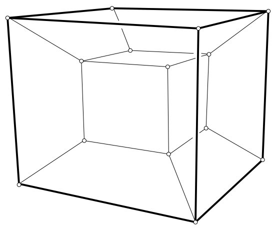
Combinatorial & Computational Properties of the Hypercube

New Results on Covering, Slicing, Clustering, and Searching on the Hypercube



Oswin Aichholzer

e-mail: oaich@igi.tu-graz.ac.at
homepage: http://www.cis.tu-graz.ac.at/igi/oaich

Combinatorial & Computational Properties of the Hypercube

New Results on Covering, Slicing, Clustering, and Searching on the Hypercube

Dissertation for the Award of the Academic Degree
"Doctor of Technical Sciences"
at
Graz University of Technology

submitted by

Oswin Aichholzer

Institute for Theoretical Computer Science (IGI),
Graz University of Technology
A-8010 Graz, Austria

April 1997

First Reader: ao.Univ.-Prof. Dr. Franz Aurenhammer, TU-Graz
Second Reader: Univ.-Prof. Dr. Helmut Alt, FU-Berlin

Kombinatorische & Algorithmische Eigenschaften von Hyperwürfel

Neue Resultate über Abdecken, Schneiden, Clustern und Suchen am Hyperwürfel

Dissertation zur Verleihung des akademischen Grades
"Doktor der Technischen Wissenschaften"
an der
Technischen Universität Graz

vorgelegt von

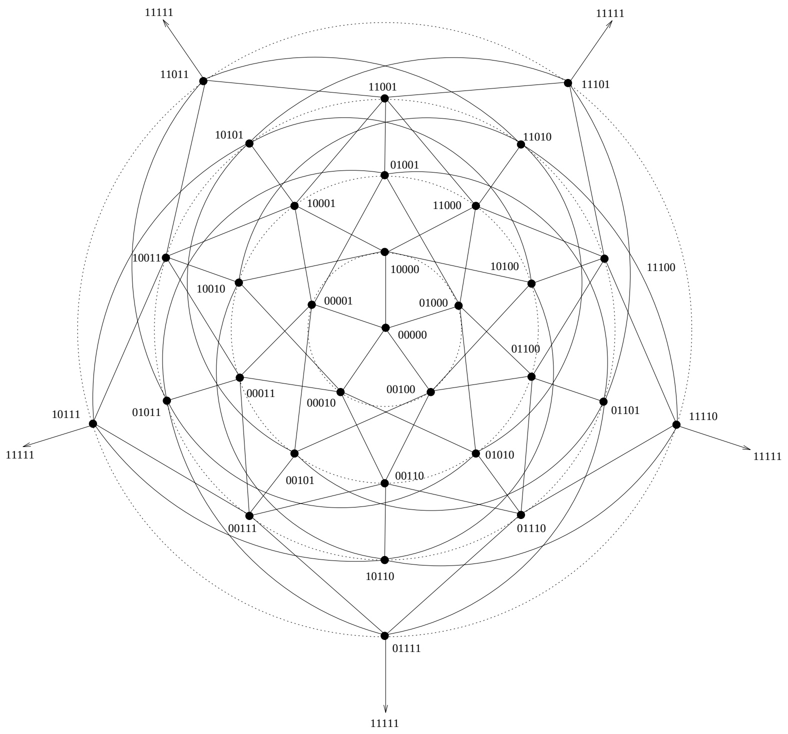
Oswin Aichholzer

Institut für Grundlagen der Informationsverarbeitung (IGI),
Technische Universität Graz
A-8010 Graz, Österreich

April 1997

Diese Dissertation ist in englischer Sprache verfaßt.

Erster Begutachter:	ao.Univ.-Prof. Dr. Franz Aurenhammer, TU-Graz
Zweiter Begutachter:	Univ.-Prof. Dr. Helmut Alt, FU-Berlin



Level separated graph of the five-dimensional hypercube

I hereby certify that the work presented in this thesis is my own and that work performed by others is appropriately cited.

Ich versichere hiermit, diese Arbeit selbständig verfaßt, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfsmittel bedient zu haben.

Abstract / Kurzfassung

Abstract. The central topic of this thesis is the d-dimensional hypercube (d-cube). Despite of its simple definition, the d-cube has been an object of study from various different points of view.

The contributions of this thesis are twofold. In the combinatorial part we investigate the structure of hyperplanes intersecting the d-cube: How many and which types of hyperplanes can be spanned by vertices of the d-cube? What is the minimum number of skew hyperplanes that cover all the vertices of the d-cube? What is the best arrangement of slicing hyperplanes to linearly separate all neighbours on the d-cube? Such results are then used e.g. for determining the maximal number of facets a 5-dimensional 0/1-polytope can achieve.

In the second part of the thesis we consider algorithmical properties of the d-cube. We first obtain efficient clustering methods for objects represented as binary strings of fixed length d, including various agglomerative hierarchical methods like single linkage and complete linkage. Utilizing these hierarchical structures we derive a new efficient approach to the $\{0,1\}$-string searching problem, where for a given set of binary strings of fixed length and a query string one asks for the most similar string in this set.

Motivation for investigating these problems stems, among other areas, from coding theory, communication theory, and learning theory.

Kurzfassung. Zentrales Thema dieser Arbeit ist der d-dimensionale Hyperwürfel. Ungeachtet seiner Einfachheit ist der Hyperwürfel Gegenstand zahlreicher Untersuchungen in verschiedensten Gebieten.

Die Beiträge dieser Dissertation gliedern sich in zwei Teile. Im kombinatorischen Teil wird die Struktur von Hyperebenen, die den Hyperwürfel schneiden, untersucht: Wieviele und welche Hyperebenen können von Ecken des Hyperwürfels aufgespannt werden? Wie lautet die kleinste Anzahl schräger Hyperebenen, die alle Ecken des Hyperwürfels abdecken? Wie sieht die beste Anordnung von schneidenden Hyperebenen aus, um alle benachbarten Ecken am Hyperwürfel zu trennen? Solche Fragen werden dann benützt um z.B. die maximale Anzahl von Flächen von 5-dimensionalen 0/1-Polytopen zu berechnen.

Im zweiten Teil der Arbeit werden algorithmische Probleme untersucht. Es werden effiziente Clusterverfahren für binäre Objekte mit fester Länge entwickelt, wie z.B. diverse agglomerative hierachische Methoden. Unter Ausnutzung der dabei gebildeten baumartigen Strukturen gelangt man zu einem neuen, effizienten Algorithmus für das "$\{0,1\}$-string searching" Problem. Dabei soll aus einer gegebene Menge binärer Objekte das zu einem Anfrage-Objekt ähnlichste ermittelt werden.

Motivation für diese Untersuchungen kommt, neben anderen Gebieten, aus der Kodierungstheorie, Nachrichtentechnik und Lern-Theorie.

To Annemarie, Margaritha and Franz

Acknowledgment & Credits

Acknowledgment. I am indebted to my friends, colleagues, and students for their help and support over the past four years, while I was preparing this theses.

First of all I want to thank my advisor, colleague and friend, Franz Aurenhammer, without whom this thesis would simply not exist. Special mention and thanks go to the members of the institute, especially to Ingrid Preininger for her managerial skills and powerful friendliness.

I have benefited from fruitful discussion with, and have been inspired by, many colleagues from several places. Among all of them I want especially thank Günter Rote who provided several valuable remarks and pointers to the literature.

Last, but not least, I would like to thank my girl friend Annemarie for her love and patience while I was traveling and working long weekends, and my Parents Margaritha and Franz for their great support during all my studies.

Credits. Some of the early work on hypercubes was supported by the Austrian Ministry for Science and Research and the Jubiläumsfond der Österreichischen Nationalbank. Work on Clustering and Searching in the hypercube was supported by the Spezialforschungsbereich F 003, Optimierung und Kontrolle.

Parts of Chapter 2 are based on a paper co-authored with Franz Aurenhammer. Previous versions of Chapters 2,7 and 8 already appeared as papers or technical reports. Parts of some of the mentioned programs were implemented by several students as “EDV-Projekte” under the supervision of the author.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

> At first, indeed, I pretended that I was describing the imaginary experiences of a fictitious person; but my enthusiasm soon forced me to throw off all disguise, and finally, in a fervent peroration, I exhorted all my hearers to divest themselves of prejudice and to become believers in the Third Dimension. Need I say that I was at once arrested and taken before the Council?
>
> An old Square in [1]

> "...Space...is spoken of as having three dimensions, which one may call Length, Breadth, and Thickness. ... But some philosophical people have been asking why three dimensions particularly - why not another direction at right angles to the other three? ... I do not mind telling you I have been at work upon this geometry of Four Dimensions for some time ..."
>
> H. G. Wells in [113]

1.1 Introduction

Among the simplest high-dimensional geometric objects is the d-dimensional hypercube (d-cube), which is the vector sum of d mutually orthogonal line segments of equal length. The d-cube is, beside the simplex and the cross polytope, the only platonic solid which exists in any dimensions. It is a convex polytope that, when having unit edge length, may be expressed as $\mathcal{C}^d = [0,1]^d$. Therefore sometimes, $\mathcal{C}^d$ is called the measure polytope, suggested by its use as the unit of content.

