An Algorithm for Approximating the Highest Density Region in d-Space

Marijn T. Waltman

Master's Thesis

Erasmus School of Economics
Erasmus University Rotterdam

Supervisor	Dr. Wilco van den Heuvel
Co-reader	Dr. H.M. Mulder
Date	August, 2014

Abstract

For a given (possibly multivariate) random variable with a known (possibly multimodal) probability density function $f(x)$, the $1-\alpha$ Highest Density Region (HDR) is the smallest possible region (or set of regions) which covers $100(1-\alpha)\%$ of the probability. Often, however, $f(x)$ is unknown and only a sample of X (denoted by V) is known. The algorithm that is discussed in this thesis aims to find an approximation to the $1-\alpha$ HDR in these cases, and in a runtime which is polynomial in $|V|$ (denoted by n). The algorithm makes use of the quantile approach, which states that given a sample y^* of $f(x)$, the $\lceil(1-\alpha)n\rceil$ elements of y^* with the largest f-values are all in the $1-\alpha$ HDR. The sample y^* can be approximated by taking the reciprocal of $w(v)$, the volume of the Voronoi cell of v, for all $v \in V$. The problem thus reduces to finding a subset $X \subset V$ consisting of K connected components of the Delaunay triangulation graph of V (where K is the expected number of modes of $f(x)$) such that $|X| = \lceil(1-\alpha)n\rceil$ and such that the sum of $w(v)$ for all $v \in X$ is minimal. This problem is denoted as the connected component and cardinality-constrained Minimum Weight Connected Subgraph (CC-CC-MWCS) problem and it is an NP-hard problem. The CC-CC-MWCS problem is solved exactly by two Dynamic Programming approaches and one Mixed-Integer Programming approach and it is solved approximately by a heuristic approach. The problem of finding the Delaunay triangulation graph of V and determining the volumes of the Voronoi cells of all $v \in V$ can be done in polynomial time w.r.t. n. Both the runtime and memory usage of the DP approaches is exponential in n and they fail to solve the problem even for low values of n. The MIP approach can be formulated using a linear number of constraints and decision variables and is able to solve problem instances for larger values of n. The heuristic approach is polynomial in n and is shown to find solutions which are very close to optimal. Therefore, using the heuristic approach to solve the CC-CC-MWCS problem means that the full algorithm to find an HDR approximation becomes polynomial in n.

Keywords: highest density region, multivariate, multimodal, Voronoi tessellation, Delaunay triangulation, connected subgraph, weighted graph problem

Preface

The completion of this thesis marks both the end of my Master's degree and the end of my time as a student at the Erasmus University in Rotterdam. My time at Erasmus has been an incredible experience and one which I will remember dearly.

I would like to thank Dr. Wilco van den Heuvel for his constant help and support while writing this thesis. Furthermore, I would like to thank Dr. Martyn Mulder for his very helpful input during our meetings.

I would also like to thank my parents and my brother for their endless support.

Contents

— 1 —

Introduction

For random variables that have a sample space which is infinitely large, it is often desirable to have a finite sized region which consists of the values it will most likely attain. For example: take any normally distributed random variable and then choose one or more closed, non-overlapping intervals such that, when combined, they contain 95% of the probability. The combined region of these intervals is called a 0.95 *density region*, or in general a $1 - \alpha$ density region (where $\alpha = 0.05$ in this example).

Often, however, more than one $1-\alpha$ density region can exist (often infinitely many) with wildly varying shapes and sizes. Therefore it is desirable to know the $1 - \alpha$ density region with the smallest possible size, since this will provide the most compact summary of the probability density function of the random variable. This region is called the $1 - \alpha$ *highest density region* [14].

If the probability density function $f(x)$ of a random variable X is known, then the $1 - \alpha$ highest density region can be computed by finding f_α, the value where

$$\int_{\{x:f(x)\geq f_\alpha\}} f(u)\ du = 1 - \alpha.$$

That is, for any density function $f(x)$ which is strictly positive and differentiable for all x, any constant c will create a finite sized region $R(c)$ consisting of all points x where $f(x) \geq c$. Since $R(c)$ is finite, it will have a finite volume, and since $R(c)$ is a subsection of the area beneath $f(x)$ (which has total volume equal to 1), its volume cannot exceed 1. Therefore the volume of $R(c)$ lies on the interval [0,1). f_α is simply the value of c such that the volume of $R(f_\alpha)$ is equal to $1 - \alpha$. For a graphical representation, see Figure 1.1.

The value of f_α can be determined by a numerical integration approach if $f(x)$ is known [13, 14], but this approach becomes computationally hard as the sample space becomes higher-dimensional. Another approach is called the 'quantile approach', where a given sample of $f(x)$ is sorted in descending order. Then, if we denote n as the sample size, the $\lceil (1 - \alpha)n \rceil$-th element of the sorted sample can be seen as an approximation of f_α. This approach will be further explained in Section 2.2. An advantage of this approach is that it does not become computationally harder as the sample space becomes higher-dimensional. A disadvantage, however, is that it requires a sample of $f(x)$ to

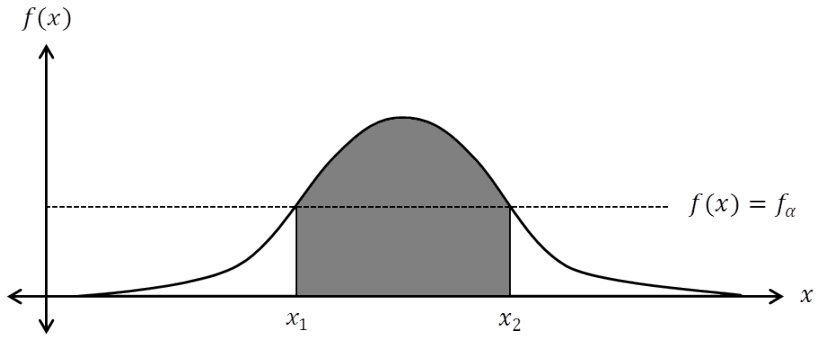

Figure 1.1: An example of the $1 - \alpha$ highest density region for a normally distributed random variable. The grey area is $R(f_\alpha)$ and has a volume equal to $1 - \alpha$; the $1 - \alpha$ highest density region is thus the interval $[x_1, x_2]$.

be known.

The central problem in this thesis is determining the $1 - \alpha$ highest density region when only a sample of X (which is denoted by V) is known. This sample can be from an arbitrarily high dimension. The main question which will be answered throughout this thesis is whether there exists an algorithm to approximate this HDR in polynomial time w.r.t. the size of V and how good the HDR approximation is when compared to the real HDR.

The probability density function $f(x)$ (and thus also a sample of $f(x)$) are presumed to be *unknown*, so in order to apply the quantile approach a sample of $f(x)$ must first be generated. The new algorithm which is proposed in this thesis makes use of the observation that when the probability density in a point is high (resp. low), then on average the distance to its closest neighbors will be small (resp. large). So the *Voronoi cell* of that point[1] will be small (resp. large) and so its volume will be small (resp. big). Therefore, the inverse of the volume of the Voronoi cells of all points from V can be used to approximate a sample of $f(x)$. Then using this sample, a highest density region can be formed out of the union of the Voronoi cells from *a subset of* V. The problem of choosing an appropriate subset of V is a problem which is discussed throughout this thesis.

This approach of using the volume of Voronoi cells to approximate the local density of a point in a set of points is not new. Villagran et al. [17] discuss a different method for approximating the posterior probability density function from a sample where the value of the density function for all points in each Voronoi cell is equal to the reciprocal of the volume of that cell. Duyckaerts and Godefroy [9] use this approach to find the density and distribution of neurones in the central nervous system.

This thesis is structured as follows. In Chapter 2, the concept of the $1 -$

[1]The Voronoi cell of a d-dimensional point v from a set V contains all points in d-space which are closer to v than to all other points in V. For a more detailed definition, see Section 2.3.

α highest density regions will be explained in further detail, and the general idea behind the new algorithm is discussed. The algorithm requires that the Delaunay triangulation of V is computed, which is discussed in Chapter 3. The algorithm also requires that the Voronoi cells of all vertices in V are constructed using the Delaunay triangulation graph, in order to find their volumes. This is discussed in Chapter 4. In Chapter 5, the problem of choosing the best subset of V (which is known as the cardinality-constrained Minimum Weight Connected Subgraph problem) is discussed in detail. In Chapter 6, the runtimes and performances of all previously discussed algorithms are analysed. Finally, Chapter 7 concludes the thesis and proposes directions for future research.

— 2 —

Algorithm Overview

In this chapter the new Highest Density Region (HDR) algorithm is introduced. First, Section 2.1 provides a definition of the $1-\alpha$ HDR of a random variable X. Section 2.2 then discusses the 'quantile approach', which can be used for computing the $1-\alpha$ HDR from any sample of X. Finally, Section 2.3 discusses the new algorithm in detail.

2.1 The Highest Density Region

Let X be a random variable with $\mathbb{R}^d$ as its sample space. Also let $f(x)$ be its probability density function and let α be a constant such that $0 < \alpha < 1$. A $1-\alpha$ *density region* is any region $R \subset \mathbb{R}^d$ such that $\int_R f(x)\ dx = 1-\alpha$, or the probability that X is in R is equal to $1-\alpha$. Then the $1-\alpha$ highest density region is defined as follows by Hyndman [14].

> The $1-\alpha$ *highest density region* (HDR) is the subset $\{x : f(x) \geq f_\alpha\}$, where f_α is the largest constant for which $P(X \in \{x : f(x) \geq f_\alpha\}) = 1-\alpha$.

Another way of viewing the $1-\alpha$ HDR is the $1-\alpha$ density region with the smallest volume of all possible $1-\alpha$ density regions [14].

The $1-\alpha$ density region is a special case of the $(1-\alpha, \gamma)$ *tolerance region*, which is any region R such that the probability that R contains exactly $100(1-\alpha)\%$ of the sample space is equal to γ [7]. The $1-\alpha$ density region is thus the $(1-\alpha, \gamma)$ tolerance region with $\gamma = 1$.

In general, the purpose of HDR's is to provide a summary of a probability distribution $f(x)$ in the form of one or more closed regions (or *clusters*) that contain most of the probability [14]. The number of clusters in the HDR is bounded by the number of modes of $f(x)$, i.e., if $f(x)$ has K modes then the HDR will have at most K clusters.

2.2 Quantile approach

Consider a probability density function $f(x)$ of the random variable X with $\mathbb{R}^d$ as its sample space. Let $x^* = \{x_1, ..., x_n\}$ be a sample of independent

observations of X with $x_i \in \mathbb{R}^d$. Then the $1-\alpha$ HDR can be computed using the *quantile approach* [14] as follows.

> Let $f(x)$ be the known probability density function of the random variable X and define the random variable $Y = f(X)$. Then f_α is the α-th quantile of Y. If $y^* = \{y_1, ..., y_m\}$ is a sample of Y, then f_α can be approximated by the $\lceil (1-\alpha)m \rceil$-th largest element from y^*, denoted by $\hat{f}_\alpha$. Then $\hat{f}_\alpha$ approaches f_α as n approaches ∞. Equivalently, the $\lceil (1-\alpha)m \rceil$ largest elements from y^* are all in the $1-\alpha$ HDR.

