text{otherwise} \ . \end{cases} \end{aligned} \tag{2.15}$$

After all this preprocessing, computing D^i_* is quite straightforward, and can be done in $O(n_i)$ time. Then, overall, computing the D^i_* for all $i = 1, \dots, k$ takes $O(\sum\limits_{i=1}^{k} n_i = n)$ time.

Following Remark 2.2.2, we finish by computing $D^i = D^i_* - 2 \cdot \sum\limits_{j=1}^{k} D(x, P^j)$, where $D(x, P^j)$ for each j is computed using one of the known algorithms from either [GSW92], [KM90], or [RR96] that run in time $O(n_i)$ on sorted data.

2.5 Algorithm and Analysis

Algorithm 1 computes the colourful simplicial depth of a point x with respect to a data set P. First, we find the polar angles corresponding to the data points in P and a fixed ray from x, and then their antipodes. This takes $O(n)$ in total. Second, we sort the arrays of polar angles θ^i while permuting their corresponding antipodal elements in $\bar{\theta}^i$, which is $O\left(\sum\limits_{i=1}^{k} n_i \log n_i\right)$. Third, we need to rotate $\bar{\theta}^i$, so that they are in an ascending order. This takes $O(n)$ time. Note then after this operation the correspondence between the elements in θ^i and $\bar{\theta}^i$ will be lost, but having them all sorted in an ascending order is exactly what we need for the purpose of finding the colourful simplicial depth.

For each $i = 1, \dots, k$, we compute the number of triangles with all three vertices of colour i that contain x, i.e. $D(x, P^i)$, using one of the implementations given in [GSW92], [KM90], or [RR96]. Without loss of generality, we decided to use the one provided in [RR96].

Algorithm 1 CSD(x, P)

```
Input: $x, P = (P^1, \ldots, P^k)$.   Output: $\hat{D}(x, P)$.
Sum1 $\leftarrow$ 0, Sum2 $\leftarrow$ 0;
for i $\leftarrow$ 1, k do
    for j $\leftarrow$ 0, $n_i - 1$ do
        $\theta^i_j \leftarrow$ polar angle of $(P^i_j - x) \bmod 2\pi$;
        $\bar{\theta}^i_j \leftarrow (\theta^i_j + \pi) \bmod 2\pi$;
    end for
    Sort($\theta^i$);                                          ▷ while permuting $\bar{\theta}^i$
    Restore the order in $\bar{\theta}^i$;
    Sum1 $\leftarrow$ Sum1 + D(x, $\theta^i$);                 ▷ use the algorithm from [RR96]
end for
A $\leftarrow$ Merge($\bar{\theta}^1, \ldots, \bar{\theta}^k$);       ▷ A is sorted
D $\leftarrow$ D(x, A);                                        ▷ use the algorithm from [RR96]
for i $\leftarrow$ 1, k do
    $A^i \leftarrow$ Merge(A, $\theta^i$);                     ▷ update $p(\theta^i_j)$ the pointers of $\theta^i_j$,
    for j $\leftarrow$ 1, $n_i$ do                             ▷ $j = j \bmod n_i$
        if $p(\theta^i_{j-1}) < p(\theta^i_j)$ then
            $C_j \leftarrow p(\theta^i_j) - p(\theta^i_{j-1}) - 1$;   ▷ $C = C^i$ - array of antipodal counts
        else
            $C_j \leftarrow n + n_i - p(\theta^i_{j-1}) + p(\theta^i_j) - 1$;
        end if
    end for
    Find l(i, 0) using binary search in $\theta^i$;
    $S_0 \leftarrow C_0$; $T_0 \leftarrow S_0$;                ▷ $S = S^i, T = T^i$ - prefix sum arrays
    for j $\leftarrow$ 1, $n_i - 1$ do
        Find l(i, j);
        $S_j \leftarrow S_{j-1} + C_j$;
        $T_j \leftarrow T_{j-1} + S_j$;
    end for
    Sum2 $\leftarrow$ Sum2 + $D^i_*(x, P)$ obtained from the formula (2.15);
end for
return $\hat{D}(x, P) = D - (\text{Sum2} - 2 * \text{Sum1})$;    ▷ $\text{Sum1} = \sum_{i=1}^{k} D(x, P^i)$
```

However, instead of passing the data points P^i as input of the algorithm, we pass the sorted polar angles θ^i directly. This will allow for the algorithm to execute in $O(n_i)$, for each i, or $O(n)$ in total. Hence lines 2-10 of the Algorithm 1 take $O\left(\sum_{i=1}^{k} n_i + \sum_{i=1}^{k} n_i \log n_i\right) = O(n \log n)$ time to complete. This follows from the facts that $\sum_{i=1}^{k} n_i = n$ and $n \log n$ is convex.

To generate the sorted array A of antipodes, we merge the k single-coloured arrays using a heap of size k (following e.g. [CLR89]) in $O(n \log k)$ time. We need to compute the monochrome depth $D(x, P)$ of x with respect to all points in P, regardless of colour. For this we can use the sorted array of antipodes rather than sorting the original array. Thus we again use the linear time monochrome algorithm [RR96] with x and A as input. Note that working with the antipodes is equivalent due to the fact that the simplicial depth of x does not change if we rotate the system of data points around the centre x.

After that, we execute a loop of k iterations – one for each colour. It starts with merging two sorted arrays A and θ^i, which is linear in the size of arrays we are merging and takes $O\left(\sum_{i=1}^{k}(n+n_i)\right) = O(kn)$ in total. Recall that during the merge we store the indices of the θ^i in resulting A^i, which allows to fill each array C^i in linear time $O(n_i)$ using the Formula 2.3. As we already mentioned in Section 2.4, prefix sum arrays can be filled in the time linear in their size. The algorithm is very simple, and can be found in the lines 24-29 of the Algorithm 1. Since the arrays C^i, S^i, and T^i take $O(3n) = O(n)$ space together for all colours i, the time needed to fill them out is also $O(n)$.

Note that the indices $l(i,j)$ appear in sequence in the array θ^i. Therefore, it is more efficient to find the first one $l(i,0)$ for each colour i, using a binary search that takes $O(\log n_i)$, hence $O(k \log n)$ in total. And then find the rest of $l(i,j)$, $j = 1, \ldots, n_i - 1$, in $O(n_i)$ time for each i by scanning through the array starting from the element $\theta^i_{l(i,0)}$.

Computing the value of $D^i_*(x,P)$ or simply D^i_* takes $O(n)$ overall. The remaining operations take constant time to execute. Therefore, total running time of Algorithm 1 is $O(kn + n + n\log n + n\log k + k\log n) = O(n\log n + kn)$. The $n\log n$ term corresponds to the initial sorting of the data points, if they are presented in sorted order, the running time drops to $O(kn)$. This compares very well to the monochrome case, where the algorithm that finds the simplicial depth $D(x,P)$ runs in $O(n\log n)$ time, and if the input data is sorted, drops to $O(n)$.

As for space, arrays θ^i, $\bar{\theta}^i$ and A take $3n = O(n)$ space in total. Note that merging k sorted arrays into A can be done in place [GG10]. At each iteration i, we create A^i of size $O(n + n_i)$, and C^i, S^i, T^i of size $O(n_i)$ each. Fortunately, we only need these arrays within the i^{th} iteration, so we can reuse them in the end (line 31 of the Algorithm 1) and reuse the space freed. This is the reason we have omitted the superscripts of C^i, S^i, and T^i in the Algorithm 1). Furthermore, to store the indices $l(i,j)$, we need $O(n)$ space, which again can be reallocated when i changes. Thus the amount of space used by our algorithm is $O(n)$, which is exactly the same as in the monochrome case. We summarize with:

Theorem 2.5.1. *Given a set of data points P in general position in k different colours, $k \geq 3$, and a point x in general position with P, the colourful simplicial depth of x relative to P, $\hat{D}(x,P)$, can be found in $O(n\log n + nk)$ time and $O(n)$ space, where $|P| = n$. If the points in P are sorted around x, the colourful simplicial depth of x with respect to P can be computed in $O(nk)$.*

Remark 2.5.2. *In Chapter 3, we will want to compute the colourful simplicial depth of the data points themselves. This can be done by computing $\hat{D}(x, P \setminus \{x\})$ and counting colourful* *simplices (triangles) that have x as a vertex. This is exactly the number of pairs of points of distinct colours that also differ from the colour of x. Such quantity can be computed in linear time.*

For a point x of colour i', this aforementioned quantity can be expressed as $\sum_{\substack{i=1\\ i\neq i'}}^{k} n_i \cdot \sum_{\substack{j=i+1\\ j\neq i'}}^{k} n_j$. A naive evaluation takes $O(k^2)$ time. To simplify it, we can use a prefix sum array K. Let $K_i = \sum_{\substack{j=1\\ j\neq i'}}^{i} n_j$. Then $\sum_{\substack{j=i+1\\ j\neq i'}}^{k} n_j = K_k - K_i$. In conclusion, to compute the CSD of x of colour i', we run Algorithm 1, and then add $\sum_{\substack{i=1\\ i\neq i'}}^{k} n_i \cdot (K_k - K_i)$ to the obtained result. Array K takes $O(k)$ space and takes linear time to fill.

2.5.1 Implementation of the Algorithm

An implementation of this algorithm is available on-line [Zas16]. For this purpose we used Java and the package *jmathplot* [Ric16] to draw the points. A uniform generator of points in general position was written to test the program [Zas17b]. We have verified our implementation by comparing it to the brute force enumeration [Zas17a]. Our brute force implementation uses the cross product to check whether a point lies inside a triangle.

Instead of keeping track of the execution time, we count operations. Every time we perform a floating point operation, such as addition, subtraction, multiplication, or division, a count is increased by 1. This includes the calculations performed on array indices. Another count we keep is the number of comparisons performed in the Algorithm 1 in steps 11 and 14, where we merge the arrays. All calculations were performed on a MacBook Pro machine with 2.9 GHz Intel Core i5 Processor and 16 GB 1867 MHz DDR3 RAM.

A comparison of our method to the brute force enumeration for fixed n and varying k is displayed in Table 2.1, where the last column is the number of floating point operations followed by the number of comparisons in the parentheses. The results for k fixed and varying n can be found in Table 2.3.

In Figure A.4 green dots show the floating operation count and red ones show the number of comparisons carried out when merging plotted against the number of points in P, for a fixed $k = 3$. Whereas Figure A.5 shows the same counts, but against the number of colours for fixed $n = 1000$.

An example of a configuration in $k = 4$ colours and $n = 500$ points is shown in Figure A.1, where $\hat{D}((0,0),P) = 1,962,624$. We've set *min* and *max* values on x and y coordinates to -500 and 500 respectively. Another example is a configuration with $k = 10$ and $n = 6000$ $\hat{D}((0,0),P) = 6,480,643,500$ (Figure A.2).

k	**n**	**CSD**	**Operation Count**	
			Brute Force Algorithm	Our Algorithm
3	1000	9,276,741	1,477,639,471	102,095 (4,986)
6	1000	23,141,494	3,693,608,639	108,283 (10,163)
12	1000	31,855,526	5,078,991,246	120,436 (18,299)
24	1000	36,519,662	5,837,724,107	145,673 (32,247)
48	1000	38,827,340	6,214,214,800	200,371 (56,355)

Table 2.1: Comparison of Operation Count for Brute Force Algorithm vs Our Algorithm for fixed n.

k	**n**	**CSD**	**Operation Count**	
			Brute Force Algorithm	Our Algorithm
3	10	6	1,471	856 (36)
3	100	9,161	1,479,255	10,235 (482)
3	1000	9,219,163	1,477,116,053	102,224 (4,986)
3	10,000	47,909,756,342	1,476,881,399,348	1,020,891 (49,986)

Table 2.2: Comparison of Operation Count for Brute Force Algorithm vs Our Algorithm for fixed k.