Despite of its simple definition, $\mathcal{C}^d$ has been an object of study from various different points of view. Purely combinatorial properties of $\mathcal{C}^d$, mainly involving certain subgraphs formed by its edges and vertices (the latter are just the various d-tuples of binary digits) have been investigated extensively in coding theory, communication theory, learning theory, and many other fields; see, e.g., [20, 44, 45, 103]. This reveals the fact that the hypercube is just the most natural geometric structure combining all binary digits of length d.

Many easily stated questions concerning the geometry of $\mathcal{C}^d$ are still unsettled or only recently solved. As an example let me mention Keller's long-standing elementary conjecture on hypercube space fillings. Based on some observations of Minkowski about extremal lattices Keller conjectured in 1930 that in any tiling of $\mathcal{R}^d$ by unit hypercubes there are two hypercubes having a complete facet in common. In the following years the conjecture has been proven for dimensions d up to 6. But surprisingly Lagarias and Shor [67] recently disproved the conjecture for $d \geq 10$. It remains open in dimensions 7,8, and 9.

This and other examples given in this thesis show that it might be dangerous to base investigations of the hypercube on the intuition got by considering low-dimensional examples.

In the last years, increased interest in problems involving the placement of hyperplanes in hypercubes can be observed. Motivation stems, among other areas, from coding theory [103], from linear programming [47, 89], and from the theory of machine learning [9, 53, 81, 92]. Again, various elementary questions turned out to be surprisingly difficult. Let us mention just a few of them which are considered in this thesis. How many and which types of hyperplanes can be spanned by vertices of $\mathcal{C}^d$ (cf. Chapter 2)? What is the minimum number of (non-axis-parallel) hyperplanes that cover all the 2^d vertices of $\mathcal{C}^d$ (cf. Chapter 4)? How many edges of $\mathcal{C}^d$ may be cut by a single hyperplane (cf. Chapter 5)? Which hyperplane intersects $\mathcal{C}^d$ in a $(d-1)$-polytope of maximum volume (cf. Chapter 5)?

Other questions are concerned with 0/1-polytopes which are subpolytopes of $\mathcal{C}^d$ spanned by its vertices. Here we ask for example how many facets a 0/1-polytope can achieve, or we are interested in the maximal edge degree of its convex hull (cf. Chapter 3).

All these questions concern combinatorial properties of the hypercube. But of course due to the central role of binary digits in computer science there are also many algorithmic issues concerning the d-cube. Again, motivation stems from the above mentioned fields and various other areas. In this thesis we investigate the general problem of grouping (clustering) binary strings of fixed length d according to a given similarity measure (Chapter 7). This is a central task for data analysis when the objects to be considered are given in binary form. It thus may be of interest for applied fields like medical evaluation or optimization.

Another interesting question is the {0,1}-string searching problem. For a given set S of binary strings of fixed length d and a query string q one asks for the most similar string in S. Thereby the dissimilarity of two given strings is the number of disagreeing bits, that is, their Hamming distance. Utilizing the results obtained for clustering a new approach to this problem can be derived (see Chapter 8).

Section 1.3 gives a detailed overview on the contributions of this thesis to the above mentioned problems.

1.2 Basic Notation

The following notation will be used throughout this thesis. Where it seems appropriate we will not hesitate to repeat parts of it to avoid confusion. Notation and definitions which are only used in single chapters or sections are given where they are needed.

With $\mathcal{C}^d = [0,1]^d$ we denote the d-dimensional hypercube, or *d-cube* for short. The set $V^d = \{0,1\}^d$ is the set of vertices of the d-cube, and can also be seen as the set of all binary strings of length d. The *Hamming distance* of two vertices $x, y \in V^d$ is the number of coordinates where x differs from y. For brevity, we will write $d_H(x,y)$. Note that x and y are the two endpoints of an edge of $\mathcal{C}^d$ iff their Hamming distance is one (and occurs at the i-th bit iff the edge is parallel to the i-th axis of $\mathcal{R}^d$).

By symmetry of $\mathcal{C}^d$, for each vertex $x = (x_1, x_2, \ldots, x_d) \in V^d$ there is a vertex $\overline{x} = (1 - x_1, 1 - x_2, \ldots, 1 - x_d) \in V^d$ which we call the *antipode* of x in $\mathcal{C}^d$.

A *hyperplane* h in Euclidean d-space $\mathcal{R}^d$ is fixed by a pair (v, b), with $v \in \mathcal{R}^d, b \in \mathcal{R}$, and $h = \{x \in \mathcal{R}^d | v \cdot x = b\}$. For brevity, we shall also write $h = (v, b)$.

Let us denote with $\mathcal{H}(\mathcal{C}^d)$ the set of all hyperplanes that can be spanned by (d affinely independent) vertices of $\mathcal{C}^d$. It follows from linear algebra (Cramer's rule) that there is always an integer solution for v provided $h \in \mathcal{H}(\mathcal{C}^d)$. It will be assumed that v is an integer vector and as short as possible, i.e., the gcd of its entries is one. For convenience, we will not distinguish between a vector and its transposed form.

1.3 Overview of the Results

Chapters 2 to 6 are devoted to combinatorial problems on the hypercube.

We start with Chapter 2 by investigating hyperplanes which are spanned by vertices of $\mathcal{C}^d$. We classify these hyperplanes by parallelism to coordinate axes, by symmetry of the d-cube vertices they avoid, as well as by so-called hull-honesty. (Hull-honest hyperplanes are those whose intersection figure with the d-cube coincides with the convex hull of the d-cube vertices they contain; they do not cut d-cube edges properly.) We describe relationships between these classes, and give the exact number of hull-honest hyperplanes, in general dimensions. An experimental enumeration of all spanned hyperplanes up to dimension eight showed us the intrinsic difficulty of developing a general enumeration scheme.

0/1-polytopes are subpolytopes of the d-cube, spanned by its vertices. Thus the faces of 0/1-polytopes are supported by the hyperplanes investigated in Chapter 2. In Chapter 3 we consider the convex hull of these 0/1-polytopes and partially answer an open question of Ziegler [117]: What is the maximal number of facets a (5-dimensional) 0/1-polytope can achieve? We show that this maximum is 40 for $d = 5$. We also give a complete enumeration of all equivalence classes of 0/1-polytopes of dimension 5. Moreover we show that the degree of a vertex of the convex hull can exceed 2^{d-1} already for $d \geq 5$. Numerical experiments indicate that the number of edges of a random 0/1-polytope depends most likely quadratically on the number of vertices.

Using the results of Chapter 2 we ask in Chapter 4 the following questions which are of interest in coding theory: What is the minimum number of skew (non-axis-parallel) hyperplanes that cover all the 2^d vertices of the d-cube $\mathcal{C}^d$? And what is the maximum number of vertices of $\mathcal{C}^d$ that can be covered by k skew hyperplanes? We give general bounds and the exact numbers for both problems if either $d \leq 6$ or $k \leq 2$. Especially we can show that for $d \geq 5$ strictly less than d skew hyperplanes are sufficient but that at least d skew hyperplanes are needed for $d \leq 4$.

So far we were interested in hyperplanes spanned by the vertices of $\mathcal{C}^d$. To the contrary in Chapter 5 we investigate hyperplanes which avoid the vertices of the d-cube and we are only interested in the edges of $\mathcal{C}^d$ they intersect. Motivation for considering such hyperplanes mainly stems from learning theory, where they are used to linearly separate subsets of hypercube vertices. We give the number and types of such hyperplanes for $d \leq 8$ and, similar to the covering problem of Chapter 4, we investigate the slicing problems: What is the minimum number of hyperplanes slicing all $d2^{d-1}$ edges of $\mathcal{C}^d$? And how many edges can be sliced by

k hyperplanes? We also give relations of such sets to learning theory, namely to the parity problem.

Various problems concerning the Hamming distance between vertices of the hypercube are considered in Chapter 6. Here we give criteria to maximize the minimum Hamming distance between vertices, for vertex sets lying in a hyperplane as well as for arbitrary subsets of V^d. We also consider the question for how many vertices of $\mathcal{C}^d$ a vertex $v \in \mathcal{C}^d$ can simultaneously be the (unique) nearest neighbour.

Chapters 7 and 8 concern algorithmic problems on the hypercube. Here we use the results derived in the previous chapters to design efficient algorithms or to show that the combinatorial complexity of several problems is too large to provide practical solutions.

In Chapter 7 we consider various clustering methods for objects represented as binary strings of fixed length d. The dissimilarity of two given objects is the number of disagreeing bits, that is, their Hamming distance. Clustering these objects can be seen as clustering a subset of the vertices of a d-dimensional hypercube, and thus is a geometric problem in d dimensions. We give algorithms for various agglomerative hierarchical methods (including single linkage and complete linkage) as well as for two-clusterings and divisive methods.

We only present linear space algorithms since for most practical applications the number of objects to be clustered is usually too large for non-linear space solutions to be practicable.

The results obtained for clustering are then utilized to derive a new approach to a persistent problem for long binary strings, which was one of the beginnings of this thesis: In Chapter 8 we consider the {0,1}-string searching problem. For a given set S of binary strings of fixed length d and a query string q one asks for the most similar string in S. We present an efficient {0,1}-string searching algorithm based on hierarchical pre-clustering. To this end we give several useful observations on the inter- and intra-cluster distances.