That is, if $f(x)$ is a known function, then $\{f(x_1), ..., f(x_n)\}$ is a known sample of Y and can be used to compute the HDR using the quantile approach. However, $f(x)$ is often unknown, as is the case in our problem. In these situations $f(x)$ must first be estimated from a known sample of Y. However, such a sample is also presumed to be unknown and must be generated from x^*. This can be done by e.g. kernel smoothing [18], where the density of a single vertex v is estimated by some predefined kernel function $K_{h_\lambda}(v, V)$ with parameter h_λ. Clearly the accuracy of the estimation of $f(x)$ is dependent on the choice of the kernel function and the choice of the parameter. The approach in this thesis is non-parametric and does not require such parameters to be chosen.

2.3 HDR Algorithm

The approach in this thesis approximates $f(x)$ by the volumes of the Voronoi cells of the vertices in V, and it builds the HDR region(s) from the Voronoi cells of all vertices that are in the HDR. This approach has also been discussed in Admiraal [1], where only unimodal distributions in $\mathbb{R}^2$ are covered. In this thesis a generalization will be introduced, which covers multimodal distributions (as well as unimodal) in arbitrarily high dimensions.

To understand some of the ideas behind the approach, first the concept of a Voronoi cell is defined.

> The *Voronoi cell* of point v (denoted by R_v) consists of all points u whose distance from v is smaller than or equal to the distance from any other point in V. That is, if we denote the distance between any two points u and v as $d(u, v)$, then
>
> $$R_v = \{u \in \mathbb{R}^d \mid d(u, v) \leq d(u, w), \forall w \in V, w \neq v\}.$$

The union of all Voronoi cells of all points in V is called the *Voronoi tesselation* of V. The dual of the Voronoi tesselation is the *Delaunay triangulation graph*. That is, the Delaunay triangulation graph denotes the adjacency between the Voronoi cells and the Voronoi tesselation can be created from the Delaunay triangulation graph. This duality will be used later on in this thesis to determine the volumes of the Voronoi cells in Chapter 4.

The approach in this thesis is based on the observation that for any two points $v_1, v_2 \in V$, if v_1 has a higher probability density than v_2, then v_1 will on average be closer to its closest neighbors than v_2 is to its closest neighbors. This implies that the Voronoi cell of v_1 will be on average smaller than that of v_2, and thus R_{v_1} will have a smaller volume than R_{v_2}.

2.3.1 HDR approximation details

In the following section the approach will be discussed in more detail. Let $w(v)$ denote the volume of R_v (the Voronoi cell of vertex v). Then note that $w(v)$ will be (approximately) proportional to $1/f(v)$. The HDR algorithm aims to find a subset $X \subset V$ where the total sum of the weights is low so that the total sum of the probability density is high. Then the union of the Voronoi cells of X determines the HDR.

Next the concept of the inverse proportionality assumption is introduced.

> The *inverse proportionality assumption* holds when $w(v)$ is exactly proportional to $1/f(v)$ for all $v \in V$.

If this assumption holds, then $\{1/w(v_1), ..., 1/w(v_n)\}$ can be seen as a sample of $f(x)$. Therefore the HDR can be computed by simply using the quantile approach, i.e., the HDR consists of the $\lceil (1-\alpha)n \rceil$ largest elements of $\{1/w(v_1), ..., 1/w(v_n)\}$. However, small random fluctuations in V may distort the Voronoi cells of some vertices, which can result in $w(v)$ not being inversely proportional to $f(v)$ for all v. Since $f(v)$ is unknown it is also unknown a priori whether or not the inverse proportionality assumption holds for V. Therefore this algorithm aims to find an approximation of the real HDR (i.e., the highest density region given the *actual* underlying probability density function) even if the inverse proportionality assumption does not necessarily hold.

Since it is not known whether the subset consisting of the $\lceil (1-\alpha)n \rceil$ vertices with the smallest weights is equal to the real HDR, some additional requirements will be added such that the resulting HDR approximation inhibits some practical properties of the real HDR.

1. The real HDR contains K clusters for some integer $K \geq 1$, so the approximation must also contain K clusters.

2. The clusters of the real HDR contain no holes, since each cluster is essentially a solid d-dimensional shape. Therefore the HDR approximation must also contain no holes.

Next, the method for computing the HDR approximation (including the added requirements) will be discussed. The clusters in the HDR of V can be seen as *disjoint connected components* in the Delaunay triangulation graph. Therefore, the subgraph $G(X)$ from all vertices in X (and with all edges between vertices in X) must consists of K disjoint connected components. Also, holes in the connected components of $G(X)$ can be viewed as clusters of $G(Y)$ (the

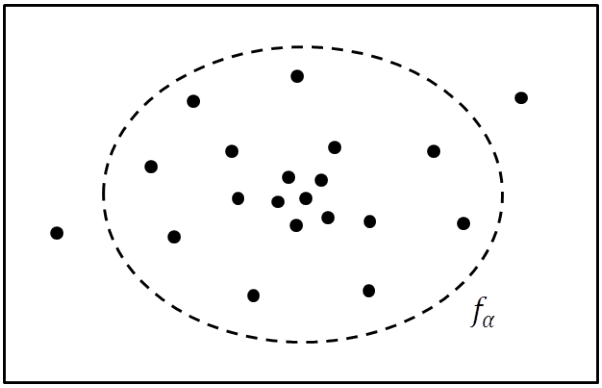

Figure 2.1: A random sample of a bivariate normal distribution with $n = 20$ (denoted by the black dots) and the border of the unknown real HDR with $\alpha = 0.1$, which consists of all points in $\mathbb{R}^2$ where $f(x) = f_\alpha$ (denoted by the dotted ellipse).

subgraph from all vertices in $Y = V \setminus X$ and all edges between vertices in Y). Therefore, these holes can be avoided by requiring $G(Y)$ to consist of exactly 1 connected component. The problem of finding the optimal subset X with minimum total vertex weight such that $G(X)$ has K connected components and $G(Y)$ has exactly 1 connected component will be discussed in Chapter 5.

2.3.2 Equality between HDR approximation and real HDR

Let $X \subset V$ denote the optimal subset from the HDR approximation (i.e. using Voronoi cell volumes to approximate $f(x)$) and let $X^* \subset V$ denote the subset of vertices which lie in the real HDR. See Figure 2.1 for an example of a sample in $\mathbb{R}^2$ with $n = 20$ and $\alpha = 0.1$ and the real HDR. The points inside the dotted ellipse form X^*. In this example it just so happens that $|X^*| = \lceil (1-\alpha)n \rceil = 18$, but due to the random nature of the sample this is not always the case. These are the cases that can occur and the ideal relation between X and X^* in these cases.

- If $|X^*| = \lceil (1-\alpha)n \rceil$, then the ideal approximation is $X = X^*$.
- If $|X^*| < \lceil (1-\alpha)n \rceil$, then the ideal approximation is $X^* \subset X$.
- If $|X^*| > \lceil (1-\alpha)n \rceil$, then the ideal approximation is $X \subset X^*$.

It is unknown which of these cases holds for any random sample, but if $|V|$ approaches ∞, then $|X^*| = \lceil (1-\alpha)n \rceil$ due to the law of large numbers. The following theorem shows that $X = X^*$ if the inverse proportionality holds for the given sample V and if $|V| \to \infty$.

Theorem 1. *Let the number of clusters in $G(X^*)$ be equal to K and let this number be known. If the inverse proportionality assumption holds for the given sample V and if $|V| \to \infty$, then $X = X^*$.*

Proof. First, let X' denote the set of the $\lceil (1-\alpha)n \rceil$ vertices with the smallest weights. Second, let X'' denote the set of the $\lceil (1-\alpha)n \rceil$ vertices with the highest $f(x)$ values. Then, by definition, we know that $X' = X''$ if and only if the inverse proportionality assumption holds for V. Also, if $n \to \infty$, then $X'' = X^*$, which is the fundamental idea behind the quantile approach. Thus we have $X' = X'' = X^*$ if and only if the inverse proportionality assumption holds and $n \to \infty$. Next it will be shown that $X = X'$ if and only if $n \to \infty$, which will conclude the proof.

If X^* meets the requirements that $G(X^*)$ consists of K connected components and $G(Y^*)$ of 1 connected component (where $Y^* = V \setminus X^*$), then X' also meets those requirements, since $X' = X^*$. This then implies that X' meets all two requirements that have been discussed in Section 2.3.1 and has the lowest possible total vertex weight, so therefore $X = X'$. Now it is left to show that X^* meets both requirements.

Firstly, the real HDR consists of K clusters, which implies that all vertices in X^* are distributed over these clusters. It may be that some clusters contain no vertices, but since the probability density in a cluster is strictly positive, this means that if $|V|$ approaches ∞, then all clusters of the real HDR contain at least one vertex from X^*. Next, take one of the clusters and let $C \subseteq X^*$ denote the subset of vertices which are in that cluster. Then $G(C)$ (the subgraph of $G(X^*)$ from all vertices in C and all edges between vertices in C) is a connected subgraph if $|V|$ approaches ∞. To show this, consider a case where C contains a connected component $C' \subset C$ which is *not* connected to the rest of C in $G(C)$. Then any added vertex to C which is in the union of the Voronoi cells of C' will be connected to C' in $G(C)$. Thus one can add vertices to C which are in the union of the Voronoi cells of C' until C' is connected to the rest of C in $G(C)$. This can be repeated for every connected component until $G(C)$ itself is connected. Since these vertices will always be added with nonzero probability as $n \to \infty$, this means that $G(C)$ is always connected as $n \to \infty$, and this counts for all clusters.

This result implies that, if $n \to \infty$, the number of disjoint connected components *can not be greater than* K. To show that it is *equal to* K, note that it can only be less than K if the connected subgraphs from two clusters are connected by an edge. This implies that at least one pair of Voronoi cells of two vertices $x_1, x_2 \in X^*$ from different clusters are adjacent. Note that the probability density between clusters is strictly positive, and also note that an added vertex from Y^* placed in the area between x_1 and x_2 can remove the adjacency between x_1 and x_2. Therefore, if $n \to \infty$, then such a vertex will exist and $G(X^*)$ consists of exactly K disjoint connected components. This means that X^* meets the first requirement.

Next, note that in the real HDR the space outside of the K clusters that make up the highest density region forms 1 cluster. This means that all vertices

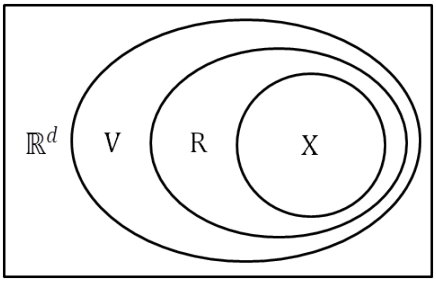

Figure 2.2: A visual representation of the sets used in this thesis.