We've tested and confirmed that the algorithm works well with points not in general position on small data sets. Figure A.3 displays an example where more than two points lie on the same line. Our algorithm computed $\hat{D}((0,0),P)$ to be 254.

k	**n**	**CSD**	**Operation Count**	
			Brute Force Algorithm	Our Algorithm
6	10	6	2,390	963 (56)
6	100	22,488	3,679,709	10,935 (982)
6	1000	23,173,687	3,693,275,030	108,134 (10,163)
6	10,000	48,918,787,471	3,692,498,319,339	1,080,524 (10,2261)

Table 2.3: Comparison of Operation Count for Brute Force Algorithm vs Our Algorithm for fixed k.

Chapter 3

Computing Colourful Simplicial Medians

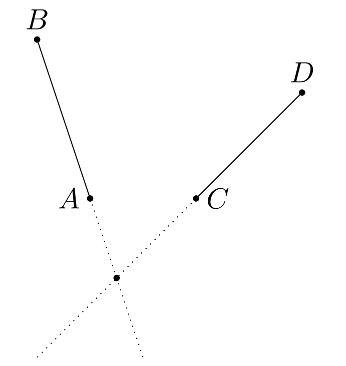
Now that we have an efficient algorithm to compute the colourful simplicial depth or CSD of a point $x \in \mathbb{R}^2$ relative to a set P, we can proceed to finding a colourful simplicial median, that is a point of maximum CSD that lies in the convex hull of P. The running time of our algorithm is $O(n^4)$, which matches the monochrome case, being independent of the number of colours. This is arguably as good as should be expected, following the observation of Lemma 3.1.1 in Section 3.1 that shows that there are in some sense $\Theta(n^4)$ candidate points for the location of the colourful median.

If we wanted to find the *in-sample colourful simplicial median*, it would suffice to compute $\hat{D}(p, P)$ for all $p \in P$. That takes $O(n^2 \log n + kn^2)$ time, or $O(kn^2)$ if we use the algorithm from [LC85] to presort the data points in P.

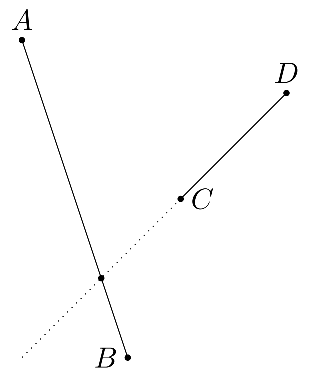
Our algorithm is founded on the topological sweep for a complete graph as presented by Rafalin and Souvaine. This removes some of the overhead of the general topological sweep framework [EG89], notably in avoiding the use of *phantom vertices*. These are intersections of an extension of a line segment with either another segment or with an extension of another segment, see Figure 3.1, and were used in a preliminary version of this work [ZS16]. Here we are going to use the topological sweep for a complete graph [RS08] which simplifies the process significantly.

3.1 Preliminaries

Consider a family of sets P^1, P^2, ... , $P^k \in \mathbb{R}^2$, $k \geq 3$, where each P^i consists of the points of some particular colour i. Define $n_i = |P^i|$, for $i = 1, \dots, k$. Let P be the union of all colour sets: $P = \bigcup_{i=1}^{k} P^i$, points in P in general position. Recall that we denote the CSD of a point $x \in \mathbb{R}^2$ relative to P by $\hat{D}(x, P)$. Our objective is to find a point x inside the convex hull of P, denoted $conv(P)$, maximizing $\hat{D}(x, P)$. Call the depth of such a point $\hat{\mu}(P)$.



(a) Extensions of two segments intersect.



(b) An extension of segment (D, C) intersects the segment (A, B).

Figure 3.1: Examples of phantom vertices.

Let S be the set of line segments formed by all possible pairs of points (A, B), where $A \in P^i$, $B \in P^j$, $i < j$. We will refer to these as *colourful segments*. We call the intersection points of the segments in S *vertices* and denote their set by V. The following lemma (from [ALST03]) is here adapted to a colourful setting:

Lemma 3.1.1. *To find a point with maximum colourful simplicial depth it suffices to consider the intersection points of the colourful segments in S.*

Proof. The segments of S partition $conv(P)$ into cells[1] of dimension 2, 1, 0 of constant colourful simplicial depth [DHST06]. Cells are relatively open and simply connected. Consider a 2-dimensional cell. Let p be a point in the interior of this cell, q a point on the interior of an edge and v a vertex, so that q and v belong to the same line segment (Fig. 3.2). Then the following inequality holds: $\hat{D}(p, P) \leq \hat{D}(q, P) \leq \hat{D}(v, P)$, since any colourful simplices containing p also contain q, and any containing q also contain v. This proves the Lemma. $\square$

Observe that drawing the colourful segments is equivalent to generating a rectilinear drawing of the complete graph K_n with a few edges removed (the monochrome ones). Thus, unless the points are concentrated in a single colour class, the Crossing Lemma (see e.g. [PRTT06]) shows that we will have $N = \Theta(n^4)$ vertices. More precisely, we have a rectilinear drawing of a complete k-partite graph; bounds for this are considered some graphs from this family by Gethner et al. [GHL$^+$17] and references therein. To find all intersection points of the segments in S, the *topological sweep* [RS08] is applied. The algorithm we present in this thesis is optimal, because we check all the data points and all the intersection points. There is $O(n^2)$ segments and, hence, $O(n^4)$ intersection points.

[1]Unlike the monochrome case, here some cells may not be convex, and some points of $conv(P)$ may fall outside any cell.

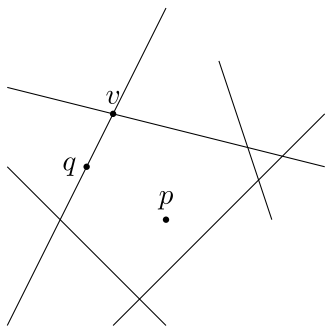


Figure 3.2: An example of a cell.

Computing the CSD of each of these $O(n^4)$ vertices gives an $O(n^5 \log n + kn^5)$ algorithm for finding a colourful simplicial median. To improve this, we follow Aloupis et al. [ALST03], and compute the depth of most vertices based on the values of their neighbours and information about the half-spaces of local segments. In [ALST03], the simplicial depth of a vertex v is computed based on the already known depth of an adjacent vertex p and on the number of triangles that we left and entered correspondingly when moving from p to v. By the number of triangles that we *leave* we mean the triangles that contained p, but do not contain v. Then the number of triangles that we *enter* consists of the triangles that contain v but not p. These numbers can be obtained by calculating the number of points on either side of the segments formed by the pairs of data points. This is due to the fact that a segment has two endpoints that form a triangle with every point that is not an endpoint of this segment. Then every point on the interior of this segment is contained in such triangle, because we examine closed containment (Fig. 3.3). Suppose that a vertex p lies on the segment (A, B) as shown in the Figure 3.3, and a vertex v lies on a completely different segment (X, Y). Then, when we are moving from p to v, we leave $\triangle ABC$, $\triangle ABD$, $\triangle ABE$, i.e. the triangles formed with the points that lie on one side of (A, B) from v. We will explain in more detail and provide a full example further, in Section 3.3.1, where we talk about the colourful case.

3.2 Notation

Let $col(A)$ denote the colour of a point A. We store the segments in S as pairs of points: $s = (A, B)$, $col(A) < col(B)$. It is helpful to view each segment as directed, i.e. a vector, with A as the tail and B as the head. Each segment s extends to a directed line dividing $\mathbb{R}^2$ into two open half-spaces: s^+ and s^-, where s^+ lies to the right of the vector s, and s^- to the left (Figure 3.4).

Denote the number of points in s^+ that have colours different from the endpoints of s by $r(s)$, and those in s^- by $l(s)$. Let $r^i(s)$ and $l^i(s)$ be the number of points of a colour i in s^+ and s^- respectively. Let $\bar{r}^i(s)$ and $\bar{l}^i(s)$ be the number of points of all k colours except

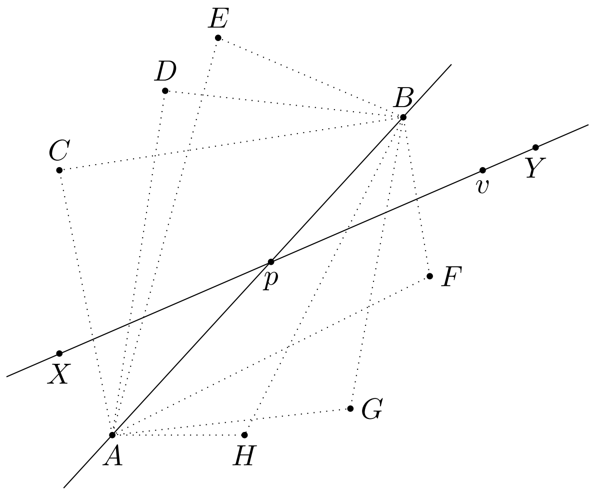


Figure 3.3: Every point on the interior of the segment (A, B), for example p, is contained in the triangles $\triangle ABC$, $\triangle ABD$, $\triangle ABE$, as well as $\triangle ABF$, $\triangle ABG$, $\triangle ABH$.

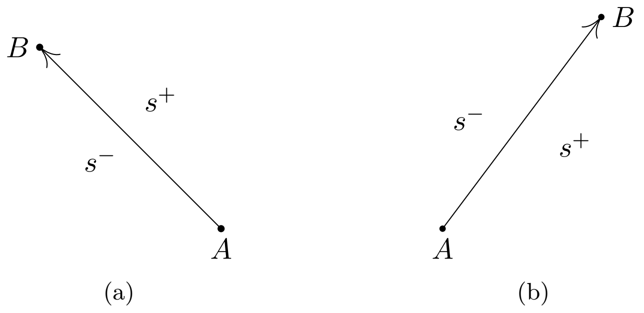


Figure 3.4: s^+ and s^- of the segment $s = (A, B)$.

for the colour i in s^+ and s^- respectively. For example, given a segment $s = (A, B)$, the aforementioned quantities should be as follows:

$$\bar{r}^{col(A)}(s) = \sum_{\substack{i=1,\\ i\neq col(A)}}^{k} r^i(s) \ , \qquad \bar{l}^{col(A)}(s) = \sum_{\substack{i=1,\\ i\neq col(A)}}^{k} l^i(s) \ . \tag{3.1}$$

Then

$$r(s) = \bar{r}^{col(A)}(s) - r^{col(B)}(s) \ , \qquad l(s) = \bar{l}^{col(A)}(s) - l^{col(B)}(s) \ . \tag{3.2}$$

The values needed to compute $r(s)$ and $l(s)$ for each segment $s \in S$ as shown in Eq. 3.2, can be obtained as byproducts of an algorithm that computes half-space depth.