All algorithms are easy to implement and the constants in their asymptotic runtime are small. We give experimental results for all methods considered, and for uniformly distributed hypercube vertices as well as for specially chosen sets. These experiments indicate that our algorithms work well in practice.

Finally Chapter 9 gives a brief overview on the literature of hypercube related problems which are not mentioned in the previous chapters. Of course this task has to be incomplete and we thus restrict the list to a small number of prominent problems, namely triangulating the hypercube, volumes of 0/1-polytopes, and related NP-complete problems, respectively.

We then conclude the thesis by showing that even for the 2-dimensional hypercube (the plane square) there are simple-looking but nevertheless open problems.

Chapter 2

Hyperplanes in Hypercubes

2.1 Introduction and Special Notation

In the last years, increased interest in problems involving the placement of hyperplanes in hypercubes can be observed. Motivation stems, among other areas, from coding theory [103], from linear programming [47, 89], and from the theory of machine learning [9, 53, 81, 92].

In this chapter we consider hyperplanes spanned by vertices of the unit d-cube. The main question will be how many and which types of hyperplanes can be spanned by vertices of $\mathcal{C}^d$. We classify these hyperplanes by parallelism to coordinate axes, by symmetry of the d-cube vertices they avoid, as well as by so-called hull-honesty. (Hull-honest hyperplanes are those whose intersection figure with the d-cube coincides with the convex hull of the d-cube vertices they contain; they do not cut d-cube edges properly.) We describe relationships between these classes, and give the exact number of hull-honest hyperplanes, in general dimensions. An experimental enumeration of all spanned hyperplanes up to dimension eight showed us the intrinsic difficulty of developing a general enumeration scheme.

An attempt is made to characterize and classify the hyperplanes in $\mathcal{H}(\mathcal{C}^d)$. An obvious criterion is *parallelism*, where we distinguish between k-parallel hyperplanes (being parallel to exactly $k > 0$ coordinate axes) and skew (0-parallel) hyperplanes. A concept that turns out to be important is *symmetry*, which is fulfilled by hyperplanes that halve the set of d-cube vertices they avoid.

We show that k-parallel hyperplanes in $\mathcal{H}(\mathcal{C}^d)$ correspond to skew ones in $\mathcal{H}(\mathcal{C}^{d-k})$, and that skew and asymmetric hyperplanes in $\mathcal{H}(\mathcal{C}^d)$ correspond to skew and symmetric ones in $\mathcal{H}(\mathcal{C}^{d+1})$, in a unique manner. This tells us that the skew and symmetric type is the only really interesting one. The enumeration of all hyperplanes of this type, however, is complicated by the following unpleasant phenomenon: In each further dimension new kinds of hyperplanes appear that seem not to be accessible by using the results for lower dimensions. In fact, a theorem by Naumann [80] gives some evidence for the intrinsic complexity of $\mathcal{H}(\mathcal{C}^d)$: Every $(d-1)$-polytope is obtainable as the intersection of a $(d-1)$-dimensional hyperplane and a hypercube of sufficiently high dimension.

We were able to enumerate experimentally the sets up to $\mathcal{H}(\mathcal{C}^8)$, but did not succeed in developing an enumeration scheme for $\mathcal{H}(\mathcal{C}^d)$ for general d. We rediscovered an upper bound on the size of the normal vector of hyperplanes in $\mathcal{H}(\mathcal{C}^d)$ which we needed to speed up the enumeration.

The situation gets strikingly simpler if we restrict attention to hyperplanes in $\mathcal{H}(\mathcal{C}^d)$ that do not (properly) cut d-cube edges. Those might be called *hull-honest* hyperplanes as their intersection figure with $\mathcal{C}^d$ coincides with the convex hull of the vertices of $\mathcal{C}^d$ they contain. We give an easy-to-use characterization of hull-honest hyperplanes that leads us to their exact total number, in general dimensions.

A previous version of this chapter appeared as [6].

2.2 Parallel and Skew Hyperplanes

Our set of interest, $\mathcal{H}(\mathcal{C}^d)$, can be partitioned into classes by considering parallelism of its hyperplanes to coordinate axes. Let us call a hyperplane $h \in \mathcal{R}^d$ *k-parallel* ($0 \leq k \leq d-1$) if h is parallel to exactly k axes of $\mathcal{R}^d$, that is, to exactly k spanning edges of $\mathcal{C}^d$. 0-parallel hyperplanes are called *skew*. A trivial but useful observation is the following:

Observation 2.1 *$h = (v, b) \in \mathcal{H}(\mathcal{C}^d)$ is k-parallel iff exactly k entries of v vanish.*

For $d > 0$ and $0 \leq k < d$, let $e_k(d)$ denote the number of k-parallel hyperplanes in $\mathcal{H}(\mathcal{C}^d)$. It is possible to relate e_k to e_0 (the number of skew hyperplanes) in a simple way.

Observation 2.2 *For $d > 0$ and $0 \leq k < d$ we have $e_k(d) = \binom{d}{k} e_0(d-k)$.*

Proof. Let $h = (v, b) \in \mathcal{H}(\mathcal{C}^d)$ be k-parallel. Then k out of d entries of v are zero and there are $\binom{d}{k}$ possibilities to place them. The remaining entries of v define a vector of dimension $d-k$ without zero entries which obviously is the normal vector of a skew hyperplane in $\mathcal{H}(\mathcal{C}^{d-k})$. The number of such hyperplanes was defined to be $e_0(d-k)$. □

Note that we can thus restrict attention to skew hyperplanes, as each non-skew hyperplane has its skew analog in some dimension lower.

2.3 Symmetric Hyperplanes

Next we study in which ways a hyperplane $h \in \mathcal{H}(\mathcal{C}^d)$ may partition the vertex set V^d. To this end, we call h a *(j, p, k)-plane* if it contains $p \geq d$ vertices and splits the rest into subsets of cardinalities j and k where $j \leq k$.

One might conjecture that a valid triple (j, p, k) uniquely determines a hyperplane in $\mathcal{H}(\mathcal{C}^d)$, up to hypercube symmetries. This is false already in dimension 5 where hyperplanes with a fixed triple (j, p, k) may have a different intersection with the d-cube. For instance, the two hyperplanes $h = ((-3, 1, 1, 1, 1), 0)$ and $h' = ((-1, -3, 1, 2, 2), 0)$ in $\mathcal{H}(\mathcal{C}^5)$ are both (11,5,16)-planes. However, h cuts 10 edges of $\mathcal{C}^5$ in their interior while h' does this 13 times. Dimension 5 is the first where this phenomenon occurs which turns out to be frequent in higher dimensions. This already reveals part of the difficulty of classifying the hyperplanes in $\mathcal{H}(\mathcal{C}^d)$.

A *symmetric* hyperplane in $\mathcal{R}^d$ is one which passes through the center $(\frac{1}{2}, \ldots, \frac{1}{2})$ of $\mathcal{C}^d$, that is, a hyperplane $(v, b) \in \mathcal{R}^d$ with $\sum_{i=1}^{d} v_i = 2b$. For symmetric hyperplanes h, every two antipodal vertices $x, \overline{x} \in V^d$ either both lie in h or they lie on opposite sides of h. Symmetric (j,p,k)-planes thus have $j = k$, that is, the vertices of the d-cube which are avoided by the hyperplane are partitioned into equal-sized subsets.

There is a bijection between asymmetric skew (j,p,k)-planes in $\mathcal{H}(\mathcal{C}^d)$ and symmetric skew $(j+k, 2p, j+k)$-planes in $\mathcal{H}(\mathcal{C}^{d+1})$.

Theorem 2.1 *The number of asymmetric skew hyperplanes in $\mathcal{H}(\mathcal{C}^d)$ that contain exactly p vertices of $\mathcal{C}^d$ is equal to the number of symmetric skew hyperplanes in $\mathcal{H}(\mathcal{C}^{d+1})$ with exactly $2p$ vertices of $\mathcal{C}^{d+1}$.*

Proof. There is an obvious bijection between hyperplanes $(v, b) \in \mathcal{R}^d$ and symmetric hyperplanes $(w, b) \in \mathcal{R}^{d+1}$, given by $w_i = v_i$ for $i = 1, \ldots, d$ and $w_{d+1} = 2b - \sum_{i=1}^{d} v_i$. Note that, under this correspondence, (v, b) is asymmetric and skew if and only if (w, b) is symmetric and skew. Furthermore, it is easy to check that a vertex x of $\mathcal{C}^d$ lies in (v, b) if and only if the antipodal vertex pair $(x, 0)$ and $(\overline{x}, 1)$ of $\mathcal{C}^{d+1}$ lies in (w, b). Hence (v, b) is an asymmetric and skew hyperplane in $\mathcal{H}(\mathcal{C}^d)$ just if (w, b) is a symmetric and skew hyperplane in $\mathcal{H}(\mathcal{C}^{d+1})$. The number of vertices of $\mathcal{C}^{d+1}$ that lie in (w, b) is exactly twice the number of vertices of $\mathcal{C}^d$ that lie in (v, b). $\square$

The merit of Theorem 2.1 is that attention can be restricted to skew symmetric hyperplanes for further classification. Note also that the number of vertices covered by a symmetric hyperplane $h \in \mathcal{H}(\mathcal{C}^d)$ is at least $2(d-1)$.