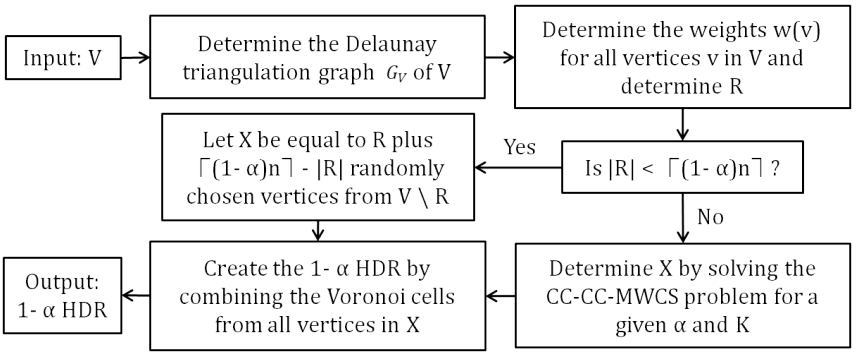

Figure 2.3: A visual representation of the full algorithm.

from Y^* are distributed over 1 cluster, and so using the same reasoning as above these vertices will form exactly 1 connected component. Therefore $G(Y^*)$ consists of exactly 1 connected component. This means that X^* meets the second and last requirement, which means that $X = X'$ if $n \to \infty$. Therefor $X = X' = X'' = X^*$ if $n \to \infty$ and if the inverse proportionality assumption holds for the given sample. □

2.3.3 Full algorithm

Now the full algorithm will be discussed. First denote $R \subset V$ as the subset of vertices of V with finite weight. This is a proper subset of V, since there will always be vertices with infinite weight (namely the vertices which are on the border of the Delaunay triangulation graph). See Figure 2.2 for a visual representation of V, R and X.

A problem that may occur is that $|R| < \lceil (1-\alpha)n \rceil$. In this case the optimal subset is R plus $\lceil (1-\alpha)n \rceil - |R|$ (randomly chosen) vertices with infinite weight. This results in a very inaccurate approximation of the real HDR since the HDR now has infinite size, whereas the original purpose of computing the HDR is to get a *finite sized* area. However, it is expected that the size of R relative to that of V will increase as n gets larger (this will also be discussed in Chapter

6). Therefore, this problem may be avoided by simply increasing the size of the sample.

The entire algorithm is shown graphically in Figure 2.3, and a more detailed overview is given below.

1. Compute the Delaunay triangulation graph $G_V = (V, E)$ of V. This will be discussed in detail in Chapter 3.

2. Given G_V, compute the volumes of the Voronoi cells of each point $v \in V$, denoted by $w(v)$. This will be discussed in detail in Chapter 4.

3. Determine R. If $|R| < \lceil (1-\alpha)n \rceil$, then the solution is R plus $\lceil (1-\alpha)n \rceil - |R|$ randomly chosen vertices from $V \setminus R$. Otherwise, continue with step 4.

4. Given G_V and $w(v)$, find the subset $X \subset V$ such that $\sum_{v \in X} w(v)$ is minimal, $G(X)$ contains at most K disjoint connected components, $G(X)$ contains no holes and $|X| = \lceil (1-\alpha)n \rceil$. This problem is denoted as the CC-CC-MWCS problem and will be discussed in detail in Chapter 5.

5. Given X, the $1-\alpha$ HDR is determined by combining all Voronoi cells of all vertices in X.

— 3 —

Delaunay Triangulations in $\mathbb{R}^d$

In this chapter, the problem of finding the Delaunay triangulation graph of a set of points in $\mathbb{R}^d$ for an arbitrary $d \geq 2$ is discussed. Section 3.1 first provides a definition of the Delaunay triangulation graph of V, and Section 3.2 discusses a method for computing this graph in detail.

3.1 Definition

Consider the set of points $V \in \mathbb{R}^d$ for an arbitrary $d \geq 2$. A *triangulated graph* is a graph which consist solely of d-dimensional *simplices*[1], and every simplex is joint to at least one other simplex by $(d-1)$-dimensional simplices [3]. For example: a triangulated graph in $\mathbb{R}^2$ consists only of triangles which are joint together by line segments, and a triangulated graph in $\mathbb{R}^3$ consists only of tetrahedrons which are joint together by triangles. A *triangulation* of V is equivalent to finding an edge set E such that $G_V = (V, E)$ is a triangulated graph. A Delaunay triangulation is a special type of triangulation and is defined as follows.

> A *Delaunay triangulation* of V is equivalent to finding a triangulation of V such that no vertex $v \in V$ is in the circumscribed hypersphere[2] of any simplex in the triangulation graph.

The Delaunay triangulation graph of V is also unique (i.e. there exists only one edge set E and corresponding graph $G_V = (V, E)$ for which the Delaunay triangulation property holds), given that the points are in general position. An example of the Delaunay triangulation graph in 2-dimensional space is given in Figure 3.1.

To illustrate the Delaunay triangulation property, consider a convex set of 4 points in $\mathbb{R}^2$. For example: consider points 1-4 from Figure 3.1. A triangulation of these points can be done in exactly 2 ways: $T_1 = \{t_1, t_2\} = \{\{1,2,3\},\{2,3,4\}\}$ and $T_2 = \{t_3, t_4\} = \{\{1,2,4\},\{1,3,4\}\}$, where $t_1, ..., t_4$ are triangles denoted by a set of 3 vertices which are its corner points. T_1 is a Delaunay triangulation,

[1] A simplex in d-space is the convex hull of a set of $d+1$ points. For example, a simplex in $\mathbb{R}^2$ is a triangle and a simplex in $\mathbb{R}^3$ is a tetrahedron.

[2] The circumscribed hypersphere of a d-dimensional simplex is the d-dimensional sphere which goes through all vertices of the simplex.

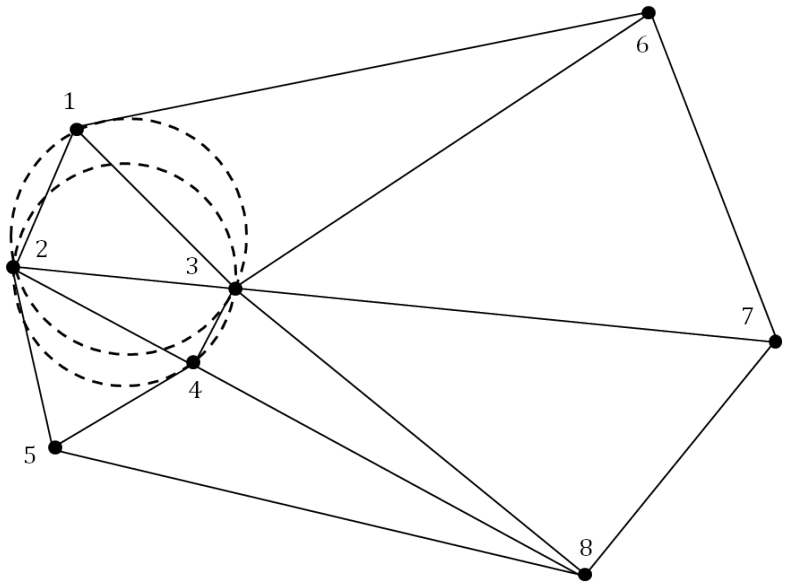

Figure 3.1: The Delaunay triangulation graph of a set of 8 points in $\mathbb{R}^2$. The dashed circles represent the circumscribed circle of the triangles $\{1, 2, 3\}$ and $\{2, 3, 4\}$.

since point 4 is not in the circumscribed circle of t_1 and point 1 is not in the circumscribed circle of t_2. These circumscribed circles are shown in the figure. This does not hold, however, for triangulation T_2 and so it is not a Delaunay triangulation.

3.2 Computation

This section discusses computing the Delaunay triangulation of a set of points $V \in \mathbb{R}^d$. Many different methods exist which only work for $d = 2$ or $d = 3$, however fewer methods exist for arbitrary values of $d \geq 2$. This thesis uses a method that works for an arbitrary $d \geq 2$ which makes use of a relation between Delaunay triangulations and convex hulls [3, pp. 104–105, 4, 5]. The method is described below.

1. For every point $v = [v_1 \dots v_d] \in V$ create the point $v' \in \mathbb{R}^{d+1}$ where $v' = [v_1 \dots v_d, ||v||^2]$. Let V' be the set of all points v' for all $v \in V$.

2. Determine the convex hull of V'. Let F be the set of all faces of the convex hull.

3. Remove all faces from F that are part of the *upper envelope* of the convex hull (i.e all faces which point upwards in the $d + 1$-th dimension)[3]. The remaining faces form the Delaunay triangulation of V.

4. To get the edges of the Delaunay triangulation graph, note that each face is a simplex in $\mathbb{R}^d$ and is determined uniquely by $d+1$ points. Since all points in a simplex are connected to each other by definition, all edges can be determined by connecting every pair of vertices from all faces in F.

The difficulty of computing the Delaunay triangulation in $\mathbb{R}^d$ is thus bounded by the difficulty of computing a convex hull in $\mathbb{R}^{d+1}$. The Quickhull method [4] will be used to compute the convex hull in this thesis. This method first constructs an initial simplex in $\mathbb{R}^d$ from a set of $d+1$ points (which is called $H \subseteq V$) which results in a set of $d+1$ d-dimensional faces (which is called F). It then repeatedly adds new faces to F by joining d points from H and 1 point from $H \setminus V$ and removes redundant faces from F until $H = V$. At that point F will contain all faces of the convex hull. The runtime complexity of this method is $O(n \log n)$ for $d \leq 3$ and $O\left(n^{\lfloor d/2 \rfloor}/\lfloor d/2 \rfloor!\right)$ for $d \geq 4$. That is, the algorithm is polynomial in n for all values of d and exponential in d for $d \geq 4$.

[3]Example: take the 2-dimensional graph from Figure 3.1. Its convex hull consists of the faces (1,6), (6,7), (7,8), (8,5), (5,2) and (2,1). The faces (2,1), (1,6) and (6,7) all point upwards in the 2nd dimension and thus form the upper envelope of the convex hull, and (7,8), (8,5) and (5,2) all point downwards and thus form the lower envelope.

— 4 —

Voronoi Cell Volumes in $\mathbb{R}^d$

In this chapter, the problem of computing the volume of the Voronoi cell of a d-dimensional point $v \in V$ is discussed when the Delaunay triangulation graph of V is known. Section 4.1 first discusses the general concept of this approach and Section 4.2 discusses a way of computing the volumes.

4.1 General concept

Consider a set of points $V \in \mathbb{R}^d$ for $d \geq 2$. Each Voronoi cell is a convex polytope in $\mathbb{R}^d$ and is either bounded (with finite volume) or unbounded (with infinite volume). Because R_v is a convex polytope it can be expressed as an intersection of half-spaces (or 'half-space representation') and can be written down as a system of linear inequalities $A_v x \leq b_v$. The matrix A_v and vector b_v can be determined for any $v \in V$ using only the Delaunay triangulation graph of V (which will be further explained in the next section). Once A_v and b_v are known, then one can compute the volume of the polytope by a number of algorithms. The algorithm which is used in this thesis is by Lasserre [15].