The *half-space depth* $HSD(x, P)$ of a point x relative to data set P is the smallest number of data points from P in a half-plane through the point x [Tuk75]. An efficient algorithm to compute the half-space depth of a point $x \in \mathbb{R}^2$ relative to a set P is provided in [RR96]. It runs in $O(|P|)$ time when P is sorted around x, which we are going to use to our advantage. This algorithm draws a line between x and every $P_i \in P$, then calculates the number of points in P that lie strictly to the left of each of these lines. However, when calculating the number of points that lie to the right, p is included into the count. We correct for this by subtracting 1 from those counts. Algorithm 4 manipulates different combinations of the colour sets, usually already sorted, in order to obtain the necessary values $r(s)$ and $l(s)$ for all $s \in S$ and store them in a map M.

We use the algorithm in [LC85] that, for each $P_i \in P$, sorts $P \setminus \{P_i\}$ around P_i in $\Theta(|P|^2)$ time. In particular, it assigns to every point $P_i \in P$ a list of indices that determine the order of points $P \setminus \{P_i\}$ in the clockwise ordering around P_i. Denote this by $List(P_i)$. Since the HSD algorithm needs the data points to be sorted in the counter-clockwise order around the centre, we read those lists in reverse order.

The next section gives extensive background on the topological sweep and how we use $r(s)$ and $l(s)$.

3.3 Computing the median

To compute the CSD of all vertices, we carry out a topological sweep described in [RS08]. It fits our purpose very well, because the set of colourful segments S is a complete graph with monochrome edges removed.

Here a topological line is swept through an arrangement of line segments, where the discovery of each new vertex is called an *elementary step*. Each vertex is processed in amortized constant time. The topological line can be thought of as a moving wall that separates the arrangement into two regions: the already discovered intersection points and the ones yet to be registered. At each elementary step we are computing the CSD of a new vertex. Thus the moving wall separates the vertices with known CSD from the vetices whose colourful simplicial depth has yet to be computed.

Combinatorially, the topological line is a *cut* in the sense of graph theory. It intersects every edge of the graph at most once. An edge intersected by a cut is called *active*. An *active segment* is the specific segment of the active edge that is intersected by the cut, it extends to the right until it is intersected by another active segment or until it meets a data point. Figure 3.5 demonstrates an example of a topological line with bold lines being active segments. An active segment can contain several vertices formed by intersections with inactive segments. We start the sweep with the *leftmost cut* which intersects no edges of the graph. This is pushed to the right until it becomes the *rightmost cut*, in the process performing elementary steps. An intersection point is called a *ready intersection* if all incoming edges have active segments that are consecutive in the cut. When no intersection points are ready, a vertex of the graph is processed by the topological sweep. In that case we execute the CSD algorithm with an input that consists of this current data point and the rest of the points in P.

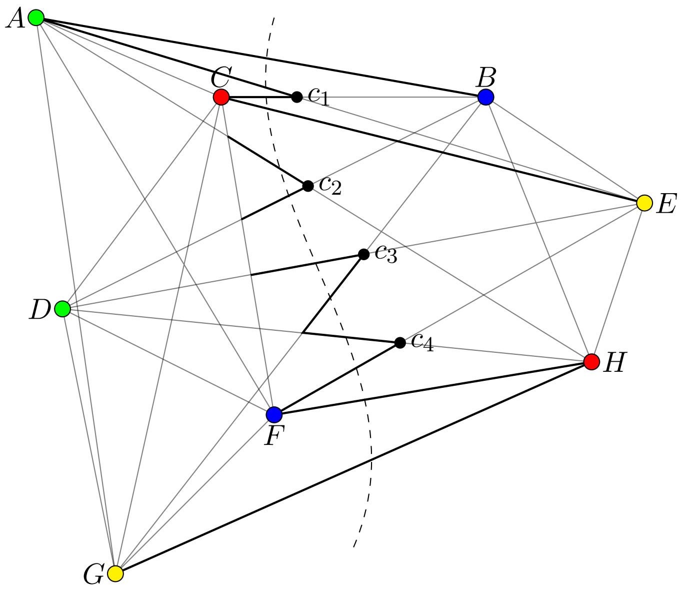


Figure 3.5: A topological line crossing the configuration.

We denote the set of ready intersections by I. For example, in Figure 3.5 current state of I is $\{c_1, c_2, c_3, c_4, H\}$. As we start with the leftmost cut, the topological line is completely outside the convex hull of P. To enter it, the topological line needs to cross one of the points on the boundary of $conv(P)$. Therefore, we initialize I with the points of P that are vertices of the boundary of $conv(P)$. At every iteration, we take a vertex from I, following the rules described in [RS08]. At the end of every iteration we update I by deleting the vertex we just crossed and by adding the vertices that have just become ready intersections to it.

For each segment $s \in S$ we store the last processed vertex that lies on this segment and denote it by $ver(s)$, along with its CSD. We also store the crossing segment that defines $ver(s)$ on s, and denote it by $cross(ver(s))$. Before starting the topological sweep, for each $s \in S$ we assign $ver(s) = \emptyset$. After completing an elementary step where we crossed a vertex v that lies at the intersection of s_i and s_j, we assign $ver(s_i) \leftarrow v$, $ver(s_j) \leftarrow v$, $cross(ver(s_i)) \leftarrow s_j$, $cross(ver(s_j)) \leftarrow s_i$. We need this so that we can compute the CSD of a newly discovered vertex using the CSD of an adjacent one. However, when the elementary step is performed on a data point p, we assign $ver(s) = \emptyset$ for each incoming segment s, i.e. for each active segment that ends in p. It is safe to do so, because the algorithm [RS08] deletes the edges terminating at a given data point from the list of active edges, so we won't hit them again. We also do nothing for the outgoing segments from p, that is the ones that weren't registered yet.

We now explain how we process a vertex v at an elementary step when we have an adjacent vertex with already computed CSD. Assume v is at the intersection of $s_i = (D, E)$ and $s_k = (B, G)$, see Figure 3.6. If both $ver(s_i)$ and $ver(s_k)$ are non-empty, we are free to choose whichever one of them to compute the CSD of v. Without loss of generality we take $ver(s_i) = p$, where $cross(ver(s_i)) = s_j$. We view this elementary step as moving along the segment s_i from its intersection point with s_j to the one with s_k. Each intersecting segment forms a triangle with every point strictly to one side. Thus when we leave segment $s_j = (C, F)$ behind, we exit as many colourful triangles that contain p as there are points on the other side of s_j of colours different from $col(C)$ and $col(F)$. In our example, we leave $\triangle CFA$, $\triangle CFD$, and $\triangle CFG$, because the points A, D, and G have colours that are different from the colours of C and F.The above mentioned triangles contain p, but not v, so we can subtract them from the CSD of p.

When we encounter segment $s_k = (B, G)$, we enter the colourful triangles that contain v formed by s_k and each point of a colour different from $col(G)$ and $col(B)$ on the other side of s_k. In our example, there is only one such triangle $\triangle BGH$, because the colour of point H is different from the colours of B and G.This triangle contains v, but not p, hence we can add it to the CSD of p after we have subtracted the 3 triangles we left behind. Here $r(s_j = (C, F)) = 3$, $l(s_k = (B, G)) = 1$. Then $\hat{D}(v) = \hat{D}(p) - r(s_j) + l(s_k) = 14 - 3 + 1 = 12$.

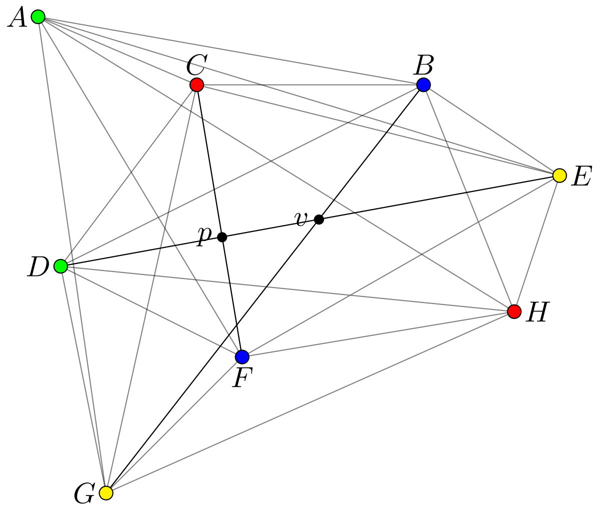


Figure 3.6: The CSD of p is 14, and the CSD of v is 12 since as we move along DE from CF to BG, we leave triangles $\triangle CFA$, $\triangle CFD$, $\triangle CFG$ containing p and enter only $\triangle BGH$ that contains v.

This is the cornerstone of our algorithm. The trick is to know which side of a segment to consider, left or right, $l(s)$ or $r(s)$. But it turns out it's quite easy to resolve. Let us denote the head and the tail of a segment s by $head(s)$ and $tail(s)$ respectively, and x and y coordinates of a point A by $A.x$ and $A.y$ respectively. Now, to compute the CSD of v knowing the CSD of p, we execute Subroutine 2. We pass it the vertex p along with its CSD which we then use to compute the CSD of v. Other arguments are two segments that define the intersection points p and v with the segment along which we are moving. Let us use Figure 3.7 as an example. Suppose we are moving from p to v along the segment

s_k. Then the input to Subroutine 2 should be: $\left(\hat{D}(p), p, v, s_i, s_j\right)$. The output is simply $\hat{D}(v)$. We use the cross product to determine which side of a vector a certain point lies in. Subroutine 2 runs in constant time.

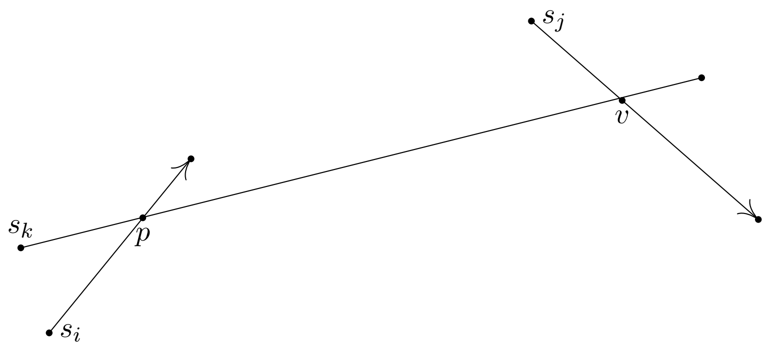


Figure 3.7: Two adjacent vertices p and v and their corresponding line segments.

```
Subroutine 2 Computing $\hat{D}(v)$ from $\hat{D}(p)$
Input: $\hat{D}(p), p, v, s_b, s_a$. Output: $\hat{D}(v)$.
 1: if Subroutine 3 $(v, s_b) < 0$ then
 2:     $\hat{D}(v) \leftarrow \hat{D}(p) - r(s_b)$;
 3: else
 4:     $\hat{D}(v) \leftarrow \hat{D}(p) - l(s_b)$;
 5: end if
 6: if Subroutine 3 $(p, s_a) < 0$ then
 7:     $\hat{D}(v) \leftarrow \hat{D}(v) + r(s_a)$;
 8: else
 9:     $\hat{D}(v) \leftarrow \hat{D}(v) + l(s_a)$;
10: end if
11: return $\hat{D}(v)$;
```

```
Subroutine 3 Cross-product
Input: $p, s = (A, B)$. Output: $\overrightarrow{AB} \times \overrightarrow{Ap}$.
 1: return $(p.x - head(s).x)(tail(s).y - head(s).y)$
 2:            $-(p.y - head(s).y)(tail(s).x - head(s).x)$;
```

When both $ver(s_i) = \emptyset$, $ver(s_k) = \emptyset$, i.e. vertex v is the first vertex to be discovered for both segments and the only vertices with known CSD adjacent to it are the actual data points, we execute $CSD(v, P)$ to find the depth. In the example displayed in Figure 3.8, any one of the vertices c_1 through c_8 may require computing the CSD from scratch. However, we will only see at most one of these per segment. For example, once $\hat{D}(c_1)$ is computed, we will not run the CSD algorithm for the vertices c_8 and c_2. On the other hand, c_5 cannot cause such a situation either, because the topological sweep moves from left to right. So would never approach c_5 from H or E.