2.4 Hull-Honest Hyperplanes

A completely different way to classify the hyperplanes in $\mathcal{H}(\mathcal{C}^d)$ is by considering their surface of intersection with $\mathcal{C}^d$. To this end, $h \in \mathcal{H}(\mathcal{C}^d)$ is called a *hull-honest* hyperplane if the convex hull of $h \cap V^d$ coincides with the polytope $h \cap \mathcal{C}^d$. Expressed in different terms, h cuts edges of the d-cube only at their vertices but not in their interior. In fact, in dimensions two (square) and three (cube) each spanned line (plane) is hull-honest. But starting with dimension four, spanned hyperplanes can properly intersect edges of the d-cube, and the convex hull of all the vertices lying on the hyperplane is no longer the surface of intersection of hyperplane and d-cube. Hull-honest hyperplanes seem to be the only type in $\mathcal{H}(\mathcal{C}^d)$ which has been studied previously, motivated by the intersection polytopes they generate [32, 29]. We now give an easy-to-use criterion to characterize hull-honest hyperplanes, and we also give the exact number of such hyperplanes, in general dimensions.

Theorem 2.2 *Let h be a hyperplane in $\mathcal{H}(\mathcal{C}^d)$ and let $v = (v_1, \ldots, v_d)$ be the (shortest integer) normal vector of h. Then h is hull-honest iff $v \in \{-1, 0, 1\}^d$.*

Proof. The assertion is trivial for $d = 1$, so let $d \geq 2$ below. Now assume that h is hull-honest. We first prove $v \in \{-1, 0, 1\}^d$ under the assumption that h contains the origin.

We claim that if v_i and v_j are non-zero entries of v of opposite sign then $|v_j| = |v_i|$. Suppose for contradiction that $|v_j| > |v_i|$. Then the adjacent vertices $x = (0, \ldots, 0_i, \ldots, 1_j, \ldots, 0)$ and $y = (0, \ldots, 1_i, \ldots, 1_j, \ldots, 0)$ lie on opposite sides of h and thus h intersects the edge between them.

Thus $|v_j| = |v_i|$ whenever v_i and v_j have opposite sign. Since not all non-zero entries of v have the same sign (since h contains the origin but also other vertices), we conclude that all non-zero entries of v must have the same absolute value A. But $v_1, \ldots, v_d$ have gcd 1, so we have $A = 1$, and $v \in \{-1, 0, 1\}^d$.

To cover hull-honest hyperplanes $h = (v, b)$ that avoid the origin, we choose some vertex x of $\mathcal{C}^d$ in h and consider $\mathcal{R}^d$ as being spanned by the d-cube edges emanating from x. This transformation effects only the signs of the entries of v, leaving their absolute values unchanged.

For the if part, let $h = (v, b)$ have $v \in \{-1, 0, 1\}^d$. Suppose h is not hull-honest. Then there is at least one edge of $\mathcal{C}^d$ which is intersected by h. The vertices of this edge, x and y, lie on opposite sides of h, hence $v \cdot x > b$ and $v \cdot y < b$. Since x, y, and v are all integer, we have $v \cdot x \geq b + 1$ and $v \cdot y \leq b - 1$ which implies $v \cdot x \geq v \cdot y + 2$. On the other hand, x and y have Hamming distance one. Say they differ at position i. This gives $|v \cdot x - v \cdot y| = |v_i|$, hence $|v_i| \geq 2$. This contradiction proves h hull-honest. □

For skew hyperplanes no entry of a normal vector is zero. Skew hull-honest hyperplanes therefore have always a normal vector with entries from $\{-1, 1\}$. We will now use this fact to derive their exact number in general dimensions. Intuitively speaking, we look at the 2^{d-1} main diagonales of the d-cube and show that there are precisely $d - 1$ hull-honest hyperplanes perpendicular to a diagonal.

Theorem 2.3 *Let v be a vector in $\{-1, 1\}^d$, $d \geq 2$, and b an integer. Then the hyperplane (v, b) is spanned by a set of vertices of $\mathcal{C}^d$ if and only if b satisfies $n(v) - d < b < n(v)$, where $n(v)$ is the number of 1's in v.*

Before proving the theorem, we use it to count the number of skew hull-honest hyperplanes in $\mathcal{H}(\mathcal{C}^d)$.

Corollary 2.1 *The number of skew hull-honest hyperplanes in $\mathcal{H}(\mathcal{C}^d)$ is given by*

$$h_0(d) = \begin{cases} 2 & d = 1, \\ (d-1)2^{d-1} & d \geq 2. \end{cases}$$

Proof. The case $d = 1$ is obvious, so let $d \geq 2$. By Theorems 2.2 and 2.3, we only need to count the number of distinct hyperplanes of the form (v, b) with $v \in \{-1, 1\}^d$ and $n(v) - d < b < n(v)$. For each $v \in \{-1, 1\}^d$, there are exactly $d - 1$ integers b within these bounds. Thus there are $2^d(d-1)$ pairs (v, b) satisfying the requirements. Each skew hull-honest hyperplane is counted twice as (v, b) and $(-v, -b)$ give the same hyperplane. □

Proof of Theorem 2.3. Note first that, if b lies outside the given range, then (v, b) contains at most one vertex of $\mathcal{C}^d$, and so the only if part of the theorem follows.

For the if part, define, for $v \in \{-1, 1\}^d$, the vertex $x(v)$ of $\mathcal{C}^d$ such that $x_i(v) = \frac{1}{2}(1 + v_i)$. Let $X(v, k)$ denote the set of all vertices of $\mathcal{C}^d$ that differ from $x(v)$ in exactly k coordinates, i.e., whose Hamming distance from $x(v)$ is exactly k. Observe that $X(v,k)$ is contained in the hyperplane $(v, n(v)-k)$. Furthermore, it is an easy exercise to show that for k between 1 and $d-1$, the vertices in $X(v,k)$ span the hyperplane. Thus for $n(v)-d<b<n(v)$ the hyperplane (v,b) is spanned by the vertices in $X(v, n(v)-b)$. □

Note that a skew and hull-honest hyperplanes $(v,b) \in \mathcal{H}(\mathcal{C}^d)$ thus contains precisely $\binom{d}{n(v)-b}$ vertices of $\mathcal{C}^d$. (In fact, the corresponding intersection figures turn out to be well-known polytopes [32] and to have interesting properties [29, 62].) Moreover, up to hypercube symmetries, there is a unique symmetric, skew, and hull-honest hyperplane in every even dimension and no such one in odd dimensions.

For $d \geq 1$, let $h(d)$ denote the total number of hull-honest hyperplanes in $\mathcal{H}(\mathcal{C}^d)$, and let $h_k(d)$ count those of them which are k-parallel $(0 \leq k \leq d-1)$.

Theorem 2.4

$$h_k(d) = \begin{cases} \binom{d}{k}(d-k-1)2^{d-k-1} & \textit{for } 0 \leq k \leq d-2, \\ 2d & \textit{for } k = d-1. \end{cases}$$

$$h(d) = \frac{3^d(2d-3)}{6} + 2d + \frac{1}{2}.$$

Proof. According to Observation 2.2 on the relation between skew and k-parallel hyperplanes we have $e_k(d) = \binom{d}{k} e_0(d-k)$. The proof of Observation 2.2 shows that hull-honesty is not effected by the construction of k-parallel hyperplanes as we only add zeros to the normal vector of the original skew hyperplane. Therefore the relation also applies to hull-honest hyperplanes: $h_k(d) = \binom{d}{k} h_0(d-k)$. By using the result of Corollary 2.1 for $h_0(d)$ we get the formula for $h_k(d)$ as claimed above. Finally, summing over all values of k gives the total number of hull-honest hyperplanes in $\mathcal{H}(\mathcal{C}^d)$,

$$h(d) \quad = \quad \sum_{k=0}^{d-2} \left[\binom{d}{k} (d-k-1) 2^{d-k-1} \right] + \underbrace{2d}_{k=d-1} .$$

This simplifies to $h(d) = \frac{3^d(2d-3)}{6} + 2d + \frac{1}{2}$ by routine manipulation using the binomial theorem. □

2.5 Exhaustive Enumeration of Hyperplanes

To get an idea of the behaviour of the numbers and types of hyperplanes in $\mathcal{H}(\mathcal{C}^d)$ for dimensions $d \geq 4$ we did some numerical investigations which are briefly reported below. For a more complete description and discussion see Aichholzer [2].