4.2 Computation

4.2.1 Computing A_v and b_v

In this section the process of finding the matrix A_v and vector b_v, which defines the Voronoi cell of v, is discussed in detail. First, denote the Delaunay triangulation graph of V as G_V. Let $N(v)$ denote the set of neighbors of v in G_V. Take any neighbor $u \in N(v)$ with corresponding edge e^{uv} and define $p^{uv} = \frac{1}{2}(u + v)$ as the point in the middle of e^{uv}. Next, create the vector $w^{uv} = u - v$ such that e^{uv} is a line segment on w^{uv}. Then consider the d-dimensional hyperplane H^{uv} which goes through p^{uv} and which has w^{uv} as its normal vector. This hyperplane divides $\mathbb{R}^d$ into two half-spaces: H_v^{uv} which contains v and H_u^{uv} which contains u. For a graphical example in $\mathbb{R}^2$, see Figure 4.1. Now note that H_v^{uv} contains all points which are closer to v than to u. Therefore the Voronoi cell R_v can also be defined as [3]:

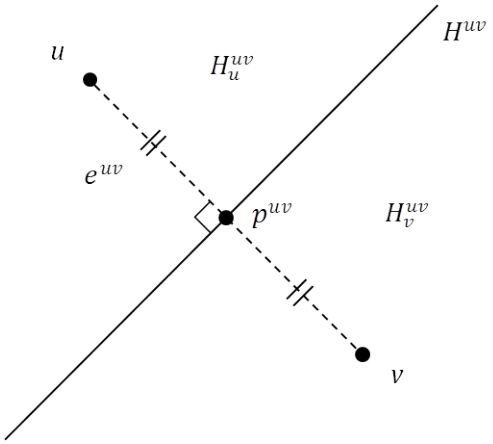

Figure 4.1: Two points $u, v \in \mathbb{R}^2$ and the hyperplane H^{uv} (or line in 2D) which splits the plane into the two half-spaces H_u^{uv} and H_v^{uv}.

$$R_v = \bigcap_{u \in N(v)} H_v^{uv}.$$

The hyperplane H^{uv} can be written as the set of all solutions $x = [x_1 ... x_d]'$ to the equation

$$f^{uv}(x) = a_1^{uv} x_1 + ... + a_d^{uv} x_d = b^{uv}. \tag{4.1}$$

The values of $a_1^{uv}, ..., a_d^{uv}$ and b^{uv} can be determined by noting that any vector $x - p^{uv}$ on H^{uv} must be orthogonal to the normal vector w^{uv}, and so H^{uv} can also be written as the set of all solutions x to the equation

$$w^{uv} \bullet (x - p^{uv}) = 0, \tag{4.2}$$

where $\bullet$ is the dot product operator. After rearranging the terms in (4.2) to match the notation used in (4.1), the values of $a_1^{uv}, ..., a_d^{uv}$ and b can be found:

$$a_i^{uv} = w_i^{uv} \text{ for } i = 1, ..., d \text{ and}$$
$$b^{uv} = w^{uv} \bullet p^{uv}.$$

The half-space H_v^{uv} can then be defined as

$$a_1^{uv} x_1 + ... + a_d^{uv} x_d \leq b^{uv}.$$

However, this only holds if $f^{uv}(v) < b^{uv}$, so therefore $a_1^{uv}, ..., a_d^{uv}$ and b^{uv} must be multiplied by -1 if $f^{uv}(v) > b^{uv}$.

Now A_v and b_v can be determined as follows. Denote the degree of v in G_V as $deg(v) = |N(v)|$ and denote $N(v)$ as $\{n_1^v, ..., n_{deg(v)}^v\}$. Also, define the row vector $a(u, v) = [a_1^{uv} ... a_d^{uv}]$ and the scalar $b(u, v) = b^{uv}$. Then:

$$A_v = \begin{bmatrix} a(n_1^v, v) \\ a(n_2^v, v) \\ \vdots \\ a(n_{deg(v)}^v, v) \end{bmatrix} \quad \text{and } b_v = \begin{bmatrix} b(n_1^v, v) \\ b(n_2^v, v) \\ \vdots \\ b(n_{deg(v)}^v, v) \end{bmatrix},$$

where A_v is a $deg(v) \times d$ matrix and b_v a $deg(v) \times 1$ vector.

4.2.2 Computing the volume

The volume of the Voronoi cell can now be computed by any method which takes only the halfspace-representation of the polyhedron as its input[1]. Examples of these methods include Lasserre's method [6, 15] and Lawrence's method [16]. In this thesis Lasserre's method will be applied. This method computes the volume recursively by making use of the following recursion:

$$Vol(d, A_v, b_v) = \frac{1}{d} \sum_{i=1}^{n} \frac{b_i}{a_{ij}} Vol(d-1, \tilde{A_v}, \tilde{b_v}),$$

where a_{ij} and b_i denote elements of A_v and b_v, respectively, j is chosen such that $a_{ij} \neq 0$, and $\tilde{A_v}$ and $\tilde{b_v}$ are A_v and b_v, respectively, after substituting $x_j = (b_i - \sum_{k \neq j} a_{ik} x_{ik})/a_{ij}$. At the end of the recursion tree $Vol(1, A_v, b_v)$ needs to be computed, which is simply the length of a 1-dimensional line segment. Lasserre's method has a runtime complexity of $O(n^d)$, i.e., the runtime is polynomial in n and exponential in d.

[1] There also exists a vertex-representation of a polyhedron, where the polyhedron is defined by a set of vertices. Methods for computing the volume of a polyhedron given only the vertex-representation also exist, as well as methods which take both representations as inputs.

— 5 —

Cardinality-constrained Minimum Weight Connected Subgraph problem

In this chapter, the cardinality-constrained Minimum Weight Connected Subgraph problem (CC-MWCS) and several new variants of this problem will be discussed in detail. Section 5.1 first provides a definition of the problem. Then several methods for solving the problem will be discussed: two dynamic programming approaches in Section 5.2, a mixed-integer programming approach in Section 5.3 and a heuristic approach in Section 5.4.

5.1 Problem definition

Consider a connected graph $G = (V, E)$, a node weight function $w : V \to \mathbb{R}^+$ and the subset $R \subset V$ of all vertices with finite weight. Then the *cardinality-constrained Minimum Weight Connected Subgraph* (CC-MWCS) problem is defined as follows.

> The CC-MWCS problem is the problem of finding the connected subgraph $G(X) = (X, E(X))$[1] of G with minimum total node weight $\sum_{v \in X} w(v)$ such that $|X| = m$ for some integer m where $1 \leq m < |V|$.

This problem is a variant of the Maximum Weight Connected Subgraph problem, which is reviewed in Álvarez–Miranda et al. [2]. It has been shown to be an NP-hard problem in general, however a polynomial algorithm exists if the input graph is a tree [10]. In our case the input graph is a triangulated graph, which is clearly not a tree, and thus this is an NP-hard problem.

This problem only applies for problem instances where $|R| \geq \lceil (1 - \alpha)n \rceil$, since if this is not the case, then the solution to this problem is trivial (see Section 2.3.3).

In this thesis a new variation of the CC-MWCS problem is introduced: the *connected component and cardinality-constrained MWCS* (CC-CC-MWCS) problem. This variant is defined as follows.

[1] Where $X \subset R$ and $E(X) \subset E$ is the set of all edges between vertices in X.

The CC-CC-MWCS problem is the problem of finding the subgraph $G(X)$ of G with minimum total node weight such that $|X| = m$ for some $1 \leq m < |R|$, $G(X)$ consists of K connected components for some $1 \leq K \leq m$, and $G(Y)$ consists of 1 connected component (where $Y = V \setminus X$ and $G(Y) = (Y, E(Y))$).

This thesis will focus only on the CC-CC-MWCS problem, and more specifically on the following two variants of this problem.

1. The *constrained* variant: the number of connected components in X must be exactly equal to K.
2. The *bounded* variant: the number of connected components in X can be at most K.

5.2 Dynamic Programming approach

The CC-CC-MWCS problem can be solved by a Dynamic Programming (DP) approach. A DP approach can be applied in two ways: a bottom-up approach, which starts from $X = \emptyset$ and repeatedly adds vertices to X until $|X| = m$, or a top-down approach, which starts from $X = R$ and repeatedly removes vertices from X until $|X| = m$. These approaches will be discussed in Sections 5.2.1 and 5.2.2, respectively.

Next some of the notation for both DP approaches will be introduced. A state $s = (s_X, s_k, s_w)$ denotes the current state of the DP algorithm, where s_X is the set of vertices which are currently in X, s_k is the number of clusters in s_X and s_w is the sum of the weights of the vertices in s_X. Both DP approaches use a breadth first search, i.e. for all states s that are visited during stage j of the DP algorithm the cardinality of s_X will be equal to j. The algorithms store two sets of states which are updated before and after each stage: $S^{current}$, which contains all states that are visited in the current stage, and S^{next}, which contains all states that have to be visited in the next stage.

The possible number of states is bounded by the possible number of subsets of V, so both DP algorithms run in $O(n2^n)$ time. The factor n is added since all operations for each state can be done in $O(n)$ time. Also, since at stage j the number of possible states is $\binom{n}{j}$, which can be bounded by $\binom{n}{n/2}$, both DP algorithms use $O\left(\binom{n}{n/2}\right)$ memory.

5.2.1 Bottom-up DP approach

The bottom-up DP approach starts from the initial state $s^0 = (\emptyset, 0, 0)$, i.e. X contains no vertices (and thus it has no clusters) and it starts with a weight equal to 0. An outline of the constrained variant is given in Algorithm 1. If the number of clusters in a given state is less than K, then the vertex to be added to X must increase the number of clusters by one (lines 6-8). Otherwise the

Algorithm 1: Bottom-up DP algorithm (constrained)

Input: A set of points $V \in \mathbb{R}^d$, a node weight function $w : V \to \mathbb{R}^+$, the subset $R \subset V$ with all vertices that have finite weight, the number of clusters K and the number of vertices m to include in the subgraph

1 $s^0 = (\emptyset, 0, 0)$
2 $S^{current} \leftarrow \{s^0\}$
3 **for** $j \leftarrow 0$ to $m - 1$ **do**
4 $S^{next} \leftarrow \emptyset$
5 **for** $s \in S^{current}$ **do**
6 **if** $s_k < K$ **then**
7 **for** $v \in R \setminus s_X$: adding v to s_X creates a new cluster **and** does not create a new hole **do**
8 $S^{next} \leftarrow S^{next} \cup (s_X \cup v, s_k + 1, s_w + w(v))$
9 **else**
10 **for** $v \in R \setminus s_X$: adding v to s_X does not create a new cluster or merge two or more clusters **and** does not create a new hole **do**
11 $S^{next} \leftarrow S^{next} \cup (s_X \cup v, s_k, s_w + w(v))$
12 $S^{current} \leftarrow S^{next}$
13 $s^{opt} \leftarrow \text{argmin}_{s \in S^{current}} \{s_w\}$
14 **return** s_X^{opt}

number of clusters is equal to K and the addition of a new vertex to X must keep the number of clusters equal to K (lines 9-11). This ensures that, at the final stage, $S^{current}$ contains all possible states with exactly K clusters.