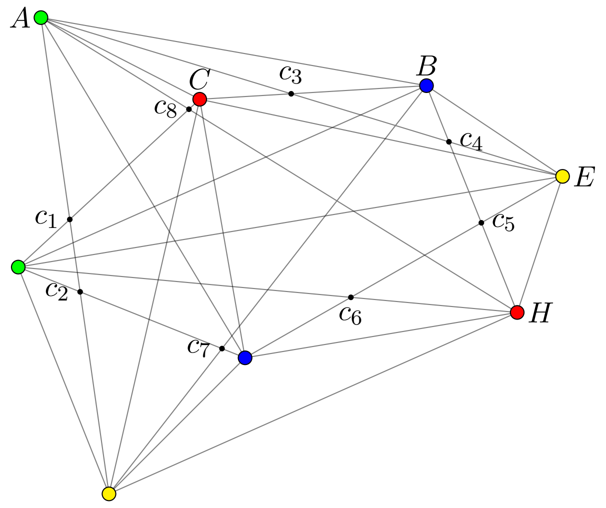


Figure 3.8: Points that may require the execution of the colourful simplicial depth algorithm.

It is tempting to assign the data points themselves as last processed vertices, i.e. allow $ver(s)$ to be a data point, and avoid running the CSD algorithm on the points that are not in P. However, consider the situation in Figure 3.8 after discovering vertex c_3. It doesn't matter whether we came to it from C or from A, because both are data points. If we assign $ver(s = (C, B)) \leftarrow C$, then we can use $\hat{D}(C)$ to calculate the $\hat{D}(c_3)$. Due to this ambiguity, we choose to run the colourful simplicial depth algorithm for all vertices whose immediate neighbours with known CSD are all data points. We bound the number of cases when we have to call the colourful simplicial depth algorithm as $O(n^2)$, as we never compute the CSD of a vertex based on a movement from one segment to another. This adds $O(n^3 \log n + kn^3)$ to the running time, but doesn't exceed $O(n^4)$.

3.3.1 Computing $r(s)$ and $l(s)$

Recall that $r(s)$ is the number of points strictly to the right of s whose colours differ from the colours of the endpoints of s. And $l(s)$ is the number of points on the left respectively. We compute these values with the help of *half-space depth* algorithm or HSD. The pseudocode is available in Algorithm 4. First, we presort the data using the algorithm in [LC85] that runs in $O(n^2)$ time. This gives us an advantage of reducing the running time of the HSD algorithm from $O(n \log n)$ to $O(n)$. For each point $P_i \in P$, the algorithm in [LC85] creates a list of indices of $P \setminus P_i$ ordered around P_i in the clockwise order. Each list we read in the reverse order and form k arrays from it: $\bar{\theta}^{col(P_i)}$ – polar angles of the points corresponding to the ordering in $List(P_i)$ with the points of $col(P_i)$ excluded, and for each colour i' other than $col(P_i)$ we form an array $\theta^{i'}$ – polar angles of the points corresponding to the ordering in $List(P_i)$ only of colour i'. Together they take $O(n)$ and we delete them at the end of every iteration. Filling them takes linear time – just going through $List(P_i)$ once.

Algorithm 4 Preprocessing: Computing $r(s), l(s)$

```
Input: $P, P^1, \ldots, P^k$. Output: $M$.
1: Construct List($P_i$) lists of points sorted around $P_i$ for each
   $i = 0, \ldots, n-1$ [LC85];
2: for $i \leftarrow 0, n-1$ do
3:    while List($P_i$) $\neq \emptyset$ do
4:       $t \leftarrow$ pop(List($P_i$));                    ▷ recall $t$ is an index
5:       if $col(P_t) \neq col(P_i)$ then
6:          $\bar{\theta}^{col(P_i)} \leftarrow$ polar angle of $P_t$;
7:          if $col(P_t) > col(P_i)$ then
8:             $\theta^{col(P_t)} \leftarrow$ polar angle of $P_t$;
9:          end if
10:      end if
11:   end while
12:   Compute $\bar{r}^{col(P_i)}(s), \bar{l}^{col(P_i)}(s)$ while running HSD($P_i, \bar{\theta}^{col(P_i)}$) [RR96],
   for every line drawn between $P_i$ and $p \in \bar{\theta}^{col(P_i)}$ create a segment
   $s = (P_i, p)$, assign $ver(s) \leftarrow \emptyset$, insert it into a map M as
   $s \rightarrow \left(\bar{r}^{col(P_i)}(s), \bar{l}^{col(P_i)}(s), ver(s) = \emptyset\right)$;
13:   for $i' \leftarrow 1, k$ do
14:      if $i' > col(P_i)$ then
15:         Compute $r^{i'}(s), l^{i'}(s)$ during the execution of HSD($P_i, \theta^{i'}$) [RR96]
   and for each segment $s = (P_i, p)$, $p \in \theta^{i'}$, override the value of $s$ in the
   map $M$: $s \rightarrow (r(s), l(s), ver(s) = \emptyset)$ by Eq. 3.2;
16:         delete $\theta^{i'}$;
17:      end if
18:   end for
19:   delete $\bar{\theta}^{col(P_i)}$;
20: end for
21: return M;
```

Then we execute $HSD(P_i, \bar{\theta}^{col(P_i)})$. A line is drawn from P_i to every point in $\bar{\theta}^{col(P_i)}$ and the number of points on either side of each line is computed. Since we have points of all k colours except for $col(P_i)$, the counts obtained correspond to the values $\bar{r}^{col(P_i)}$ and $\bar{l}^{col(P_i)}$ we defined in Section 3.2. Note that in Algorithm 4 we find and store all the colourful segments in a map M, where initially the values of each key are $\bar{r}^{col(P_i)}(s)$, $\bar{l}^{col(P_i)}(s)$, and $ver(s) = \emptyset$.

Afterwards, for each colour i' such that $i' > col(P_i)$ (to avoid repeated segments), we run $HSD(P_i, \theta^{i'})$. Overall this will draw a line from P_i to the same set of points. However, for a colour i' a line will be drawn from P_i to the points of this colour, and the points of colour i' on either side of this line will be counted. Previously we denoted such counts of points by $r^{i'}$ and $l^{i'}$. Now we have all the necessary counts to derive the $r(s)$ and $l(s)$. We do so during the execution of $HSD(P_i, \theta^{i'})$, where for every segments s that must already be in the map we update its values to be $r(s)$, $l(s)$, and $ver(s) = \emptyset$, using the formulae provided in 3.2. At the end of the Algorithm 4 we have a map M with all colourful segments as keys along with their corresponding values $r(s)$, $l(s)$, $ver(s)$.

We illustrate Algorithm 4 with an example. Consider Figure 3.9. We start with H as a point around which the rest of the data is ordered. Here $\bar{\theta}^{red} = \{E, B, A, D, F, G\}$, $\theta^{blue} = \{B, F\}$, $\theta^{green} = \{A, D\}$, $\theta^{yellow} = \{E, G\}$. First, we run $HSD(H, \bar{\theta}^{red})$. That draws all the lines displayed in Figure 3.9 and counts points on both sides of them. We get: $\bar{r}^{red}(H, E) = 0$ (recall this is the number of points of all colours except red on the right side of the segment (H, E)), $\bar{l}^{red}(H, E) = 5$, $\bar{r}^{red}(H, B) = 1$, $\bar{l}^{red}(H, B) = 4$, $\bar{r}^{red}(H, A) = 2$, $\bar{l}^{red}(H, A) = 3$, $\bar{r}^{red}(H, D) = 3$, $\bar{l}^{red}(H, D) = 2$, $\bar{r}^{red}(H, F) = 4$, $\bar{l}^{red}(H, F) = 1$, $\bar{r}^{red}(H, G) = 5$, and lastly $\bar{l}^{red}(H, G) = 0$. Then we execute $HSD(H, \theta^{blue})$ which draws lines from H to only blue points, as a result we obtain $r^{blue}(H, B) = 0$, $l^{blue}(H, B) = 1$, $r^{blue}(H, F) = 1$, $r^{blue}(H, F) = 0$. Then $r(H, B) = \bar{r}^{red}(H, B) - r^{blue}(H, B) = 1 - 0 = 1$, $l(H, B) = \bar{l}^{red}(H, B) - l^{blue}(H, B) = 4 - 1 = 3$, $r(H, F) = \bar{r}^{red}(H, F) - r^{blue}(H, F) = 4 - 1 = 3$, $l(H, F) = \bar{l}^{red}(H, F) - l^{blue}(H, F) = 1 - 0 = 1$. The computations above match the data displayed in the Figure 3.9: there is 1 point on the right of the segment (H, B) that is neither blue or red; there are 3 points on the left of (H, B) that are neither blue or red; there are 3 non-blue and non-red points on the right of (H, F); there is 1 yellow point on the left of (H, F). After that we also run $HSD(H, \theta^{green})$ and $HSD(H, \theta^{yellow})$ and perform similar deductions. Then we proceed to do the same with the next point around the circle which is E in this case, until we reach G, the last point in the ordering.

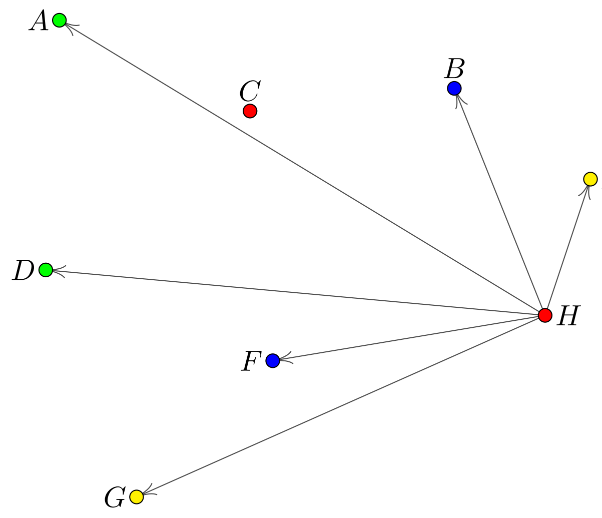


Figure 3.9: How the half-space depth algorithm works.