In order to make the enumeration of $\mathcal{H}(\mathcal{C}^d)$ efficient we considered the following question: For a hyperplane $(v,b) \in \mathcal{H}(\mathcal{C}^d)$, how large can the magnitude of the entries of v be? (Recall that we assumed v to be integer and as short as possible.) A similar problem was considered

Dimension d	1	2	3	4	5	6	7	8	9	10
$2 \cdot \left(\frac{d+1}{4}\right)^{\frac{d+1}{2}}$	1.00	1.30	2.00	3.49	6.75	14.18	32.00	76.89	195.3	521.6
max $\lvert\det(\{0,1\}^{d\times d})\rvert$	1	1	2	3	5	9	32	56	144	320
enumerative max. vector entry	1	1	1	2	3	5	9	18	42	≥ 98 ≤ 144

Table 2.1: Upper bound of Theorem 2.5, the determinant of $(0,1)$-matrices, and the size of normal vectors.

in the threshold circuit literature many years ago, and the upper bound $d^{d/2}$ is well known. Actually, the bound there is slightly weaker, $(d+1)^{(d+1)/2}$, since it matters from which side the determining points of a threshold hyperplane come from.

We found the improved bound below which, as we have recently learned, was already given by Muroga et al. [77] and can be found in more detail in the book of Muroga [76]. Actually, this improvement was rediscovered several times (cf. the survey papers of Orponen [82] and Parberry [84]), most recently by Håstad [53].

Fact 2.1 *Let A be a $(0,1)$-matrix of size $d \times d$. Then $|\det(A)| \leq 2\left(\frac{d+1}{4}\right)^{\frac{d+1}{2}}$ which is tight for $d = 2^k - 1$, $k \in \mathcal{N}$.*

Theorem 2.5 *Let $h = (v, b) \in \mathcal{H}(\mathcal{C}^d)$ and $v = (v_1, \dots, v_d)$.*

$$|v_i| \leq 2^{-(d-1)} d^{\frac{d}{2}} \quad \textit{for } i = 1, \dots, d,$$

$$|b| \leq 2^{-d}(d+1)^{\frac{d+1}{2}}.$$

For threshold circuits the bound for the weights v_i and the threshold level b is the same, whereas we can bound v_i stronger than b, for the reason pointed out above.

The bound in Fact 2.1 is achieved via a transformation from Hadamard matrices of dimension $d+1$. A *Hadamard matrix M* is an $n \times n$ matrix with entries from $\{-1, 1\}$ which satisfies $MM^T = M^T M = nI_n$. In [50] it is shown that there always exists a Hadamard matrix when $n = 2^k$, and its existence is conjectured for every $n = 4k$. (This conjecture has been verified for $n < 428$; see [33, 50]). Depending on $d \bmod 4$, some better bounds on the determinant of $(0,1)$-matrices of order d and therefore for Theorem 2.5 are known; see [33] and references therein.

The bound on v in Theorem 2.5 is not tight for dimensions $d = 2^k$, as one might conjecture from Fact 2.1. We can give a counterexample for dimension 8: The bound on order 7 matrix determinants is 32 and has to be tight by Fact 2.1. This is also the bound on normal vector entries in dimension 8 that results from Theorem 2.5. But by exhaustive enumerative computation of $\mathcal{H}(\mathcal{C}^8)$ the largest normal vector entry was 18. This is somewhat surprising because testing all those hyperplanes also means testing all linear systems of order 7 with coefficients in $\{0, 1\}$ and with unique solutions. By Cramer's rule each solution coefficient is the ratio of

Dimension d	Number of hyperplanes in $\mathcal{H}(\mathcal{C}^d)$	Number of skew hyperplanes $e_0(d)$	Number of different types of skew hyperplanes	Number of skew hull-honest hyperplanes $h_0(d)$
1	2	2	1	2
2	6	2	1	2
3	20	8	1	8
4	140	88	3	24
5	3254	2704	8	64
6	252434	234688	35	160
7	71343208	69640192	219	384
8	86246755608	85682904704	1293	896
9	?	?	?	2048
10	?	?	?	4608

Table 2.2: Numbers of different kinds of hyperplanes in the d-cube.

two integers, each having an absolute value ≤ 18 (and not 32, to which the corresponding determinants could increase). Given the importance of linear systems with coefficients in $\{0,1\}$ in many fields, the gap between maximal solution coefficients and maximal determinant values is worth mentioning. See [33] for some related work in the context of determinant functions.

Table 2.1 displays the upper bound of Theorem 2.5 versus the determinant of $(0,1)$-matrices and the size of normal vectors, respectively, as far as we computed them.

Using the above bounds we computed all spanned hyperplanes in dimensions up to 8. Their numbers turned out to increase superexponentially with d (roughly 2^{d^2}), as one would expect from the trivial upper bound $\binom{2^d}{d}$. (Each d-tuple chosen from the 2^d vertices of $\mathcal{C}^d$ potentially gives rise to a hyperplane.) An exhaustive enumeration for dimensions higher than 8 currently seems to be out of reach. For $\mathcal{H}(\mathcal{C}^8)$ our program, though exploiting theoretical results reported in the present chapter and in [2], needed about 12 days on a DEC 5000/240, and for $\mathcal{H}(\mathcal{C}^9)$ we estimate a running time of about 35 years.

The experimental results, on the one hand, led us to observe structural properties of $\mathcal{H}(\mathcal{C}^d)$ which could be made rigorous, but on the other showed us the intrinsic difficulty of classifying d-cube hyperplanes. In each further dimension we encountered new types of hyperplanes which could not be brought into connection with hyperplanes in lower dimensions. Although we only needed to deal with skew symmetric hyperplanes this did not really help, since the skew type prevails over all other types, and symmetry gains only one dimension. For instance, in $\mathcal{H}(\mathcal{C}^8)$ about 99.3 percent of all hyperplanes are skew.

Most hyperplanes are spanned by very few vertices. In $\mathcal{H}(\mathcal{C}^8)$ about 30 percent cover only the minimum of 8 vertices. A correlation between the number of vertices covered by a hyperplane, its degree of parallelism, and the size of its normal vector can be observed: The smaller the normal vector and the higher the degree of parallelism, the more vertices may be covered. Saks [92] showed that the maximum number of vertices covered by a skew hyperplane is $\binom{d}{\lfloor \frac{d}{2} \rfloor}$. Section 2.4 on hull-honest hyperplanes tells us that this bound is attainable. It is not difficult to deduce that a k-parallel hyperplane thus can cover not more than $2^k\binom{d-k}{\lfloor\frac{d-k}{2}\rfloor}$ vertices. Note that $(d-1)$-parallel hyperplanes cover exactly half of the 2^d vertices of $\mathcal{C}^d$, the maximum possible.

Some detailed enumerative results are listed in Table 2.2.

Chapter 3

0/1-Polytopes

3.1 Introduction

The hypercube is, beside the simplex and the cross polytope, the only platonic solid which exists in any dimensions. The theory of convex polytopes provides classical results concerning sections and projections of hypercubes (see e.g. [32, 48]) as well as for so-called subpolytopes of the d-cube [114]. In this chapter we consider the convex hull of a subset $V \subseteq \{0,1\}^d$ of hypercube vertices. A polytope P representing the convex hull of V is called a *0/1-polytope*. Thus the faces of 0/1-polytopes are supported by the hyperplanes of the set $\mathcal{H}(\mathcal{C}^d)$, investigated in Chapter 2.

Motivation for considering 0/1-polytopes comes from Combinatorial Optimization (e.g. the Travelling Salesman Polytope, or the Assignment Polytope) and many other fields, see e.g. Ziegler [116, 117].

A point $p \in P$ is an *extreme point* of P provided $x, y \in P$, $0 < \lambda < 1$, and $p = \lambda x + (1-\lambda)y$ imply $p = x = y$. In other words, p is an extreme point of P if it does not belong to the relative interior of any segment contained in the closure of P. It is not hard to see that any vertex of V is an extreme point of P: consider the d-dimensional hypersphere S^d with center $(\frac{1}{2}, \dots, \frac{1}{2})$ and radius $\frac{\sqrt{d}}{2}$. All vertices of the d-cube $\mathcal{C}^d$ lie on S^d and thus any subset V of vertices entirely consists of extreme vertices. Thus a 0/1-polytope P is a so-called subpolytope of the hypercube.

The main result we obtain in this chapter is to determine the maximum number of facets of a five dimensional 0/1-polytope. We show that this maximum is 40, answering an open question of Ziegler [117]. We do this by a complete enumeration of all equivalence classes of 0/1-polytopes of dimension 5. Moreover we show that the degree of a vertex of the convex hull can exceed 2^{d-1} already for $d \geq 5$. Numerical experiments also indicate that the number of edges of a 0/1-polytope depends quadratically on the number of vertices already for moderately large dimensio.