The bounded variant is outlined in Algorithm 2. In contrast to the constrained variant, this approach adds vertices to X that do not increase or decrease the number of clusters even when the number of clusters is less than K (lines 9-10). This increases the number of states that are visited and so in general the bounded variant will be slower than the constrained variant of the bottom-up DP approach.

5.2.2 Top-down DP approach

The top-down DP approach starts from the initial state $s^0 = (R, L, \sum_{v \in R} w(v))$ where L is the number of clusters in R. That is, $X = R$ and the starting weight is equal to the sum of all weights which are not infinity (because those vertices will not be in the final solution). The bounded variant is outlined in Algorithm 3. If the number of clusters in a given state is less than K, then the vertex to be removed from V' must increase the number of clusters without exceeding K (lines 6-8). Otherwise, a new state is created with the same number of clusters

Algorithm 2: Bottom-up DP algorithm (bounded)

Input: A set of points $V \in \mathbb{R}^d$, a node weight function $w : V \to \mathbb{R}^+$, the subset $R \subset V$ with all vertices that have finite weight, the number of clusters K and the number of vertices m to include in the subgraph

1 $s^0 = (\emptyset, 0, 0)$
2 $S^{current} \leftarrow \{s^0\}$
3 **for** $j \leftarrow 0$ to $m - 1$ **do**
4 $S^{next} \leftarrow \emptyset$
5 **for** $s \in S^{current}$ **do**
6 **if** $s_k < K$ **then**
7 **for** $v \in R \setminus s_X$: adding v to s_X creates a new cluster **and** does not create a new hole **do**
8 $S^{next} \leftarrow S^{next} \cup (s_X \cup v, s_k + 1, s_w + w(v))$
9 **for** $v \in R \setminus s_X$: adding v to s_X does not create a new cluster or merge two or more clusters **and** does not create a new hole **do**
10 $S^{next} \leftarrow S^{next} \cup (s_X \cup v, s_k, s_w + w(v))$
11 $S^{current} \leftarrow S^{next}$
12 $s^{opt} \leftarrow \text{argmin}_{s \in S^{current}} \{s_w\}$
13 **return** s_X^{opt}

as the old state (lines 9-10).

The constrained variant is essentially the same as the bounded variant, but line 12 is replaced by

$$s^{opt} \leftarrow \underset{s \in S^{current} \,:\, s_k = K}{\text{argmin}} \{s_w\}.$$

The removal of any vertex from X may create more than one new cluster. Since this implies that a state with $s_k = K$ clusters can be achieved at the final stage even if a state in the stage before it has less than $K - 1$ clusters, this limits the opportunities to prune a state (i.e. to stop branching from that state) early on in the DP process (which can be done in the constrained variant of the bottom-up approach). Therefore the constrained variant will be as fast as the bounded variant of the top-down DP approach.

5.2.3 Creating and avoiding new clusters

In the above DP algorithms vertices need to be added and removed from X while creating new clusters or avoiding new clusters from being created in $G(X)$. This may be accomplished in the following ways.

- If adding a vertex v to X must create a new cluster, then v must not be adjacent to any vertex in X.

Algorithm 3: Top-down DP algorithm (bounded)

Input: A set of points $V \in \mathbb{R}^d$, a node weight function $w : V \to \mathbb{R}^+$, the subset $R \subset V$ with all vertices that have finite weight (which has L clusters), the desired number of vertices m in X and the desired number of clusters K

```
1  s^0 = (R, L, Σ_{v∈R} w(v))
2  S^current ← {s^0}
3  for j ← |R| to |V| − m + 1 do
4      S^next ← ∅
5      for s ∈ S^current do
6          if s_k < K then
7              for v ∈ s_X : removing v from s_X creates one or more new
               clusters and the total number of clusters ≤ K and it does not
               create a new hole do
8                  S^next ← S^next ∪ (s_X \ v, s_k+the number of new clusters
                   that are created after removing v, s_w − w(v))
9          for v ∈ s_X : removing v from s_X does not create a new cluster
           and does not create a new hole do
10             S^next ← S^next ∪ (s_X \ v, s_k, s_w − w(v))
11     S^current ← S^next
12 s^opt ← argmin_{s∈S^current} {s_w}
13 return s^opt_X
```

- If adding a vertex v to X must avoid a new cluster from being created, then v must be adjacent to at least one vertex in X.
- If adding a vertex v to X must avoid two or more clusters from being merged into one cluster, then v must be adjacent to only vertices in X that are from the same cluster.
- If removing a vertex v from X must create a new cluster, then v must be a cut vertex (or articulation vertex) of X. The cut vertices of X can be determined in linear time by a depth first search algorithm [11, 12].
- If removing a vertex v from X must avoid a new cluster from being created, then v must not be a cut vertex of X.

Holes in $G(X)$ can be avoided by avoiding new clusters from being created in $G(Y)$.

5.2.4 Completion Bound

Both DP approaches make use of the so called reaching method, which means that new solution states are created from previous solution states. This means

that applying pruning methods[2] can have a big impact on the runtime, since entire subtrees of the Dynamic Programming state tree can be removed at once [8]. One such pruning method is the Completion Bound method, which works as follows.

> First find a feasible solution to the problem by using some heuristic method. Then for the current state in the DP approach, find a lower bound to the solution value if the solution of the current state were to be completed. If this lower bound is higher than the value of the heuristic method, then this means that any continuation of the current state would not lead to a better solution than the current best solution, which is the heuristic solution. Therefore the current state does not have to be continued any further, and so it can be pruned.

To apply the Completion Bound method to the DP approaches, a polynomial heuristic approach is needed. This will be discussed in Section 5.4. The methods for computing the lower bounds for the bottom-up and top-down DP approaches are given below. Note that the solutions for the lower bounds do not have to be feasible, they are merely the best possible continuations of the current solution state.

- Bottom-up: for a given state s, the lower bound is equal to s_w plus the sum of the $(m - |s_X|)$ vertices from the set $R \setminus s_X$ with the lowest weights.
- Top-down: for a given state s, the lower bound is equal to s_w minus the sum of the $(|s_X| - m)$ vertices from the set s_X with the highest weights.

5.3 Mixed-Integer Programming approach

The CC-CC-MWCS can also be solved by a Mixed-Integer Programming (MIP) approach. The MIP approach proposed in this thesis is based on the following definition of a feasible solution of the bounded CC-MWCS.

> Let s be a supernode connected to all vertices in X and let t be a supernode connected to all vertices in Y. Then X is a feasible solution to the bounded variant if $|X| = m$, there exists a spanning tree T_X of $X \cup s$ with s as the root node such that the outdegree of s is equal to K and there exists a spanning tree T_Y of $Y \cup t$ with t as the root node such that the outdegree of t is equal to 1.

Since the connected components of X (or Y) can only be connected through s (or t), this is equivalent to X consisting of K connected components and Y consisting of 1 connected component, which is consistent with the definition of the CC-CC-MWCS. However, the spanning tree of a connected graph can

[2] Any method which determines whether a new state can be omitted if the optimal solution cannot be reached from that state

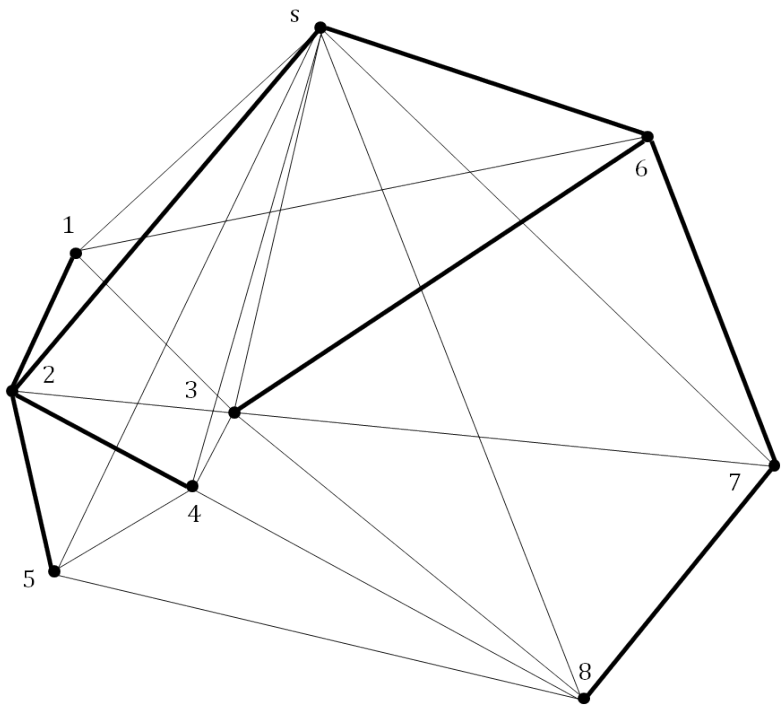

Figure 5.1: An example of a spanning tree of the graph in Figure 3.1 with an added supernode s which has two subtrees. The thick edges are the edges which are in the spanning tree, and the thin edges are the edges which are in the graph, but not in the spanning tree.

contain more than one subtree of the supernode, such that each subtree is connected to at least one other subtree by an edge which is not in the spanning tree (for example, see Figure 5.1). Thus a graph with less than K connected components can also be constructed using the above definition, and the solutions that are consistent with the definition are only feasible for the bounded variant. A feasible solution of the constrained variant, therefore, requires additional constraints. These constraints are discussed in Section 5.3.2.

Note also that for both the constrained and bounded variants it is possible to use a heuristic solution (see Section 5.4) as a starting solution for the MIP approach, which reduces the runtimes. The reduction of the runtimes is dependent on how close the heuristic solution is to the optimal solution.

5.3.1 Bounded variant

In this section the MIP formulation of the bounded variant for a given graph $G = (V, E)$ is discussed. The decision variables that are used throughout this formulation are given below.

- x_i: A binary decision variable, which is 1 if vertex i is in X and 0 if it is in Y.
- y_{ij}^X, y_{ij}^Y: The flow going through edge (i,j) which comes from s (resp. t). Only edges in T_X (resp. T_Y) will have strictly positive flow, otherwise the flow is equal to 0.
- z_{ij}^X, z_{ij}^Y: A binary decision variable, which is 1 if $y_{ij}^X > 0$ (resp. $y_{ij}^Y > 0$) and 0 otherwise.