3.4 Running Time and Space Analysis

Algorithm 5 is the main algorithm that computes the colourful simplicial median. First, it obtains the map that contains half-space counts $r(s)$ and $l(s)$ for all $s \in S$ by running the Algorithm 4. The map we use in [Zas16] is a HashMap, which is a data structure that contains keys and their corresponding values. A value can be obtained from the map by its key using the method *get(key)* in O(1). Adding a value to the map is done via the method *put(key, value)* which also takes constant time. It takes $O(n^2)$ to generate the $List(P_i)$ for all $P_i \in P$ [LC85], then $O(n)$ time to create the auxiliary arrays. Running the HSD algorithm overall takes $O(n \cdot n + \sum_{i=1}^{k} n_i \cdot \sum_{i'=i+1}^{k} n_{i'}) \approx O(n^2)$. Hence the total running time of Algorithm 4 is $O(n^2)$. The map M together with $List(P_i)$ require $O(n^2)$ storage. After executing Algorithm 4, $ver(s) = \emptyset$ for all $s \in S$.

Algorithm 5 Computing $\hat{\mu}(P)$

```
Input: $P, P^1, \dots, P^k$. Output: $v, \hat{\mu}(P)$.
 1: $M \leftarrow$ Algorithm 4 $(P, P^1, \dots, P^k)$;
 2: $S \leftarrow$ M.keys sorted by slopes;
 3: max $\leftarrow 0$;
 4: for $i \leftarrow 0, n-1$ do
 5:     $\theta^i$ = polar angles of List$(P_i)$;
 6: end for
 7: $I \leftarrow$ points of P that are vertices of the boundary of conv(P);
 8: while $I \neq \emptyset$ do                       ▷ Start of the topological sweep.
 9:     $v \leftarrow$ pop(I);    ▷ Only take $v$ from $I$ if it satisfies the rules in [RS08]
10:     if $v \in P$ then                                          ▷ Let it be $P_i$
11:         $\hat{D}(v) = CSD(v, \theta^i)$;
12:     end if
13:     if $ver(s_i) = \emptyset$ & $ver(s_k) = \emptyset$ then    ▷ v lies at the intersection of $s_i$
                                                   and $s_k$
14:         $\hat{D}(v) = CSD(v, P)$;
15:     else if $ver(s_i) \neq \emptyset$ then
16:         $\hat{D}(v) \leftarrow$ Subroutine 2 $(\hat{D}(p), p, v, s_j, s_k)$;   ▷ $p = ver(s_i), s_j = cross(ver(s_i))$
17:     else
18:         $\hat{D}(v) \leftarrow$ Subroutine 2 $(\hat{D}(p), p, v, s_j, s_i)$;   ▷ $p = ver(s_k), s_j = cross(ver(s_k))$
19:     end if
20:     if $\hat{D}(v) >$ max then
21:         max $\leftarrow \hat{D}(v)$;
22:         median $\leftarrow v$;
23:     end if
24:     $ver(s_i) \leftarrow v$, $ver(s_k) \leftarrow v$, $cross(ver(s_i)) \leftarrow s_k$, $cross(ver(s_k)) \leftarrow s_i$;
25:     Push the intersection points of active edges into $I$;
26: end while                                    ▷ End of the topological sweep.
27: return (median, max).
```

In step 2 of Algorithm 5 we take the keys of M and sort them by the slopes, which yields the set S. It takes $O(n^2 \log n)$ time. We assume non-degeneracy and no vertical lines (these can use some special handling, see [EM88]). Since we have already presorted the points in

P, we create arrays of polar angles θ^i for each $P_i \in P$ in steps 4-6. The polar angles in θ^i are already sorted around P_i.

Steps 7-24 represent the topological sweep which, as we mentioned earlier, runs in $O(n^4)$ on our data. However, steps 10 and 13 in Algorithm 5 are not O(1) and could potentially make the sweep more expensive. Step 10 is hit exactly n times and overall takes $O(kn^2)$. Step 13 could happen $O(n^2)$ times as we mentioned earlier and, therefore, takes $O(n^3 \log n + kn^3)$ in total. We conclude that overall this algorithm takes $O(n^4)$ time, where dominating complexity lies in the topological sweep, establishing Theorem 3.4.1.

We do not store all the vertices in V, but only one per segment in map M. The storage need for this algorithm is $O(n^2)$.

Algorithm 5 returns a point that has maximum colourful simplicial depth along with its CSD. It is simple to modify the algorithm to return a list of all such points if there is more than one. We believe that maintaining such a list will not increase the required storage, i.e. it will contain $O(n^2)$ points throughout the execution of the algorithm, but we haven't proved this.

Theorem 3.4.1. *Given a set of data points P in general position in k different colours, $k \geq 3$, a colourful simplicial median of P can be found in $O(n^4)$ time, where $|P| = n$.*

Chapter 4

Three-dimensional case

In this Chapter we discuss what is known about the simplicial depth in $\mathbb{R}^3$ and why it is hard to adapt to the coloured case. Gil et al. [GSW92] developed an algorithm that finds the monochrome simplicial depth of a point $x \in \mathbb{R}^3$ relative to a dataset P in $O(n^2)$ time, where $n = |P|$. Later, Cheng and Ouyang [CO01] detected a slight flaw in that algorithm and presented a fixed version of it, retaining the running time of $O(n^2)$.

A simplex in $\mathbb{R}^3$ is a tetrahedron, so both algorithms [GSW92] and [CO01] count in how many tetrahedra x is contained. To adapt these results to a colourful setting, we would need to count only the *colourful tetrahedra* that contain x, i.e. tetrahedra with all 4 vertices of distinct colours. Comparing to the two-dimensional case, the problem complicates significantly, especially with colours involved. Just like in the two-dimensional case the points were projected onto a unit circle, in $\mathbb{R}^3$ they are projected on a unit sphere centred at x.

We give an overview of how Cheng and Ouyang [CO01] compute the simplicial depth in $\mathbb{R}^3$ in Section 4.1. Then in Section 4.2 we discuss the challenges of colourizing this problem.

4.1 Simplicial Depth in $\mathbb{R}^3$

Suppose we are given a set of points P in general position in $\mathbb{R}^3$ and a point $x \in \mathbb{R}^3$ in general position with P. We project the points in P onto a unit sphere $\mathcal{B}(x)$ centred at x. A point where the vector from the origin with the direction $(P_i - x)$ intersects the sphere is denoted by θ_i, and its antipode by $\bar{\theta}_i$. The following lemma is from [GSW92]:

Lemma 4.1.1. *Let P_i' be any point on the ray from x through P_i; the tetrahedron $\triangle P_iP_jP_kP_l$ contains x if and only if the tetrahedron $\triangle P_i'P_jP_kP_l$ contains x.*

Corollary 4.1.2. *The tetrahedron $\triangle P_iP_jP_kP_l$ contains x if and only if the tetrahedron $\triangle\theta_i\theta_j\theta_k\theta_l$ contains the origin.*

Definition 4.1.3. *A* spherical triangle *$\triangle_S\theta_i\theta_j\theta_k$ is the area on the surface of the unit sphere bounded by the short arcs of the great circles passing through any pair of points in $\{\theta_i\theta_j\theta_k\}$, given that θ_i, θ_j, and θ_k are not antipodal.*

Then follows Lemma 4.1.4, which is a three dimensional version of Lemma 2.1.5:

Lemma 4.1.4. *The tetrahedron $\theta_i\theta_j\theta_k\theta_l$ contains the origin if and only if the spherical triangle $\triangle_S\theta_i\theta_j\theta_k$ contains $\bar{\theta}_l$.*

Let us denote the set of all $\theta_0, \ldots, \theta_{n-1}$ by θ. Then by Lemmas 4.1.1 and 4.1.4, the simplicial depth of x can be found by counting the number of spherical triangles that contain $\bar{\theta}_i$, for each $\theta_i \in \theta$, where the vertices of the spherical triangle are formed with $\theta \setminus \{\theta_i\}$.

First, we need to make sure to pick a point y that is in general position with $\theta \cup \{x\}$. Then the line $\vec{xy}$ forms an *axis of rotation.* We denote a plane that contains x and is orthogonal to $\vec{xy}$ by Λ. We project the points in θ onto it, and denote the polar angles of these projections by $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$, where $0 \leq \alpha_0 < \alpha_1 < \ldots < \alpha_{n-1} < 2\pi$. Then we relabel the points in θ, so that the indexing corresponds to the α_i's. As usual, $\bar{\alpha_i}$ stands for a point antipodal to α_i.

We need to carefully choose how to draw a plane Π_i, which divides the sphere into two hemispheres, where the upper one contains $\bar{\theta}_i$, and the lower one – θ_i. It also has to retain the property that if we project all the data points onto Π_i, the projections will be in general position, and x will be in general position with them. [CO01] suggests that this plane should contain the axis of rotation and the line on Λ that goes through the origin and has the polar angle $\alpha_i + \epsilon_i/2$. How ϵ_i is chosen is also explained in the paper.

Suppose we have a plane Π_l. Then it contains the origin and separates θ_l from θ_i, θ_j, and θ_k. Let Π'_l be a plane parallel to Π_l, but at some distance from the origin. Denote the *radial projection* of a point p onto Π'_l by $\Pi'_l(p)$. Clearly, $\Pi'_l(p) = \Pi'_l(\bar{p})$. Note that a spherical triangle radially projected on a plane is just a triangle. Hence the lemma (also from [CO01]):

Lemma 4.1.5. *The spherical triangle $\triangle_s\theta_i\theta_j\theta_k$ contains $\bar{\theta}_l$ if and only if, in the plane Π'_l, the triangle $\triangle\Pi'_l(\theta_i)\Pi'_l(\theta_j)\Pi'_l(\theta_k)$ contains $\Pi'_l(\bar{\theta}_l)$.*

The algorithm in [CO01] rotates the dividing plane Π_i for all i. At every step it counts the triangular containments of antipodal points from the upper hemisphere projected on the plane Π'_i with respect to the regular points from the upper hemisphere projected on the plane Π'_i, by Lemma 4.1.5.

Suppose the upper hemisphere $\mathcal{U}_i$ defined by Π_i contains the points $\{\theta_{i+1}, \ldots, \theta_j, \bar{\theta}_{j+1}, \ldots, \bar{\theta}_i\}$. Then the lower one is $\mathcal{L}_i = \{\theta_{j+1}, \ldots, \theta_i, \bar{\theta}_{i+1}, \ldots, \bar{\theta}_j\}$. Let us also denote the regular points, i.e. not antipodes, in $\mathcal{U}_i$ by $\mathcal{U}_i^R$, and the antipodes by $\mathcal{U}_i^A$. Same holds for the lower hemisphere. Therefore, $\mathcal{U}_i^R = \{\theta_{i+1}, \ldots, \theta_j\}$, $\mathcal{U}_i^A = \{\bar{\theta}_{j+1}, \ldots, \bar{\theta}_i\}$, and $\mathcal{U}_i = \mathcal{U}_i^R \cup \mathcal{U}_i^A$. When we rotate Π_i to Π_{i+1}, θ_{i+1} transitions to the lower hemisphere and some points $\theta_{j+1}, \ldots, \theta_{j+k}$ do the opposite. Clearly, a new antipodal point $\bar{\theta}_{i+1}$ comes into the upper hemisphere and $\bar{\theta}_{j+1}, \ldots, \bar{\theta}_{j+k}$ leave. We don't have to perform any calculations for the antipodes that left. However, we have to consider what changes as new both regular and antipodal points come in. Instead of computing the simplicial depth of all antipodes from scratch after every rotation, the following *two steps* are executed:

1. for the points in $\mathcal{U}_i^A \cap \mathcal{U}_{i+1}^A$ calculate the number of spherical triangles that contain them, considering that one or more vertices of the triangles lie in $\mathcal{U}_{i+1}^R \setminus \mathcal{U}_i^R$, and the rest of the vertices – in $\mathcal{U}_i^R \cap \mathcal{U}_{i+1}^R$;

2. compute its simplicial depth of $\bar{\theta}_{i+1}$ relative to $\mathcal{U}_{i+1}^R$;

Further detail on how Cheng and Ouyang fixed the flaw from [GSW92], as well as proof that each tetrahedron containing the origin will be counted twice can be found in [CO01].