3.2 Facets of 0/1-Polytopes

Figure 3.1 shows all types of three-dimensional 0/1-polytopes. Given is the number of vertices, edges, facets, and the number of equivalent copies, respectively. There are 12 different types of

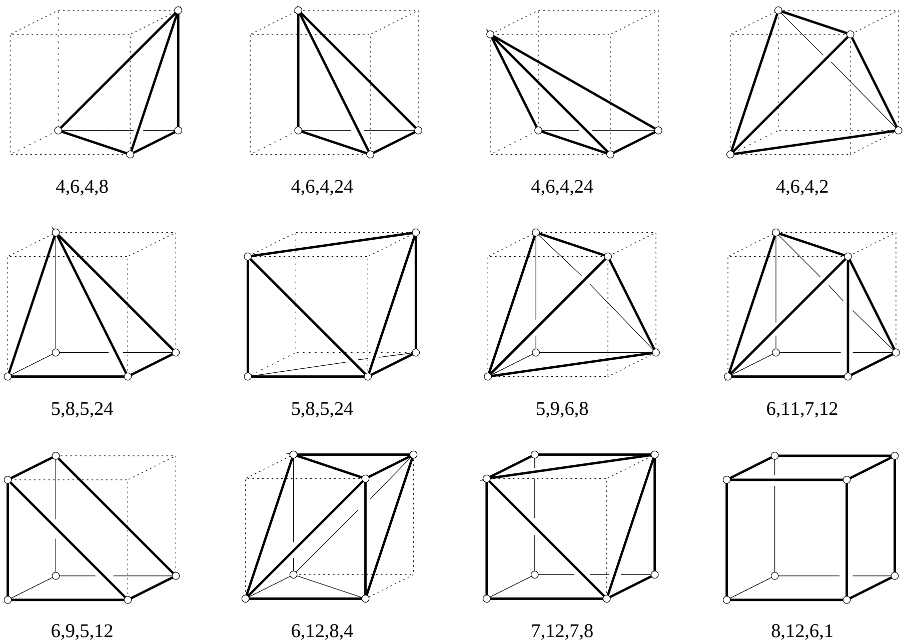


Figure 3.1: Three dimensional 0/1-polytopes.

0/1-polytopes, each representing an equivalence class of the given cardinality (for a definition of equivalence classes see below). In total this gives 151 0/1-polytopes in three-space. The remaining 105 possible subsets of vertices of the 3-cube span polytopes of dimension at most two.

For arbitrary dimension d there are $2^{(2^d)}$ possible vertex subsets of the d-cube. Most of them span a d-dimensional 0/1-polytope. In fact, Ziegler et. al. have shown recently that there are at least $2^{(2^{d-2})}$ different combinatorial types of d-dimensional 0/1-polytopes for $d > 5$, see Problem 2.6 in [115]. For $d \leq 4$ a complete enumeration of all sets can simply be done by using standard software. But starting with dimension 5 the number of possible sets increases tremendously. For $d = 6$ there are already about $1.8 \cdot 10^{19}$ possible sets. Thus dimension 5 seems to be the first non-trivial and last feasible dimension for enumerating all such sets. In this work we give for the first time a complete enumeration of all non-equivalent 0/1-polytopes for dimension 5. To this end we use some results of Chapter 2 and several observations given in the next section.

3.3 Complete Enumeration for Dimension 5

A straight forward method to enumerate all subsets of vertices of $\mathcal{C}^5$ would be impractical because of the huge number of $2^{(2^5)} \approx 4.3\ 10^9$ such sets. E.g. if we could compute about one hundred 5d-convex hulls per second, it would last for over one year to enumerate all sets. Fortunately many of these sets are equivalent, i.e. are reflections and rotations of each other. Thus for every equivalence class we need only to consider one representative.

We therefore divide the enumeration into two parts. First we generate exactly one representative vertex set for every equivalence class. We then can easily perform various investigations of these sets in reasonable time, e.g. hunting for the 0/1-polytope with the maximal number of facets.

3.3.1 Generating Representatives of Equivalence Classes

Let us first consider the equivalence relations. A *reflection* of a vertex set V of the d-cube for an axis i exchanges all zeros and ones on the ith bit position of all vertices of V. Reflections at k axes are thus reflections at a $(d-k)$-subspace. By combining reflections for a certain set of axes we can thus transform a given vertex of V to an arbitrary vertex of the hypercube. For example one can "move" any arbitrary vertex of the set to the origin. Clearly reflections do only change the orientation of the vertex set, but not the properties of their convex hull.

The second operation is called *rotation*. Here we allow arbitrary permutations of the coordinate axes. This operation reflects the fact that we do not weight the bits of a vertex according to their position, i.e. all directions in d-space have the same significance. Again, rotations might affect the absolute position of the vertices and the orientation of the set, but the structure of their convex hull remains unchanged.

A *class of vertex sets* contains all vertex sets which can be transformed into each other by repeated application of the above two operations. Obviously all vertex sets within a class lead to equivalent convex hulls and we need only to consider one set per class.

Each vertex of the d-cube can be associated, in a natural way, with its *value*, which is just the decimal value of the binary string corresponding to the vector coordinates. The *vertex vector* $b = (b_0, b_1, \ldots, b_{2^d-1})$ of a set V of vertices from the d-cube is a binary vector of length 2^d with $b_i = 1$ if and only if the vertex with value i is an element of V. For example the vertex vector of the set $\{(0,0,0), (0,1,1), (1,0,1)\}$ is $b = (1,0,0,1,0,1,0,0)$. Clearly, reflections and rotations of a vertex set change its vertex vector.

For any class of vertex sets we can select the set with the lexicographically maximal vertex vector as its representative. In the following we will show that these sets have properties which can be utilized to generate them.

Consider a vertex set V with n vertices of the d-cube and suppose that the vertices are in increasing lexicographical order. We then build a binary $(n \times d)$-matrix $A(V) = (v_1, \ldots, v_n)^T$ with the vertices of V as its rows. A *block* of $A(V)$ is a maximal set of indices of consecutive identical column vectors. This means that permuting the columns within a block does not change the matrix $A(V)$ and thus does not affect the vertices of V.

A vertex $v_i \in V$ has the *block property* if for any block b built by $A(v_1, \ldots, v_{i-1})$ and for any pair $k < l$ of indices within b we have $v_i(k) \leq v_i(l)$, where $v_i(k)$ denotes the kth bit of v_i. In other words, within a block built by the previous vertices, v_i always has the form $(0, 0, \ldots, 0, 1, 1, \ldots, 1)$, i.e. "all ones are shifted to the right".

A set V of vertices fulfills the block property if every vertex of V has the block property. If this is the case it follows by induction that identical column vectors of $A(V)$ are consecutive (recall that the vertices in V have to be in increasing lexicographical order).

Let us give an example for C^3:

$$\begin{aligned}
v_1 &= (0,0,0,0,0,0|1,1) \\
v_2 &= (0,0|1,1,1,1|0,0) \\
v_3 &= (1,1|0,0,0|1|1,1) \\
v_4 &= (0|1|0|1,1|0|0|1)
\end{aligned}$$

$A(v_1, v_2, v_3)$ has blocks of length 2,3,1 and 2, respectively. Thus the additional vertex v_4 fulfills the block property. Note that this also holds for v_1, v_2 and v_3, such that the whole vertex set $V = \{v_1, v_2, v_3, v_4\}$ has the block property.

Lemma 3.1 *In a class of equivalent vertex sets, a maximal vertex vector is attained by a vertex set which (a) contains the origin and (b) has the block property.*

Proof. We have to show that any vertex set $V = \{v_1, \ldots, v_n\}$ which does not have properties (a) and (b) can be transformed into a equivalent set V' with a (lexicographically) larger vertex vector, fulfilling (a) and (b).

First recall that if the set does not include the origin then by reflection we can move an arbitrary vertex of the set to the origin. This clearly increases the vertex vector. Now consider the vertices in increasing order. Suppose that a vertex v_i, $2 \leq i \leq n$, does not have the block property according to the vertices $v_1, \ldots, v_{i-1}$. By permutation of the positions within the blocks we can rearrange the bits of v_i such that it fulfills the block property. This reduces the value of v_i and does clearly not affect all vertices $v_1, \ldots, v_{i-1}$. Note that although the permutations might change $v_{i+1}, \ldots, v_n$, too, the reduction of the value of v_i still dominates the lexicographic order. Thus v_i does not only have the block property, but we also increased the vertex vector of the set.

By induction we conclude that a vertex set attaining the maximal vertex vector has to fulfill properties (a) and (b). $\square$

k	$\binom{2^5}{k}$	generated sets	equivalence classes	class cardinality min. / average / max.		
0	1	1	1	1	1.0	1
1	32	1	1	32	32.0	32
2	496	5	5	16	99.2	160
3	4960	30	10	160	496.0	960
4	35960	230	47	40	765.1	1920
5	201376	1821	131	32	1537.2	3840
6	906192	12789	472	32	1919.9	3840
7	3365856	73880	1326	32	2538.4	3840
8	10518300	344054	3779	20	2783.4	3840
9	28048800	1300574	9013	160	3112.0	3840
10	64512240	4053531	19963	16	3231.6	3840
11	129024480	10582906	38073	32	3388.9	3840
12	225792840	23453731	65664	16	3438.6	3840
13	347373600	44578853	98804	160	3515.8	3840
14	471435600	73233239	133576	80	3529.3	3840
15	565722720	104566543	158658	32	3565.7	3840
16	601080390	130282523	169112	2	3554.3	3840

Table 3.1: Number and cardinality of equivalence classes of vertex sets of the 5-cube.

The algorithm to generate exactly one representative vertex set per equivalence class has now two stages.