Additional notation is as follows: $E_s^+ = E \cup s$ and $E_t^+ = E \cup t$ (i.e. E_s^+ and E_t^+ are the set of edges of the graph plus s or t, respectively). The entire formulation is given below.

$$\min \sum_{i \in V} w(i) x_i \tag{5.1}$$

s.t.

$$\sum_{i \in V} x_i = m \tag{5.2}$$

$$\sum_{j \in V} y_{sj}^X = m \tag{5.3a}$$

$$\sum_{j \in V_s^+ : (i,j) \in E_s^+} y_{ji}^X - \sum_{j \in V : (i,j) \in E_s^+} y_{ij}^X = x_i \qquad \forall i \in V \tag{5.3b}$$

$$y_{ij}^X \leq m z_{ij}^X \qquad \forall (i,j) \in E_s^+ \tag{5.3c}$$

$$z_{ij}^X + z_{ji}^X \leq x_i \qquad \forall (i,j) \in E \tag{5.3d}$$

$$\sum_{i \in V} z_{sj}^X = K \tag{5.3e}$$

$$\sum_{j \in V} y_{tj}^Y = n - m \tag{5.4a}$$

$$\sum_{j \in V_t^+ : (i,j) \in E_t^+} y_{ji}^Y - \sum_{j \in V : (i,j) \in E_t^+} y_{ij}^Y = 1 - x_i \qquad \forall i \in V \tag{5.4b}$$

$$y_{ij}^Y \leq (n-m) z_{ij}^Y \qquad \forall (i,j) \in E_t^+ \tag{5.4c}$$

$$z_{ij}^Y + z_{ji}^Y \leq 1 - x_i \qquad \forall (i,j) \in E \tag{5.4d}$$

$$\sum_{j \in V} z_{tj}^Y = 1 \tag{5.4e}$$

$$x_i \in \{0,1\} \qquad \forall i \in V \tag{5.5a}$$

$$y_{ij}^X \in \{0,1,...,m\} \qquad \forall (i,j) \in E_s^+ \tag{5.5b}$$

$$z_{ij}^X \in \{0,1\} \qquad \forall (i,j) \in E_s^+ \tag{5.5c}$$

$$y_{ij}^Y \in \{0,1,...,n-m\} \qquad \forall (i,j) \in E_t^+ \tag{5.5d}$$

$$z_{ij}^Y \in \{0,1\} \qquad \forall (i,j) \in E_t^+ \tag{5.5e}$$

The objective function (5.1) minimizes the sum of the weights of all vertices in X. (5.2) sets $|X| = m$. Constraints (5.3a)–(5.3e) define T_X (the spanning tree of X) as follows. The tree is represented by a directed flow through the graph which starts at s, where the flow is equal to m (enforced by (5.3a)), and the total outflow of a vertex i is always 1 less than its inflow, if i is in X (enforced by (5.3b)). This ensures that T_X is a directed tree without cycles and with s as its root node. Constraint (5.3c) defines z_{ij}^X based on the value of y_{ij}^X. Constraint (5.3d) ensures that an edge (i, j) is a directed edge and so (i, j) and (j, i) cannot both appear in T_X. Lastly, (5.3e) ensures that the outdegree of s is equal to K. Constraints (5.4a)–(5.4e) define T_Y (the spanning tree of Y) in a similar way as T_X has been defined with some differences: T_Y contains $n-m$ vertices, it has t as its root node and the outdegree of t must be equal to 1. Finally, constraints (5.5a)–(5.5e) define the decision variables.

This formulation has a linear number of constraints $(4|V| + 4|E| + 5)$ and a linear number of decision variables $(5|V| + 4|E|)$.

5.3.2 Constrained variant

Next the additional constraints to formulate the constrained variant are discussed. These additional constraints label every vertex in each subtree of s in T_X by the index of the root node of the subtree. This will give each subtree of s its own distinct label. The constraints then prohibit any edge (i, j) where $x_i = x_j = 1$ and the label of i is not equal to the label of j, since this implies that (i, j) connects two subtrees of s, which is exactly what a feasible solution to the constrained variant must not contain. The additional decision variables are given below.

- l_i: The label given to vertex i.
- h_{ij}: A binary decision variable, which is 1 if $|l_i - l_j| > 0$ and 0 if $l_i - l_j = 0$.

The additional constraints are given below.

$$l_i \leq nx_i \qquad \forall i \in V \qquad (5.6)$$

$$l_i \geq (i+1)z_{si}^X \qquad \forall i \in V \qquad (5.7)$$

$$l_i \leq n - (n - (i+1))z_{si}^X \qquad \forall i \in V \qquad (5.8)$$

$$l_i - l_j \leq n - nz_{ij}^X \qquad \forall (i,j) \in E \qquad (5.9)$$

$$nh_{ij} \geq l_i - l_j \qquad \forall (i,j) \in E \qquad (5.10)$$

$$nh_{ij} \geq l_j - l_i \qquad \forall (i,j) \in E \qquad (5.11)$$

$$h_{ij} + x_i + x_j \leq 2 \qquad \forall (i,j) \in E \qquad (5.12)$$

$$l_i \in \{0, 1, ..., n\} \qquad \forall i \in V \qquad (5.13)$$

$$h_{ij} \in \{0, 1\} \qquad \forall (i,j) \in E \qquad (5.14)$$

The next paragraph will provide a brief explanation to the above constraints. Constraint (5.6) sets $l_i = 0$ for all $i \in Y$ while keeping $l_i \leq n$ for all $i \in X$.

Algorithm 4: Heuristic solution to the constrained and bounded variant

Input: A graph $G = (V, E)$, a node weight function $w(v)$, the desired size m of X and the desired number of clusters K, and the subset $R \subset V$ with all vertices that have finite weight. Let R consist of L disjoint connected components.

1 $X \leftarrow \emptyset$

2 Randomly choose $\min\{L, K\}$ vertices from R, such that each vertex is in a different cluster of R, and add these vertices to X

3 **if** $K > L$ **then** randomly choose $K - L$ vertices from $R \setminus X$ such that each vertex forms a new cluster in X, and add these vertices to X

4 Add the remaining $m - K$ vertices by repeatedly choosing the lowest weighted vertex from $R \setminus X$ which is adjacent to exactly one cluster of X

Constraints (5.7) and (5.8) set $l_i = i + 1$ if $i \in X$ and i is directly connected to s, and keep $0 \leq l_i \leq n$ for all other vertices[3]. The labels of all remaining vertices (i.e. all $i \in X$ that are not directly connected to s) are set to the label of the vertex that it is connected to by (5.9). Constraints (5.10) and (5.11) force h_{ij} to be equal to 1 if $|l_i - l_j| > 0$; if $|l_i - l_j| = 0$ then h_{ij} can be 0 or 1, but if necessary it will be forced to 0 by (5.12). Constraint (5.12) ensures that edges where $h_{ij} = x_i = x_j = 1$ (or $h_{ij} + x_i + x_j = 3$), which connect two subtrees of s, are not allowed. Finally, constraints (5.13) and (5.14) define the decision variables.

The total number of constraints for the constrained variant is again linear $(7|V|+8|E|+5)$ and the number of decision variables is also linear $(6|V|+5|E|)$.

5.4 Heuristic approach

A polynomial heuristic approach (i.e. an approximation algorithm in polynomial time) for finding a feasible solution to the CC-CC-MWCS problem is outlined in Algorithm 4. This algorithm forces X to consist of exactly K clusters, so it returns a feasible solution to both the constrained and the bounded variant. The heuristic starts by randomly choosing the first K vertices for each of the K clusters. Then it continues by repeatedly adding the lowest weighted vertex to X such that no new cluster is created and no two or more clusters are merged, until X consists of m vertices.

Since the algorithm starts by selecting K vertices randomly, the heuristic can return any of at most $\binom{|R|}{K}$ different solutions. To improve the value of the heuristic, this algorithm will be executed multiple times, after which the solution with the lowest value will be selected.

Since the last $m - K$ vertices are chosen deterministically, the algorithm can not in general reach every possible solution, and thus in some cases it can not

[3]The label is set to $i + 1$ because the vertices are indexed from 0 to $n - 1$. If vertices are indexed from 1 to n, then the label must be set to i.

reach the optimal solution. Another heuristic approach where the last $m - K$ vertices are chosen randomly (such that no added vertex creates a new cluster or merges two or more clusters) could be implemented, but a metaheuristic (such as simulated annealing) would be required in order to improve the solution value of this approach.

— 6 —

Computational Experiments

In this chapter, all algorithms which have been discussed in this thesis are applied to several randomly generated datasets in order to analyze the differences in the runtimes for different parameter values. These parameters are: d (the dimension of the data), n (the number of observations), K (the number of clusters) and α (the fraction of vertices to be included in the HDR). Also the performance of the heuristic approach will be evaluated.

All algorithms in this thesis have been implemented in Java and they were run on a Windows 7 PC with a 3.10 GHz Intel Core i5-2400 64-bit quad core processor with 4.00 GB of RAM. The MIP formulations have been implemented using the CPLEX API for Java. Multithreading had been enabled during the execution process and up to 4 threads have been used at a time.

6.1 Problem instances

The problem instances are random samples consisting of n independent observations from a d-variate normal distribution with the $d \times 1$ means vector $\mu = [0, ..., 0]'$ and the $d \times d$ covariance matrix $\Sigma = I_d$. If $K = 2$, then the sample is split up over two d-variate normal distributions where one has means $\mu = [0, ..., 0]'$ and the other has means $\mu = [10, 0, ..., 0]'$. The value of 10 is chosen so as to create sufficient separation between both clusters.

Tables 6.1 and 6.2 show some general statistics of the resulting problem instances. Table 6.1 shows the fraction of vertices with finite weight, averaged over 50 instances. It shows that this fraction gets higher as n increases and as d decreases. This is because a graph will contain relatively more border points if the number of points is low and if the number of dimensions is high. Therefore, in order for the set R of vertices with finite weights to be large, it is important that n is also large.

Table 6.2 shows the fraction of 500 datasets where the heuristic could not find a feasible solution for the input values of α and K and for low values of n. Low values of n are examined since complications may arise in these cases, and since the heuristic is used for the DP approaches, which will be tested for low values of n. There may be no feasible solution for $K = 1$ if R contains multiple clusters and no one cluster is larger than m, and for $K = 2$ this may be possible if the structure of R does not permit a solution with 2 connected

		n			
		100	200	400	800
	2	0.9068	0.9503	0.9733	0.9857
d	3	0.7776	0.8703	0.9226	0.9574
	4	0.6104	0.7458	0.8295	0.8993

Table 6.1: Average fraction of vertices with finite weights for different values of n and d.

	K	1			2		
	α	0.1	0.3	0.5	0.1	0.3	0.5
	25	0.050	0.065	0.000	0.830	0.045	0.025
n	50	0.030	0.010	0.000	0.340	0.010	0.005
	100	0.000	0.000	0.000	0.000	0.000	0.000

Table 6.2: Fraction of datasets where the heuristic could not find a feasible solution for low values of n, where $d = 2$ and n, α and K can vary.

components and m vertices. The results show that this fraction decreases as n increases. This may be the result of R increasing in size as n increases, which would increase the number of feasible graphs and thus increase the likelihood of the heuristic finding a feasible solution. It can also be seen that as soon as n increases to a larger value such as 100, then the heuristic can almost always find a feasible solution.

6.2 Delaunay and Voronoi method analysis

The results of the Delaunay triangulation method can be seen in Table 6.3a. The Quickhull algorithm, which was used to compute the Delaunay triangulation graphs, has been implemented manually in Java. The average runtimes seem to increase linearly as n increases and they increase exponentially as d increases, which is consistent with the Quickhull algorithms runtime complexity. Also note that the algorithm runs very quickly even for very high values of n.