4.1.1 Runtime

Projecting the points in P on the sphere, then projecting the points in θ on the plane Λ and sorting them takes $O(n \log n)$. Projecting θ onto the Π_i' for each i is $O(n^2)$ total. Computing the simplicial depth of all antipodes during the first iteration can be done in $O(n^2)$, with the data presorted using the algorithm in [LC85]. Lemma 6 in [GSW92] uses geometric duality to prove that we don't need to sort the points in Π_i''s after each rotation. Instead we remove the points that left from the arrangement, and insert each point that entered in its place. This takes $O(n)$ time for each newly entered point. But since each point switches hemispheres only once during the execution of the algorithm, the amortized runtime is $O(n^2)$. Lemma 7 in [GSW92] also uses to duality to show that computing the triangular containments for every iteration after the first takes $O(n^2)$ in total. Hence, the overall running time of the algorithm that computes the simplicial depth of a point $x \in \mathbb{R}^3$ is $O(n^2)$. Hence the in-sample median can be found in $O(n^3)$ time.

4.2 Challenges of Computing the CSD in $\mathbb{R}^3$

In order to compute the colourful simplicial depth of a point $x \in \mathbb{R}^3$ relative to a data set $P = \{P^1, \dots, P^k\}$ in k different colours, it would be sufficient to follow the Lemma 4.1.5, but count *colourful* spherical triangular containments instead. This is possible as long as we know how to manage the colour arrays and the common array.

Suppose we projected all $\theta^1, \dots, \theta^k$ onto the first plane Π_0. Then we use [LC85] to sort the points. After that we need to do what's described in the end of Section 4.1 as *two steps*. Cheng and Ouyang [CO01] point out that the mistake of Gil et al. [GSW92] was to treat the first iteration differently, which led to the slight overcounting of tetrahedra. The number of tetrahedra that contain x after the rotation from Π_{n-1} to Π_0 was counted twice. That's why we won't differentiate between the rotations, except that we will sort the points for Π_0 and then simply update.

The second step is much simpler to carry out. We compute the CSD of a point $\Pi'_{i+1}\left(\bar{\theta}_{i+1}\right)$ (an antipode that just entered the upper hemisphere) relative to the already sorted set $\{\Pi'_{i+1}(\theta_{i+2}), \ldots, \Pi'_{i+1}(\theta_{j+k})\}$ (all the regular points in current upper hemisphere). Recall that we also keep these as separate colour sets, so doing this should be straightforward.

The first step, however, involves a triangle having one or more vertices from one data set, while the rest comes from a different data set. We could merge these data sets, which in our case is simply $\mathcal{U}^R_{i+1}$, compute the sum of CSDs of the antipodal points in $\mathcal{U}^A_i \cap \mathcal{U}^A_{i+1}$ with respect to $\mathcal{U}^R_{i+1}$, and then subtract the sum of CSDs of the same antipodal points relative to the points in $\mathcal{U}^R_i \cap \mathcal{U}^R_{i+1}$. This seems more expensive than simply running the CSD algorithm for all points in $\mathcal{U}^A_{i+1}$ relative to the set $\mathcal{U}^R_{i+1}$. So that's what we could do at each iteration. This would bring the running time up to $O(n^3k)$, assuming the data was sorted at each iteration.

According to [GSW92], the new arrangement after each rotation can be computed in $O(n^2)$ amortized time, but we are not sure how it translates to the colourful case, where we have the colour classes and the common array. This remains an open question.

In conclusion, if there is no need to sort the data after each rotation of the plane, we approximate the running time of computing the colourful simplicial depth of $x \in \mathbb{R}^3$ with respect to P as $O(kn^3)$. However, if we were to sort the data at each iteration, the overall running time would remain the same, because it takes $O(n^2)$ time to sort the data and then $O(n^2k)$ to compute the CSD of all points in $\mathcal{U}^A_{i+1}$ relative to the set $\mathcal{U}^R_{i+1}$, as mentioned above.

Chapter 5

Conclusions and Questions

Our first main result, Theorem 2.5.1, is an algorithm computing the colourful simplicial depth of a point x relative to a configuration $P = \left(P^1, P^2, \ldots, P^k\right)$ of n points in $\mathbb{R}^2$ in k colour classes in $O(n \log n + kn)$ time, or in $O(kn)$ time if the input is sorted around the origin. If we assume, as seems likely, that we cannot do better without sorting the input, then for fixed k this result is optimal up to a constant factor. It is an interesting question whether we can improve the dependence on k. The space requirement for our algorithm matches the one of the monochrome simplicial depth algorithm, which is $O(n)$.

Computing colourful simplicial depth in higher dimension is very challenging, in particular because there is no longer a natural (circular) order of the points. Non-trivial algorithms do exist as discussed in Chapter 4, but we do not know of any non-trivial algorithms for the general case. We offered in Chapter 4 some preliminary thoughts on extending the 3-d algorithm to the colourful case, but work remains to be done. The general case remains an appealing challenge even for monochrome simplicial depth. Indeed, for $(d+1)$ colours in $\mathbb{R}^d$, it is not even clear how efficiently one can exhibit a single colourful simplex containing a given point [BO97], [DHST08].

The second main result, Theorem 3.4.1, is an algorithm computing the colourful simplicial median of a configuration $P = \left(P^1, P^2, \ldots, P^k\right)$ of n points in $\mathbb{R}^2$ in $O(n^4)$ time, independent of k. This running time is optimal assuming we cannot avoid generating all $\Theta(n^4)$ intersection points of the colourful segments formed by pairs of points from P. One would need to come up with a way of decreasing the pool of candidates for a colourful simplicial median to improve the running time. The space used by our algorithm is $O(n^2)$.

Bibliography

[ABB$^+$09] Jorge L. Arocha, Imre Bárány, Javier Bracho, Ruy Fabila, and Luis Montejano. Very colorful theorems. *Discrete & Computational Geometry*, 42(2):142–154, 2009.

[ABP$^+$16] Karim Adiprasito, Philip Brinkmann, Arnau Padrol, Pavel Paták, Zuzana Patáková, and Raman Sanyal. Colorful simplicial depth, Minkowski sums, and generalized Gale transforms. Preprint. ArXiv:1607.00347, 2016.

[Alo06] Greg Aloupis. Geometric measures of data depth. In *Data depth: robust multivariate analysis, computational geometry and applications*, volume 72 of *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, pages 147–158. Amer. Math. Soc., Providence, RI, 2006.

[ALST03] Greg Aloupis, Stefan Langerman, Michael Soss, and Godfried Toussaint. Algorithms for bivariate medians and a Fermat-Torricelli problem for lines. *Comput. Geom.*, 26(1):69–79, 2003.

[Bal95] Ivan J. Balaban. An optimal algorithm for finding segments intersections. In *Proceedings of the Eleventh Annual Symposium on Computational Geometry*, SCG '95, pages 211–219, New York, NY, USA, 1995. ACM.

[Bar76] V. Barnett. The ordering of multivariate data. *Journal of the Royal Statistical Society. Series A (General)*, 139(3):318–355, 1976.

[Bár82] Imre Bárány. A generalization of Carathéodory's theorem. *Discrete Math.*, 40(2-3):141–152, 1982.

[BO79] J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on Computers*, C-28(9):643–647, Sept 1979.

[BO97] Imre Bárány and Shmuel Onn. Colourful linear programming and its relatives. *Math. Oper. Res.*, 22(3):550–567, 1997.

[BRS06] Michael A. Burr, Eynat Rafalin, and Diane L. Souvaine. Simplicial depth: an improved definition, analysis, and efficiency for the finite sample case. In *Data depth: robust multivariate analysis, computational geometry and applications*, volume 72 of *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, pages 195–209. Amer. Math. Soc., Providence, RI, 2006.

[CE92] Bernard Chazelle and Herbert Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *J. Assoc. Comput. Mach.*, 39(1):1–54, 1992.

[CLR89] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press and McGraw-Hill Book Company, 1989.

[CO01] Andrew Y. Cheng and Ming Ouyang. On algorithms for simplicial depth. In *Proceedings of the 13th Canadian Conference on Computational Geometry*, pages 53–56, 2001.

[DG92] David L. Donoho and Miriam Gasko. Breakdown properties of location estimates based on halfspace depth and projected outlyingness. *Ann. Statist.*, 20(4):1803–1827, 12 1992.

[DHST06] Antoine Deza, Sui Huang, Tamon Stephen, and Tamás Terlaky. Colourful simplicial depth. *Discrete Comput. Geom.*, 35(4):597–615, 2006.

[DHST08] Antoine Deza, Sui Huang, Tamon Stephen, and Tamás Terlaky. The colourful feasibility problem. *Discrete Appl. Math.*, 156(11):2166–2177, 2008.

[EG89] Herbert Edelsbrunner and Leonidas J. Guibas. Topologically sweeping an arrangement. *J. Comput. System Sci.*, 38(1):165–194, 1989. 18th Annual ACM Symposium on Theory of Computing (Berkeley, CA, 1986).

[EM88] Herbert Edelsbrunner and Ernst Peter Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. In *Proceedings of the Fourth Annual Symposium on Computational Geometry (Urbana, IL, 1988)*, pages 118–133. ACM, New York, 1988.

[FR05] Komei Fukuda and Vera Rosta. Data depth and maximum feasible subsystems. In *Graph theory and combinatorial optimization*, volume 8 of *GERAD 25th Anniv. Ser.*, pages 37–67. Springer, New York, 2005.

[GG10] Viliam Geffert and Jozef Gajdoš. Multiway in-place merging. *Theoret. Comput. Sci.*, 411(16-18):1793–1808, 2010.

[GHL$^+$17] Ellen Gethner, Leslie Hogben, Bernard Lidický, Florian Pfender, Amanda Ruiz, and Michael Young. On crossing numbers of complete tripartite and balanced complete multipartite graphs. *Journal of Graph Theory*, 84(4):552–565, 2017.

[GMP08] Maxime Genest, Jean-Claude Masse, and Jean-Francois Plante. depth: Nonparametric depth functions for multivariate analysis. `http://CRAN.R-project.org/package=depth`, 2008. Last updated January 7, 2017.