In the first part we generate only vertices which fulfill properties (a) and (b) of Lemma 3.1. We use a set of nested for-loops with fixed first element v_0. For every vertex v_i, $0 \leq i \leq 2^5 - 1$, we precomputed two bit masks. The first mask has a '1' at every bit position where v_i forces a new block to start, i.e. for the second bit of every '01' or '10' combination of v_i. When adding a new vertex to a set we can therefore update the block information by a simple constant time binary OR operation (assuming constant dimension d). The second mask has a '1' for the second bit of every '10' pair of v_i. In order to ensure that an additional vertex v_i has the block property there must start a new block of the previous vertex set for every '1' in this mask. This means that for every '1' in the second mask of v_i there must be a '1' in the block mask of the previous vertices, which is just the OR-combination of their first masks. For constant dimension we can thus decide in constant time whether a new vertex fulfills the block property, i.e. whether it has to be considered. An example of a vertex (for dimension 10) and its two bit masks is given below.

$$\begin{aligned} v_i = &\quad (0,0,1,1,0,0,1,0,1,1) \\ mask_1 = &\quad (0,0,1,0,1,0,1,1,1,0) \\ mask_2 = &\quad (0,0,0,0,1,0,0,1,0,0) \end{aligned}$$

With these two masks we can generate all vertex sets with properties (a) and (b) of Lemma 3.1 very efficiently.

For the second part of our algorithm we apply all possible reflections and rotations to the generated sets. For dimension 5 this gives up to $2^5 \cdot 5! = 3840$ possible copies per set. A vertex set is stored only if the corresponding vertex vector yields the maximum among all resulting vertex vectors. In this way we guarantee that for every equivalence class we store one unique representative.

Table 3.1 lists the results for $k = 0 \ldots 16$ vertices of $\mathcal{C}^5$. Sets with $k > 16$ vertices can be seen as the complement of sets with $32 - k$ vertices and can easily be obtained from them. The first three columns give the number of all vertex sets, the number of generated sets (part one of the above description) and the number of equivalence classes, respectively. It can be seen that less than $\frac{1}{6}$ of all vertex sets with ≤ 16 vertices fulfill properties (a) and (b) of Lemma 3.1. Thus we needed to apply the more time consuming rotations and reflections only to this smaller set. The third column shows that there are less than 700000 equivalence classes for $k \leq 16$ and about 525000 for $k > 16$. Therefore the subsequent investigations have to enumerate about 1.2 million sets, which is less than 0.03% of the initial number of sets. This enables us to consider various related questions within a couple of hours instead of month or years.

Rote [91] mentioned that the number of equivalence classes can also be computed by Pólya's theory of counting [86, 25]. Based on Burnside's lemma [28] the cycle index of the permutation group is built, which then, by Pólya's theorem, gives the number of equivalence classes, see e.g. [61, 68] for more details. The results obtained from this theoretical approach coincide with the output of our program. Note, however, that with Pólya's counting theorem it is not possible to obtain a representative vertex set for each equivalence class.

The last three columns of Table 3.1 show the minimum, average and maximal cardinality of the equivalence classes. There are highly symmetric vertex sets, e.g. for $k = 16$ there is a class with only two sets: The first consists of all vertices with an even number of ones and the second is its complement. On the other hand there are classes with the maximal number of 3840 different sets. On average a class contains over 3000 sets, which results in the comparatively small number of classes.

We used 4 bytes to store the 32-bit vertex vector of every representative vertex set and one additional byte to store the number of vertices, which is clearly redundant but useful for fast access. This results in about 3.5 MB of data, which can easily be handled by any standard computer equipment. In fact we used a simple Personal Computer to compute these results. In a second file we also stored the cardinality of each class (about 1.4 MB). To enable the interested reader to carry out her or his own investigations the files are available upon request from the author.

3.3.2 Counting Facets and Edges

The clear advantage of our approach is that we need to precompute the set of all representative vertex sets only once. Any function which requires a complete enumeration of all vertex sets of the 5-cube can then be computed easily within a few hours on a standard computer equipment. In this subsection we use this to search for 0/1-polytopes in dimension 5 which maximize the number of facets, and to consider the number of edges of their convex hulls. In other words we count the 1-dimensional and $(d-1)$-dimensional faces of the convex hulls of subsets of the vertices of $\mathcal{C}^5$. Using the generated vertex sets, investigations of other faces would of course also be possible.

Motivation for counting the number of facets of 0/1-polytopes is given by the papers of Ziegler [116, 117], Henk et.al. [54] and Kortenkamp et. al. [65]. We follow the notation of Ziegler who denotes the maximal number of facets of a d-dimensional 0/1-polytope with $f(d)$. It is well known and easy to check that $f(d) = 2^d$ for $d \leq 4$. For dimension 5 a 0/1-polytope with 40 facets is known, i.e. $f(5) \geq 40$, but it was unknown whether this bound is tight. The general question is how fast $f(d)$ grows with d. In particular it is unknown if there exists a constant c such that $f(d) < c^d$ for all d [116, 117]. The latest results in the race for 0/1-polytopes with many facets of up to dimension 14 can be obtained from the homepage of Kortenkamp [64].

To determine $f(5)$ we made a complete enumeration of the convex hulls of all representative vertex sets generated in the last subsection. We used two different methods to compute the number of facets of each set. First we ran a simple implementation of the incremental method to compute higher dimensional convex hulls for point sets in $\mathcal{R}^d$. The algorithms worked well and within several hours we could enumerate all classes.

Note that when using a standard convex hull algorithm one has to be careful, since vertices of $\mathcal{C}^d$ are highly degenerate in the sense that there are many linear dependences between them. This may cause numerical troubles in computing the convex hull and in fact many of the existing software packages fail on this task (compare e.g. [26]). For example perturbation methods may result in superpolynomially sized convex hulls for the d-cube [13].

For the second approach we used a tailor-made algorithm for C^5. We utilized the fact, that we have exactly 3254 hyperplanes spanned by vertices of the 5-cube, see Chapter 2. This means that for 0/1-polytopes there are 3254 possible ways to support their facets. For every hyperplane we precomputed three vertex vectors. The first two contain all vertices of C^5

k	full-dim. vertex sets	full-dim. classes	number of facets min. / average / max.		
6	556192	237	6	6.00	6
7	2925056	1062	7	8.35	12
8	10071580	3462	7	11.06	20
9	27678880	8781	7	13.84	26
10	64264048	19767	7	16.41	32
11	128892000	37976	8	18.56	34
12	225738120	65600	7	20.15	40
13	347356800	98786	8	21.16	38
14	471432000	133565	8	21.67	38
15	565722240	158656	8	21.80	36
16	601080360	169110	8	21.64	35
17	565722720	158658	9	21.30	34
18	471435600	133576	8	20.82	31
19	347373600	98804	10	20.24	30
20	225792840	65664	9	19.57	28
21	129024480	38073	10	18.83	27
22	64512240	19963	10	18.03	26
23	28048800	9013	10	17.19	22
24	10518300	3779	9	16.35	20
25	3365856	1326	12	15.52	18
26	906192	472	11	14.74	16
27	201376	131	12	13.99	15
28	35960	47	11	13.28	14
29	4960	10	12	12.58	13
30	496	5	11	11.84	12
31	32	1	11	11.00	11
32	1	1	10	10.00	10

Table 3.2: 5-cube: number of full-dimensional vertex sets, equivalence classes and number of facets of 0/1-polytopes with k vertices.

which are in or above (in or below, respectively) the hyperplane. The third set contains all vertices covered by the hyperplane. Using the first two vertex vectors we can by simple bit operations decide in constant time, whether a hyperplane supports a given set V of vertices or not. Using the third vertex vector we can identify all vertices of V which are covered by the hyperplane. Solving a simple linear system of equations shows us whether these vertices span the hyperplane. If and only if this is the case the hyperplane counts for one facet. Reiterating this for all 3254 possible hyperplanes gives the number of facets of the convex hull of V.

It turned out that the second approach worked quite well. On a standard Personal Computer (166 MHz Pentium© processor) we needed less than one hour to examine all classes of up to 16 vertices, and an additional hour for the complementary sets.

Table 3.2 summarizes these experimental results. The first column gives the number k of

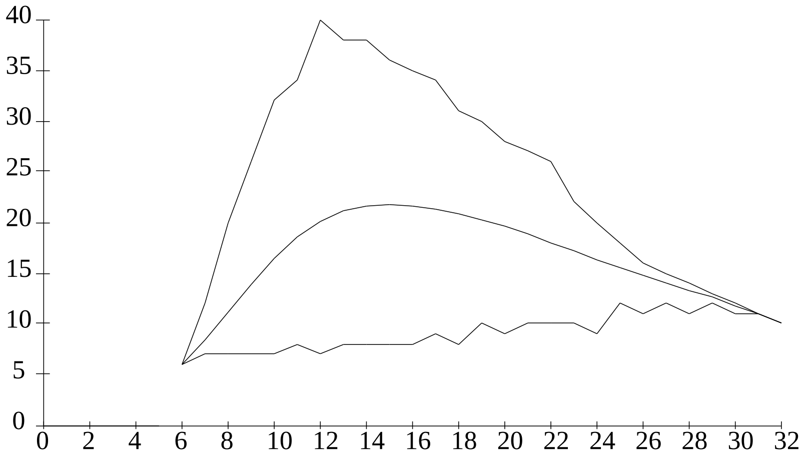


Figure 3.2: Minimal, average and maximal number of facets of 5-dimensional 0/1-polytopes over the number of vertices.

vertices and the next two columns give the number of vertex sets and the number of classes, respectively, which provide a full-dimensional convex hull. In other words this is the number of sets (classes) where not all vertices lie in a hyperplane or lower dimensional subspace. Clearly, for counting the number of facets or edges we only took these sets into account.