Table 6.3b shows the results of the Voronoi cell volume computation algorithm. Lasserre's method, which was used to compute the volumes, has been implemented by making use of the open-source Mines Java Toolkit. Similar to the Delaunay triangulation method, the runtimes of Lasserre's method also seem to increase linearly as n increases and exponentially as d increases, which is consistent with the runtime complexity of Lasserre's method. The runtimes are again very low even for very high values of n.

		n			
		100	200	400	800
d	2	0.0267	0.0421	0.0807	0.1874
	3	0.0849	0.1981	0.5420	1.5060
	4	0.4777	1.5992	5.5895	25.150

(a) Delaunay triangulation

		n			
		100	200	400	800
d	2	0.0005	0.0011	0.0022	0.0045
	3	0.0050	0.0129	0.0314	0.0780
	4	0.5410	2.1469	6.9159	18.702

(b) Voronoi cell volumes

Table 6.3: Average runtimes (in seconds) of the Delaunay triangulation algorithm and the Voronoi cell volume algorithm for different values of n and d.

6.3 Exact CC-CC-MWCS method analysis

The average runtimes over 50 runs of the six described CC-CC-MWCS algorithms can be seen in Table 6.4. These algorithms are the (bounded and constrained) bottom-up and top-down dynamic programming (DP) approaches and the (bounded and constrained) mixed-integer programming (MIP) approaches. The DP approaches all make use of the Completion Bound pruning algorithm.

The runtime differences for changes in the parameter values are given below.

- n: For both DP algorithms the runtimes seem to increase exponentially as n increases, as expected. For the MIP approaches this increase does not seem to grow exponentially, but rather polynomially.

- α: The runtime for the bottom-up approach increases for lower values of α and the runtime for the top-down approach seem to increase for higher values of α. This is clear as lower values of α mean higher values of m for the bottom-up approach (which increases the number of stages the bottom-up approach has to visit) and lower values of m for the top-down approach (which decreases the number of stages the top-down approach has to visit). The runtime of the MIP approaches seems to decrease for lower values of α, which can be attributed to a decrease in the number of feasible solutions if α is low.

- K: The runtimes for the DP approaches does not seem to differ significantly for different values of K. Only if n and α are large does this increase become apparent. This may be because the larger α gets, the lower m gets. In general it is expected that the number of feasible solutions increases as K gets larger and as m gets smaller, however this only becomes apparent

	K	1			2		
	α	0.05	0.25	0.5	0.05	0.25	0.5
	10	0.0135	0.0241	0.0150	*	*	*
n	20	0.1509	0.0659	0.0653	0.1128	0.0738	0.0562
	40	**	**	4.0071	**	**	10.368

(a) Bottom-up DP (constrained)

	K	1			2		
	α	0.05	0.25	0.5	0.05	0.25	0.5
	10	0.0015	0.0006	0.0009	0.0014	0.0008	0.0010
n	20	0.1197	0.0132	0.0026	0.1160	0.0338	0.0036
	40	**	**	3.8349	**	**	10.245

(b) Bottom-up DP (bounded)

	K	1			2		
	α	0.05	0.25	0.5	0.05	0.25	0.5
	10	0.0006	0.0022	0.0006	*	*	*
n	20	0.0009	0.0006	0.0043	0.0002	0.0009	0.0031
	40	0.1075	0.1529	7.3471	0.1653	0.1585	14.090

(c) Top-down DP (constrained)

	K	1			2		
	α	0.05	0.25	0.5	0.05	0.25	0.5
	10	0.0016	0.0004	0.0009	0.0021	0.0005	0.0018
n	20	0.0003	0.0019	0.0028	0.0007	0.0008	0.0032
	40	0.0015	0.0160	7.3342	0.0035	0.0090	14.022

(d) Top-down DP (bounded)

	K	1			2		
	α	0.05	0.25	0.5	0.05	0.25	0.5
	10	0.0322	0.0327	0.0231	*	*	*
n	20	0.0662	0.0759	0.0831	0.2158	0.1188	0.0932
	40	0.1696	0.1733	0.2257	1.6720	0.1921	0.2886

(e) MIP (constrained)

	K	1			2		
	α	0.05	0.25	0.5	0.05	0.25	0.5
	10	0.0116	0.0232	0.0144	0.0231	0.0303	0.0324
n	20	0.0244	0.0506	0.0622	0.0316	0.0478	0.0506
	40	0.1059	0.1348	0.1500	0.1624	0.1481	0.1546

(f) MIP (bounded)

Table 6.4: Runtimes (in seconds) of the various MWCS algorithms for different values of K, α and n.
* n is too low so a feasible solution where $K = 2$ almost always does not exist.
** m is too high so a memory error occurred while running the method.

n	Runtime (in seconds)	Log difference
1000	0.003	-
2000	0.012	0.602
4000	0.051	0.628
8000	0.245	0.682
16000	1.061	0.637

Table 6.5: Runtimes of the heuristic for increasing values of n.

for larger values of n. The runtime of the MIP approach seems unaffected by an increasing K, which is as expected, since the number of constraints and decision variables do not depend on K.

It also appears that there is no significant difference between the constrained and the bounded variant for both DP approaches. Only for the MIP approach does the bounded variant show a significant decrease in the runtimes compared to the constrained variant. This decrease might be the result of the added constraints which results in a tighter formulation and thus a decrease in the runtimes.

Note that the bottom-up DP approach fails to find a solution due to memory problems even at values of n as low as 40 and values of $\alpha < 0.5$. To compare the DP approaches in terms of memory usage: for $\alpha = 0.25$, $d = 2$ and $K = 1$ the bottom-up DP approach fails to find a solution at $n \geq 29$, while the top-down DP approach fails at $n \geq 96$. Because of this issue, it appears that both DP algorithms are not very suitable for any practical problem instances. The top-down algorithm appears to perform well for instance where α is low, but this can be mostly attributed to the instances where the heuristic finds the optimal solution, since in these cases a lot of states can be pruned very early on in the DP process. If the heuristic does not find the optimal solution, then the runtime is substantially higher. For example: if $n = 40$, $K = 1$, $d = 2$ and $\alpha = 0.25$, then the average runtime of the constrained top-down approach when the heuristic solution does not equal the optimal solution increases from 0.1529 to 0.7410 seconds.

6.4 Heuristic analysis

In this section the performance of the heuristic will be discussed. The runtime of one run of the heuristic is shown in Table 6.5 for values of n that increase by a factor of 2 each time. This runtime is very low even for values as high as 16000. For a given n, 'Log difference' denotes log(runtime for n) − log(runtime for $\frac{1}{2}n$). Since this is very similar for all n, it confirms that the runtime of the heuristic approach is indeed polynomial in the input size n.

Table 6.6 shows data about the performance of the heuristic solution. In all cases the heuristic algorithm is executed 50 times and the best solution from those trials is taken as the final solution. It can be seen that the heuristic is able

	K	1			2		
	α	0.1	0.3	0.5	0.1	0.3	0.5
n	25	1.000	0.990	0.965	*	0.965	0.882
	50	1.000	0.989	0.830	0.967	0.953	0.744

Table 6.6: Fraction of problem instances where the heuristic solution is equal to the optimal solution, where $d = 2$ and n, α and K can vary.
* The number of non-feasible solutions is too high.

to find the optimal solution very often, in some instance this was true for 100% (or close to 100%) of all randomly generated datasets. This fraction seems to increase as α gets lower and it seems to decrease as K and n get larger. The effect of n can be attributed to there being more than one solution which are very close to optimal if n gets large.

Also, when the heuristic solution is not equal to the optimal solution, then the average relative deviation from optimality is in general low; for $K = 1$, $\alpha = 0.5$ and $n = 50$ this is approximately 3%, however for $K = 2$, $\alpha = 0.5$ and $n = 50$, this increases to 22%. Overall the heuristic performs very well and could be used as a substitute for the exact CC-CC-MWCS methods in a practical application of this algorithm.

6.5 Real HDR vs. HDR Approximation

In order to make a comparison between the real HDR and the HDR approximation, the volume of the real HDR must first be determined. In our example of the unimodal multivariate normal distribution with $\Sigma = I_d$, this can be done as follows. The $100(1-\alpha)\%$ highest density region in this case is the hyper-ellipsoid denoted by

$$(x-\mu)'\Sigma^{-1}(x-\mu) \leq \chi^2(\alpha, d),$$

where $\chi^2(\alpha, d)$ is defined as the number such that $P[Z > \chi^2(\alpha, d)] = \alpha$ where Z is a $\chi^2(d)$-distributed random variable [7, pp. 608–609]. Or, in other words:

$$F_d\left(\chi^2(\alpha, d)\right) = 1 - \alpha,$$

where $F_d(x)$ is the cumulative density function of the $\chi^2(d)$-distribution. Then the volume $V_d(\alpha)$ of this ellipsoid can be determined by

$$V_d(\alpha) = \frac{2}{d}\frac{\pi^{d/2}}{\Gamma(d/2)}\left(\chi^2(\alpha, d)\right)^{d/2}\det(\Sigma^{1/2}). \tag{6.1}$$

This expression is derived from the volume of a d-dimensional hypersphere with radius $\left(\chi^2(\alpha, d)\right)^{1/2}$ which has been transformed by the linear transformation matrix $\Sigma^{1/2}$. Therefore the volume increases by a factor $\det(\Sigma^{1/2})$.

		α		
		0.1	0.3	0.5
	10	*	*	7.6972
	20	*	14.452	0.1907
n	40	*	0.3704	-0.0510
	80	1.3677	0.0303	-0.1168
	160	0.1196	-0.0390	-0.1235

Table 6.7: Relative difference between volume of approximate HDR (averaged over 50 problem instances) and the theoretical volume.
* The number of problem instances where $|R| \geq \lceil (1-\alpha)n \rceil$ is too low.

If $K = 1$ and $d = 2$, then the $\chi^2(2)$-distribution can be simplified to an exponential distribution with $\lambda = \frac{1}{2}$, which results in $\chi^2(\alpha, d) = -2\ln(\alpha)$ [1, 7]. Then, since $\det(\Sigma^{1/2}) = \det(I_d^{1/2}) = 1$, equation (6.1) reduces to

$$V_2(\alpha) = -2\ln(\alpha)\pi.$$

For higher values of d it becomes increasingly difficult to determine $\chi^2(\alpha, d)$, and subsequently also $V_d(\alpha)$. Therefore only the case where $K = 1$ and $d = 2$ will be examined.

Table 6.7 shows the relative difference between the volume of the approximate HDR (averaged over 50 problem instances) and the theoretical volume as computed by $V_2(\alpha)$. Only problem instances where $|R| \geq \lceil (1-\alpha)n \rceil$ are considered. It can be seen that in general the total volume of the HDR approximation decreases as n increases. This is as expected, since with more vertices the Voronoi cell around each individual vertex decreases and thus the total volume of the solutions will also decrease. The very large differences for very small n may be the result of very oddly shaped Voronoi cells if n is small.