[GSW92] Joseph Gil, William Steiger, and Avi Wigderson. Geometric medians. *Discrete Math.*, 108(1-3):37–51, 1992.

[HPT08] Andreas F. Holmsen, János Pach, and Helge Tverberg. Points surrounding the origin. *Combinatorica*, 28(6):633–644, 2008.

[KBWZ16] Daniel Kosiorowski, Mateusz Bocian, Anna Wegrzynkiewicz, and Zygmunt Zawadzki. Depthproc: Statistical depth functions for multivariate analysis. `http://CRAN.R-project.org/package=DepthProc`, 2016.

[KM90] Samir Khuller and Joseph S. B. Mitchell. On a triangle counting problem. *Inform. Process. Lett.*, 33(6):319–321, 1990.

[LC85] D. T. Lee and Y. T. Ching. The power of geometric duality revisited. *Inform. Process. Lett.*, 21(3):117–122, 1985.

[Liu90] Regina Y. Liu. On a notion of data depth based on random simplices. *Ann. Statist.*, 18(1):405–414, 1990.

[LPS99] Regina Y. Liu, Jesse M. Pares, and Kesar Singh. Multivariate analysis by data depth: descriptive statistics, graphics and inference. *Ann. Statist.*, 27(3):783–858, 1999. With discussion and a rejoinder by Liu and Singh.

[Mos12] K. Mosler. Depth statistics. *ArXiv e-prints*, July 2012.

[MS17] Frédéric Meunier and Pauline Sarrabezolles. Colorful linear programming, Nash equilibrium, and pivots. *Discrete Appl. Math.*, 2017. To appear.

[MW14] Jirí Matousek and Uli Wagner. On Gromov's method of selecting heavily covered points. *Discrete Comput. Geom.*, 52(1):1–33, 2014.

[Oja83] Hannu Oja. Descriptive statistics for multivariate distributions. *Statistics & Probability Letters*, 1(6):327 – 332, 1983.

[PRTT06] János Pach, Radoš Radoičić, Gábor Tardos, and Géza Tóth. Improving the crossing lemma by finding more crossings in sparse graphs. *Discrete Comput. Geom.*, 36(4):527–552, 2006.

[Ric16] Yann Richet. Java interactive 2d and 3d plots (no opengl). `http://github.com/yannrichet/jmathplot`, 2016. Last updated July 27, 2016.

[RR96] Peter J Rousseeuw and Ida Ruts. Bivariate location depth. *Applied Statistics: Journal of the Royal Statistical Society Series C*, 45(4):516–526, 1996.

[RS08] Eynat Rafalin and Diane L. Souvaine. Topological sweep of the complete graph. *Discrete Appl. Math.*, 156(17):3276–3290, 2008.

[Sar15] Pauline Sarrabezolles. The colourful simplicial depth conjecture. *J. Combin. Theory Ser. A*, 130:119–128, 2015.

[Sma90] Christopher G. Small. A survey of multidimensional medians. *International Statistical Review / Revue Internationale de Statistique*, 58(3):263–277, 1990.

[Tuk75] John W. Tukey. Mathematics and the picturing of data. In *Proceedings of the International Congress of Mathematicians (Vancouver, B. C., 1974), Vol. 2*, pages 523–531. Canad. Math. Congress, Montreal, Que., 1975.

[Zas16] Olga Zasenko. Colourful simplicial depth in the plane. Java code, available at, `http://github.com/olgazasenko/ColourfulSimplicialDepthInThePlane`, 2016. Last updated January 29, 2017.

[Zas17a] Olga Zasenko. Brute force solution of the problem of finding the colourful simplicial depth. Java code, available at, `http://github.com/olgazasenko/BruteForceTriangle`, 2017. Last updated April 1, 2017.

[Zas17b] Olga Zasenko. Points in general position generator. Java code, available at, `http://github.com/olgazasenko/PointInGeneralPositionGenerator`, 2017. Last updated April 1, 2017.

[ZS00] Yijun Zuo and Robert Serfling. General notions of statistical depth function. *Ann. Statist.*, 28(2):461–482, 2000.

[ZS16] Olga Zasenko and Tamon Stephen. Algorithms for colourful simplicial depth and medians in the plane. In *Combinatorial Optimization and Applications - 10th International Conference, COCOA 2016, Hong Kong, China, December 16-18, 2016, Proceedings*, volume 10043 of *Lecture Notes in Comput. Sci.*, pages 378–392. Springer, 2016.

Appendix A

Plots

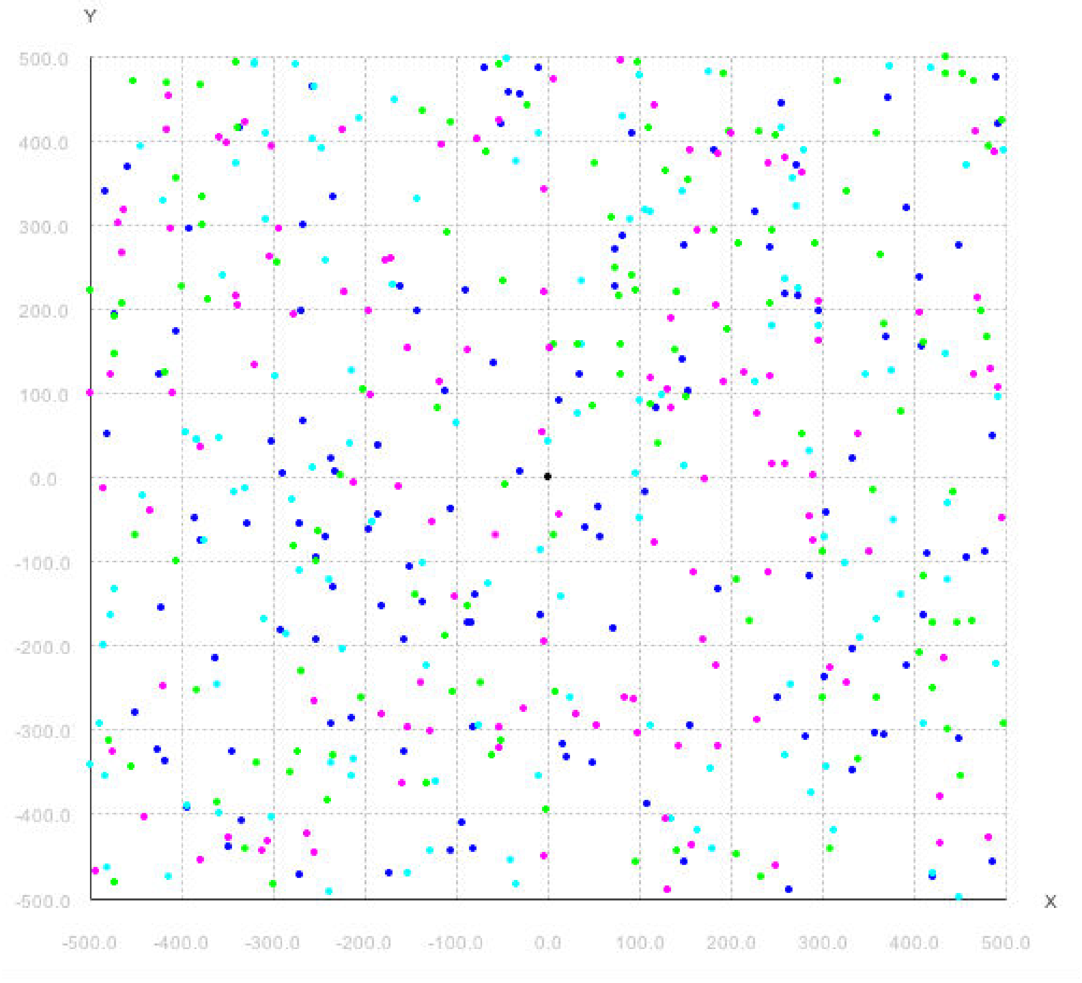


Figure A.1: A configuration P of 500 points in 4 different colours.

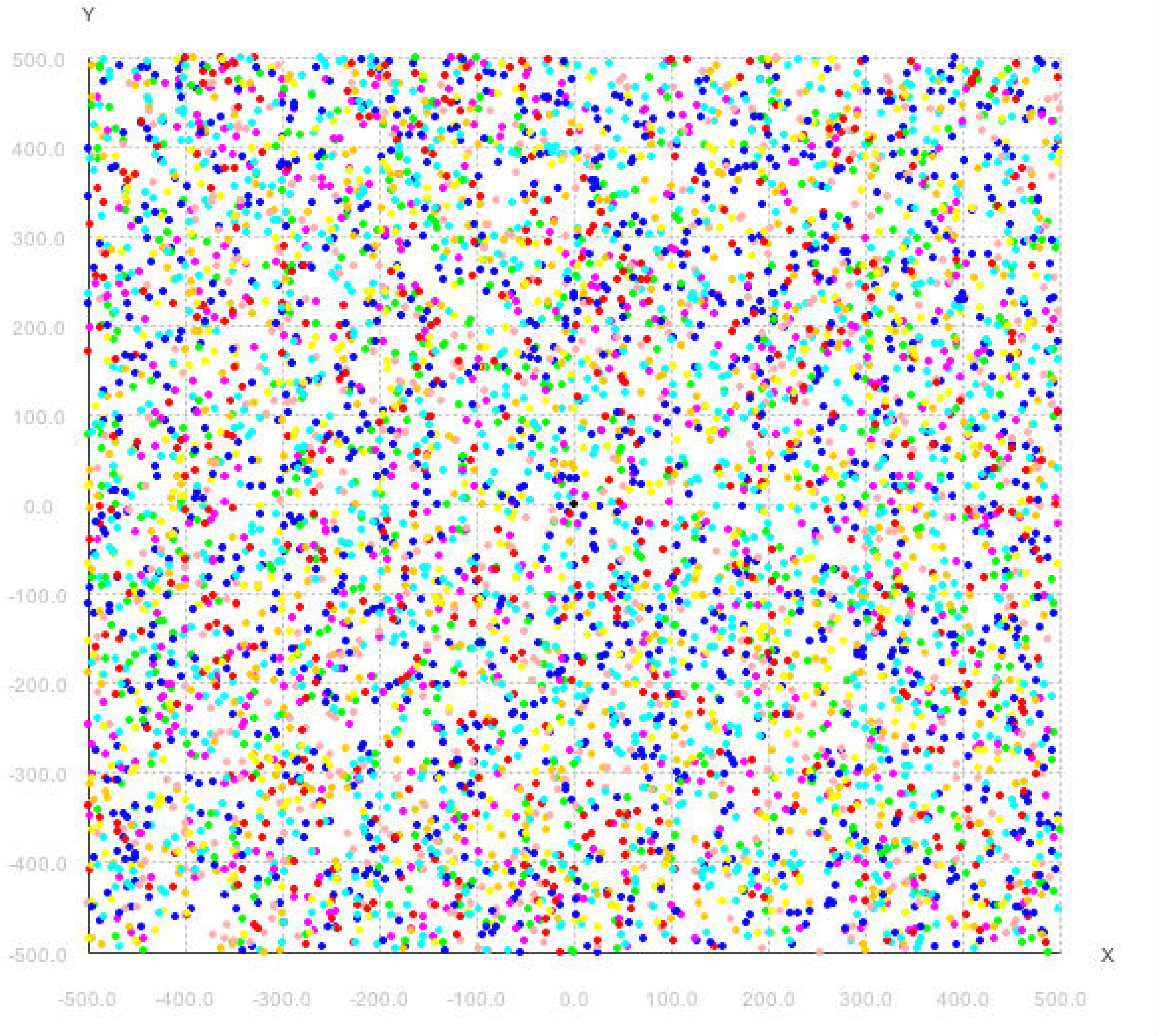


Figure A.2: A configuration P of 6000 points in 10 different colours.