Interestingly, the relative difference becomes negative after n becomes sufficiently large. This implies that the volume of the resulting HDR is smaller than the volume of the theoretical HDR. However, this exact problem can also be seen in the results of Admiraal [1, pp. 13–14], where the combined area of the approximate HDR decreases as n increases and eventually gets smaller than the real HDR volume. Figure 6.1 shows this difference for larger values of n. Because of the high values of n the heuristic approach has been applied. It shows that the relative difference stabilizes around -0.04 for $\alpha = 0.1$, -0.08 for $\alpha = 0.3$ and -0.13 for $\alpha = 0.5$. Thus it seems that, for the bivariate normal distribution, the volume of the approximation is lower than the volume of the real HDR, but this difference decreases as α decreases.

This result may be caused by the problem that occurs when putting straight edged polyhedrons in a curved shape: there will always be some space left over. For an example, see Figure 6.2. Note the large leftover space within the ellipse (real HDR) which is not in the grey area (HDR approximation). This leftover space may decrease for smaller values of α because then the HDR approximation

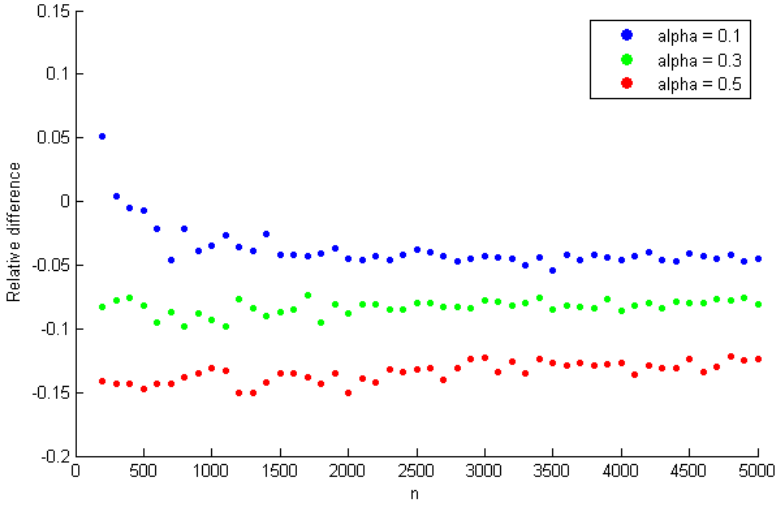

Figure 6.1: Relative difference between volume of heuristic solution and volume of real HDR.

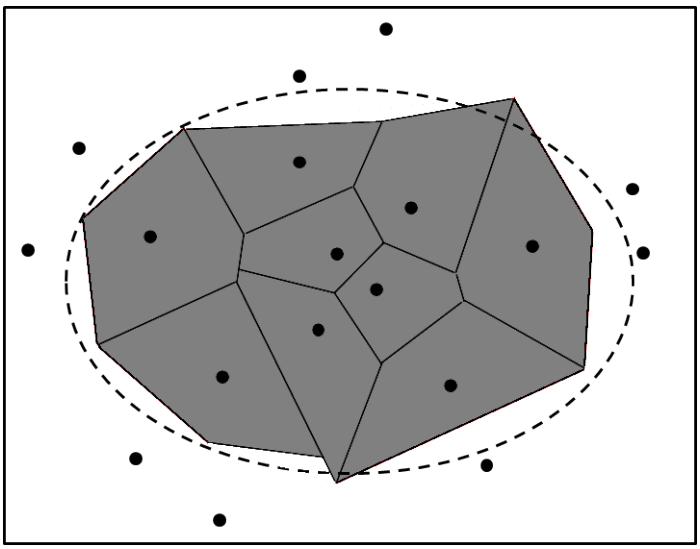

Figure 6.2: An example of a sample in $\mathbb{R}^2$ with $\alpha = 0.5$ and $n = 18$. The grey area is the area of the HDR approximation and the dotted ellipse denotes the border of the real HDR.

consists of more polyhedrons and thus it has more edges on its border. And the more edges the border of the HDR approximation has, the better it will be able to fit the curved shape of the real HDR.

— 7 —

Conclusions

In this thesis the problem of finding the $1-\alpha$ highest density region for a given sample V has been reduced to finding a subset $X \subset V$ such that the subgraph $G(X)$ of the Delaunay triangulation graph of V (with all vertices in X and all edges between vertices in X) has the lowest possible total vertex weight and such that it meets certain requirements. These requirements are: $|X| = \lceil (1-\alpha)n \rceil$, $G(X)$ consists of K disjoint connected components, where K is a user defined constant, and $G(Y)$ consists of 1 connected component, where $Y = V \setminus X$. This problem is denoted as the connected component and cardinality-constrained Minimum Weight Connected Subgraph (CC-CC-MWCS) problem. Two variants of this problem are introduced: the constrained variant (where the number of connected components must be equal to K) and the bounded variant (where the number of connected components can be lower than or equal to K).

The problem of finding the Delaunay triangulation graph of V can be done in polynomial time w.r.t. n and in exponential time w.r.t. d. The problem of constructing the Voronoi cells of all vertices in V and then finding their volumes can also be done in polynomial time w.r.t. n and in exponential time w.r.t. d. The CC-CC-MWCS problem can be solved exactly by two dynamic programming approaches: a bottom-up and a top-down approach. Both approaches run in exponential time w.r.t. n, however the top-down approach still performs reasonably well if α is low and if the heuristic solution can find a solution which is very close to optimal. The problem can also be solved by a mixed-integer programming approach, where both the constrained and bounded variants can be formulated using a polynomial number of constraints and a polynomial number of decision variables.

It has been shown by Theorem 1 that if n approaches ∞ and if the inverse proportionality assumption holds, then the resulting subset X will be the same as the subset of nodes X^* which are in the real HDR. However, the total volume of the HDR approximation has been shown to be less than the total volume of the theoretical HDR for a unimodal bivariate normal distribution if $n = |V|$ is large. However without further testing or a formal proof, it is not clear whether this is true for all distributions (where the $1-\alpha$ HDR is unique).

A randomized, polynomial heuristic approach has been discussed. It has shown to be able to find the optimal solution very often and if it did not find the optimal solution then its value is still reasonably close to the optimal solution

value. Using the heuristic to solve the CC-CC-MWCS problem results in the full algorithm being polynomial w.r.t. n. Therefore to answer the main question of this thesis: a polynomial algorithm to find an approximation of the real $1-\alpha$ HDR does exist if the heuristic method is applied instead of an exact method to solve the CC-CC-MWCS problem. Using a heuristic may even prove to be a necessity in practice, since the solution of this algorithm improves as n gets larger, and thus large initial samples will be required. For future implementations of this algorithm, it is therefore recommended to either use the mixed-integer programming approach for an exact solution of the CC-CC-MWCS problem or the heuristic approach for an approximate solution.

7.1 Future research

Several paths for future research exist on this topic. A better heuristic than the one described in this thesis may be created for the CC-CC-MWCS problem for the bounded variant and also for the constrained variant. For example, the current heuristic uses a bottom-up approach, but a heuristic may also be created using a top-down approach. The current heuristic may also be improved upon by letting both the initial vertices of the clusters and the vertices which are added to the clusters be chosen randomly. This way all possible solutions of the problem can be reached with nonzero probability. A metaheuristic such as simulated annealing or a genetic algorithm can also be applied in this case to improve the performance of the heuristic.

Currently the case where $|R| < \lceil (1-\alpha)n \rceil$ results in a solution where at least one infinitely large Voronoi cell is added to the HDR. This results in a solution with infinite size, which goes against the original goal of computing the HDR: to obtain a finite sized region to summarize a random variable or a sample. Multiple ways of fixing this problem can be tested. For example, observations can be added to V a priori, which will increase $|R|$. This can be repeated until $|R| \geq \lceil (1-\alpha)n \rceil$. The infinite sized Voronoi cells can also be truncated afterwards, such that they end up being finite sized.

Finally, the algorithm can be tested with samples from other distributions than the unimodal bivariate normal distribution to see whether the volume of the HDR approximation is again lower than the volume of the real HDR. A formal proof can also be given to show that this result applies in general. If this result does turn out to be generalizable, then some method needs to be created which would increase the volume of the HDR approximation enough so that it is closer to the volume of the real HDR. An idea for such a method could be to express the Voronoi cells of the HDR by their corner points and to take the convex hull of these points as the HDR approximation. This will have a larger volume than just the total volume of the Voronoi cells and its shape will be much rounder and thus closer to the real HDR. If the real HDR has a non-convex shape, then one could express it as a union of several convex shapes and determine the convex hulls separately.

Bibliography

[1] J. Admiraal. Computing the highest density region using voronoi and delaunay techniques. Bachelor's Thesis, 2013.

[2] E. Álvarez–Miranda, I. Ljubić, and P. Mutzel. The maximum weight connected subgraph problem. In *Facets of Combinatorial Optimization*, pages 245–270. 2013.

[3] F. Aurenhammer, R. Klein, and D. T. Lee. *Voronoi Diagrams and Delaunay Triangulations*. World Scientific, 2013.

[4] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.*, 22(4):469–483, 1996.

[5] K. Q. Brown. Voronoi diagrams from convex hulls. *Information Processing Letters*, 9(5):223–228, 1979.

[6] B. Büeler, A. Enge, and K. Fukuda. Exact volume computation for polytopes: A practical study. In *Polytopes –Combinatorics and Computation*, volume 29 of *DMV Seminar*, pages 131–154. 2000.

[7] V. Chew. Confidence, prediction, and tolerance regions for the multivariate normal distribution. *Journal of the American Statistical Association*, 61 (315):605–617, 1966.

[8] E. V. Denardo and B. L. Fox. Shortest-route methods: 1. reaching, pruning, and buckets. *Operations Research*, 27(1):161–186, 1979.

[9] C. Duyckaerts and G. Godefroy. Voronoi tessellation to study the numerical density and the spatial distribution of neurones. *Journal of Chemical Neuroanatomy*, 20(1):83–92, 2000.

[10] D. S. Hochbaum and A. Pathria. Node-optimal connected k-subgraphs. Manuscript, UC Berkeley, 1994.

[11] L. Holder. Graph algorithms: Applications. http://www.eecs.wsu.edu/~holder/courses/CptS223/spr08/slides/graphapps.pdf.

[12] J. Hopcroft and R. Tarjan. Efficient algorithms for graph manipulation. *Communications of the ACM*, 16(6), 1973.

[13] R. J. Hyndman. An algorithm for constructing highest density regions. Unpublished Manuscript, 1990.

[14] R. J. Hyndman. Computing and graphing highest density regions. *The American Statistician*, 50(2):120–126, 1996.

[15] J. Lasserre. An analytical expression and an algorithm for the volume of a convex polyhedron in $\mathbb{R}^n$. *Journal of Optimization Theory and Applications*, 39(3):363–377, 1983.

[16] J. Lawrence. Polytope volume computation. *Mathematics of Computation*, 57(195):259–271, 1991.

[17] A. Villagran, G. Huerta, M. Vannucci, C. Jackson, and A. Nosedal. Nonparametric sampling approximation via voronoi tesellations. 2013.

[18] M. Wand and M. Jones. *Kernel Smoothing*. CRC Press, 1995.