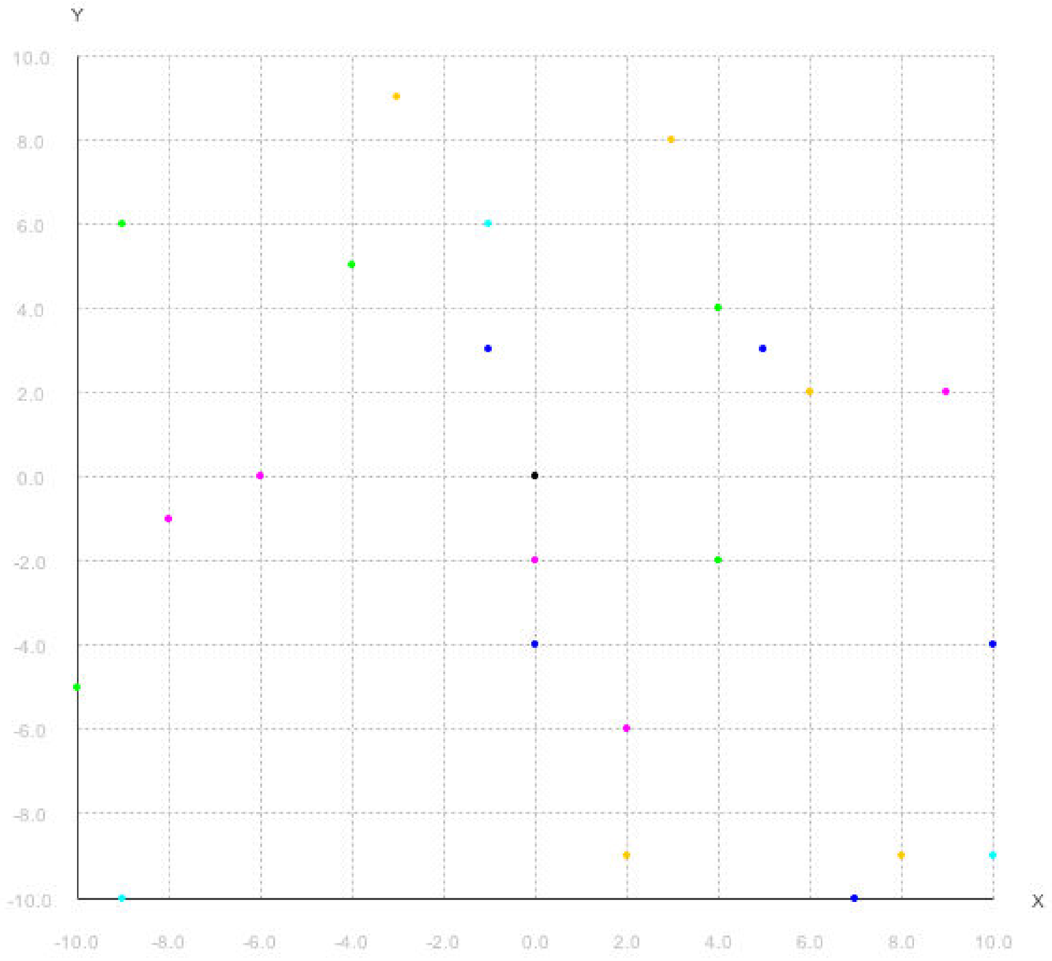


Figure A.3: A degenerate configuration P of 23 points in 5 different colours.

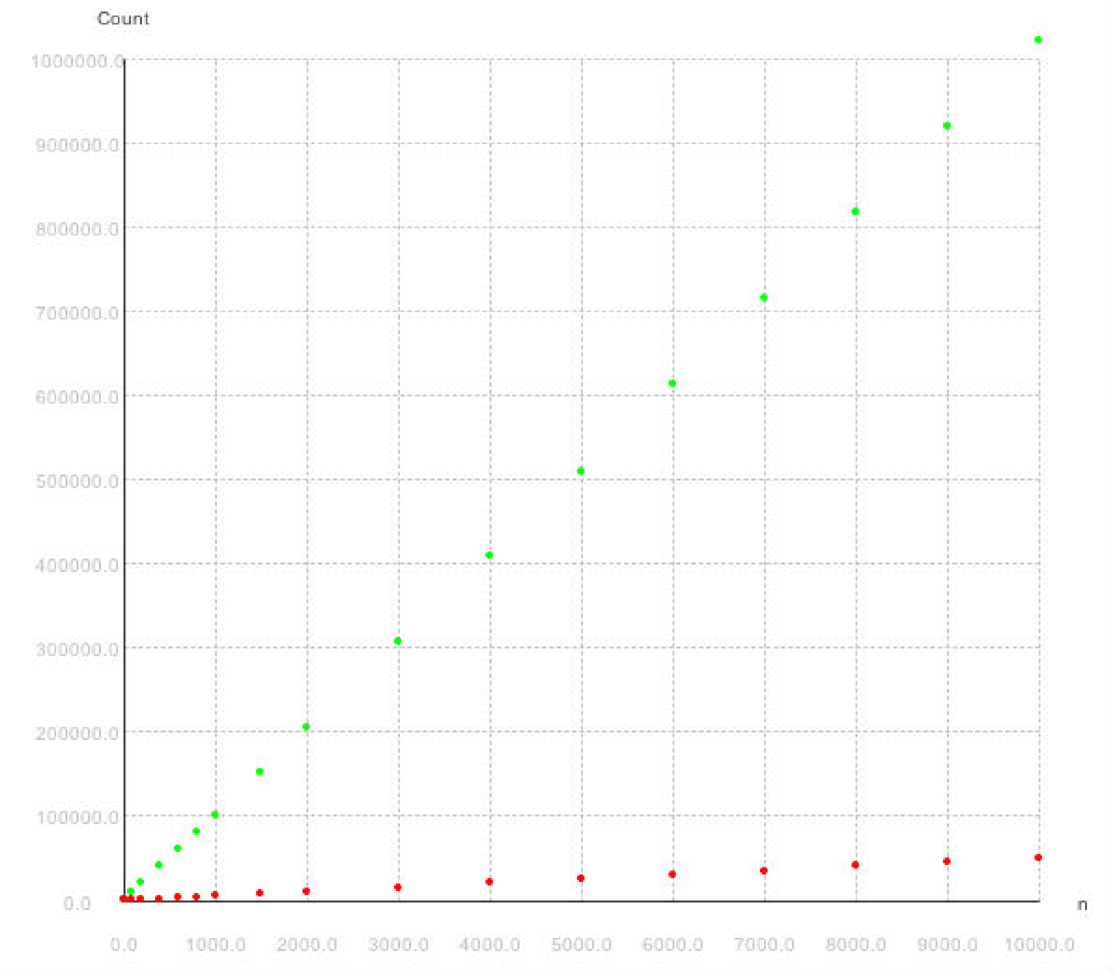


Figure A.4: Operation count versus n, for $k = 3$. Here green indicates the number of floating operations performed, whereas red is the number of comparisons in steps 11 and 14 of Algorithm 1.

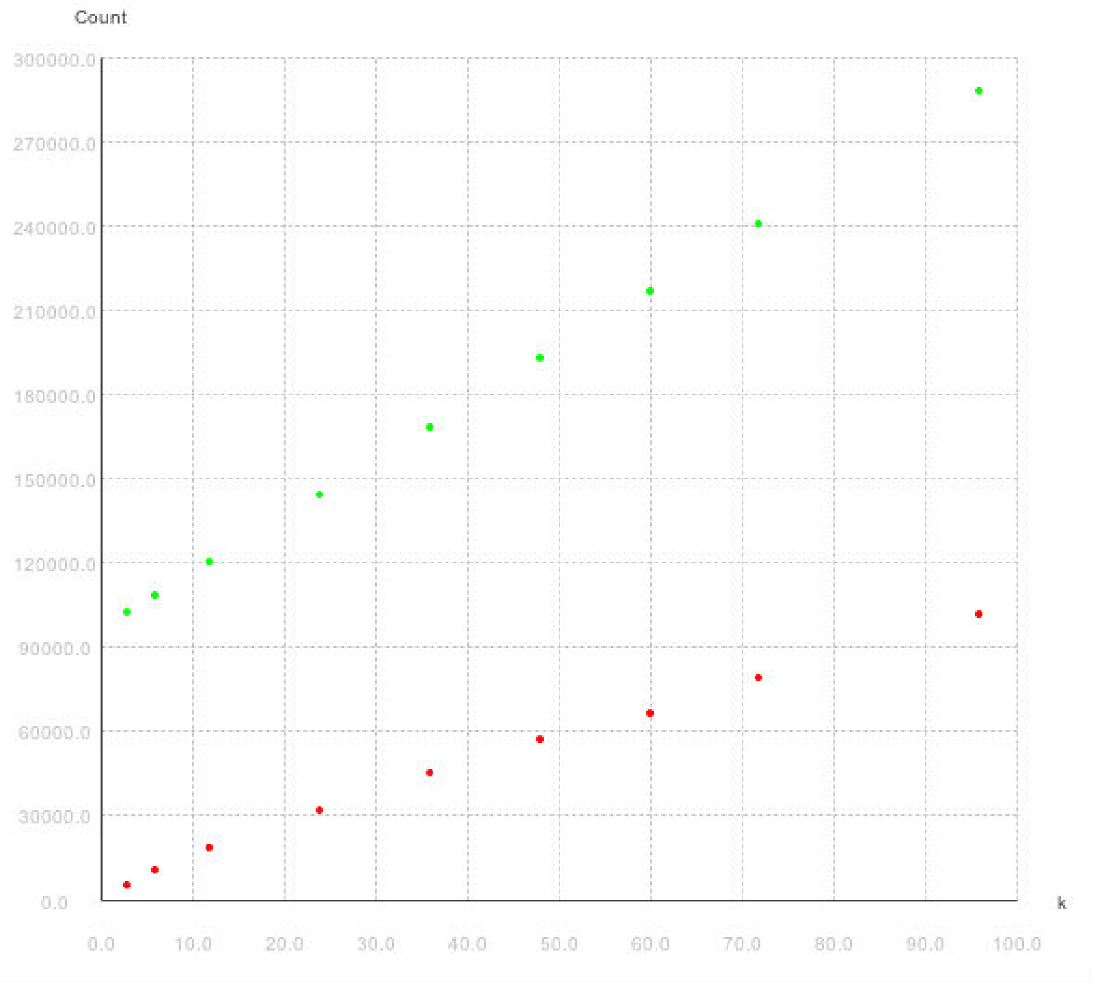


Figure A.5: Operation count versus k, for $n = 1000$. Here green indicates the number of floating operations performed, whereas red is the number of comparisons in steps 11 and 14 of Algorithm 1.