Columns 4 to 6 give the minimal, average and maximal number of facets for a fixed number of vertices. Figure 3.2 shows the corresponding curves. Note that the average is taken over the fulldimensional vertex sets, rather than over the classes. The table shows that the average number of facets is maximized for $k = 15$, whereas the best single example has $k = 12$ vertices. This might also be true for higher dimensions and thus it has to be taken into account when looking for "maximizing" sets by investigation of random vertex sets. The most promising vertex sets for maximizing the number of facets might have significantly less vertices than the sets with the maximal average number of vertices.

$$
\begin{array}{lll}
v_1 = (0,0,0,0,0) & v_5 = (0,1,1,0,1) & v_9 = (1,0,1,1,1) \\
v_2 = (0,0,0,0,1) & v_6 = (0,1,1,1,1) & v_{10} = (1,1,0,0,1) \\
v_3 = (0,0,1,1,0) & v_7 = (1,0,0,1,1) & v_{11} = (1,1,0,1,0) \\
v_4 = (0,1,0,1,0) & v_8 = (1,0,1,0,0) & v_{12} = (1,1,1,0,0)
\end{array}
$$

Table 3.3: The 12 vertices of the 5-dimensional 0/1-polytope with the maximal number of 40 facets.

Our results show that there exists only one unique vertex set of $\mathcal{C}^5$ (up to hypercube symmetries) with 12 vertices and 40 facets and that this is indeed the maximal number of facets for dimension 5.

Lemma 3.2 *The maximal number of facets of a 5-dimensional 0/1-polytope is $f(5) = 40$. This number is achieved by the vertex set given in Table 3.3 which is unique up to hypercube symmetries.*

normal vector of facet	covered vertices	normal vector of facet	covered vertices
f_1 = ((-1,0,0,0,0),0)	$v_1, v_2, v_3, v_4, v_5, v_6$	f_{21} = ((0,1,0,0,0),1)	$v_4, v_5, v_6, v_{10}, v_{11}, v_{12}$
f_2 = ((-1,-1,-1,1,0),0)	v_1, v_2, v_3, v_4, v_7	f_{22} = ((-1,1,-2,1,2),2)	$v_2, v_4, v_6, v_7, v_{10}$
f_3 = ((0,0,1,0,0),1)	$v_3, v_5, v_6, v_8, v_9, v_{12}$	f_{23} = ((-1,1,-1,0,1),1)	$v_2, v_4, v_5, v_6, v_{10}$
f_4 = ((1,0,0,0,0),1)	$v_7, v_8, v_9, v_{10}, v_{11}, v_{12}$	f_{24} = ((0,-1,-1,1,-1),0)	$v_1, v_3, v_4, v_7, v_{11}$
f_5 = ((1,-1,0,-1,1),1)	$v_2, v_7, v_8, v_9, v_{10}$	f_{25} = ((0,0,0,0,-1),0)	$v_1, v_3, v_4, v_8, v_{11}, v_{12}$
f_6 = ((-2,-1,-1,2,1),1)	v_2, v_3, v_4, v_6, v_7	f_{26} = ((1,-2,-1,1,-2),0)	$v_1, v_3, v_7, v_8, v_{11}$
f_7 = ((-1,-1,1,-1,0),0)	v_1, v_2, v_3, v_5, v_8	f_{27} = ((2,-1,-2,-1,-1),0)	$v_1, v_7, v_8, v_{10}, v_{11}$
f_8 = ((0,-1,0,0,0),0)	$v_1, v_2, v_3, v_7, v_8, v_9$	f_{28} = ((1,-1,0,1,-1),1)	$v_3, v_7, v_8, v_9, v_{11}$
f_9 = ((1,1,0,1,1),3)	$v_6, v_7, v_9, v_{10}, v_{11}$	f_{29} = ((-2,1,1,-1,-2),0)	$v_1, v_3, v_4, v_5, v_{12}$
f_{10} = ((1,-1,1,-2,2),2)	$v_2, v_5, v_8, v_9, v_{10}$	f_{30} = ((-1,0,1,-1,-1),0)	$v_1, v_3, v_5, v_8, v_{12}$
f_{11} = ((0,1,-1,1,1),2)	$v_4, v_6, v_7, v_{10}, v_{11}$	f_{31} = ((-1,2,-1,-2,-1),0)	$v_1, v_4, v_5, v_{10}, v_{12}$
f_{12} = ((-1,-1,1,0,1),1)	v_2, v_3, v_5, v_6, v_9	f_{32} = ((0,1,-1,-1,-1),0)	$v_1, v_4, v_{10}, v_{11}, v_{12}$
f_{13} = ((-1,-2,2,-1,1),1)	v_2, v_3, v_5, v_8, v_9	f_{33} = ((1,0,-1,-1,-1),0)	$v_1, v_8, v_{10}, v_{11}, v_{12}$
f_{14} = ((-1,-1,0,1,1),1)	v_2, v_3, v_6, v_7, v_9	f_{34} = ((-1,1,1,0,-1),1)	$v_3, v_4, v_5, v_6, v_{12}$
f_{15} = ((0,0,0,0,1),1)	$v_2, v_5, v_6, v_7, v_9, v_{10}$	f_{35} = ((0,1,1,1,-1),2)	$v_3, v_4, v_6, v_{11}, v_{12}$
f_{16} = ((0,0,0,1,0),1)	$v_3, v_4, v_6, v_7, v_9, v_{11}$	f_{36} = ((1,1,2,2,-1),4)	$v_3, v_6, v_9, v_{11}, v_{12}$
f_{17} = ((-1,1,-1,-1,0),0)	$v_1, v_2, v_4, v_5, v_{10}$	f_{37} = ((1,0,1,1,-1),2)	$v_3, v_8, v_9, v_{11}, v_{12}$
f_{18} = ((0,0,-1,0,0),0)	$v_1, v_2, v_4, v_7, v_{10}, v_{11}$	f_{38} = ((1,1,1,0,1),3)	$v_5, v_6, v_9, v_{10}, v_{12}$
f_{19} = ((0,0,0,-1,0),0)	$v_1, v_2, v_5, v_8, v_{10}, v_{12}$	f_{39} = ((1,0,1,-1,1),2)	$v_5, v_8, v_9, v_{10}, v_{12}$
f_{20} = ((1,-1,-1,-1,0),0)	$v_1, v_2, v_7, v_8, v_{10}$	f_{40} = ((2,2,1,1,1),5)	$v_6, v_9, v_{10}, v_{11}, v_{12}$

Table 3.4: The 40 facets of the 0/1-polytope given in Table 3.3.

It is interesting to observe that the equivalence class of the "best" example given in Table 3.3 contains 192 vertex sets. All other sets with 12 vertices have ≤ 36 facets.

Table 3.4 lists the 40 hyperplanes which support the facets of this 0/1-polytope. The right column gives in addition the number of vertices of each facet. Thus it can be seen that 10 facets cover 6 vertices and all remaining facets cover 5 vertices, i.e. are 4-dimensional simplices.

Let us now consider the number of edges of 0/1-polytopes. Motivation stems from graph-based algorithms which use the fact that every vertex of a subset of hypercube vertices is extreme, i.e. lies on the convex hull. Thus for many problems local optimization yields a global optimum. Clearly the number of edges and the degree of the vertices will effect the efficiency of the algorithms.

To compute the number of edges of the convex hull of a vertex set we again used both, a standard technique and a tailor-made algorithm. The later version is based on Observation 3.1 below. A subcube $\mathcal{C}^k$, $k \leq d$, is *spanned* by two vertices v_i and v_j of the d-cube, if it is the lowest dimensional subcube which contains v_i and v_j. Note that $\mathcal{C}^k$ need not include the origin. Clearly $\mathcal{C}^k$ reflects the directions where c_i and c_j have different bits, i.e. k is the Hamming distance between v_i and v_j, and the edge $e = v_i v_j$ is a diagonal of $\mathcal{C}^k$.

Observation 3.1 *Let V be a vertex set of the d-cube. An edge $e = v_i v_j$ is part of the convex hull of V iff e is part of the convex hull of the subcube spanned by v_i and v_j.*

The correctness of the observation can be seen from the fact that the k-dimensional (part of the) convex hull of a k-dimensional point set does not change when we add points in dimension $k+1$ only on one side of these set.

Let V be a subset of vertices of the d-cube. From Observation 3.1 it follows that we can compute the edges of the convex hull of V by inserting all edges spanned by vertices of V in order of non-decreasing length. For every edge e we consider the subcube spanned by its two endpoints. If a subcube has more than one diagonal then non of them can belong to the convex hull. Else we have to check whether the diagonal e is part of it. To this end consider the set E of edges emanating from one endpoint of e. Note that all these edges are shorter than e and therefore we already know that they are part of the convex hull. Then e belongs to the convex hull iff e is no positive linear combination of edges from E. Thus we have to solve a corresponding linear system for E and e is part of the convex hull iff the system has no feasible solution.

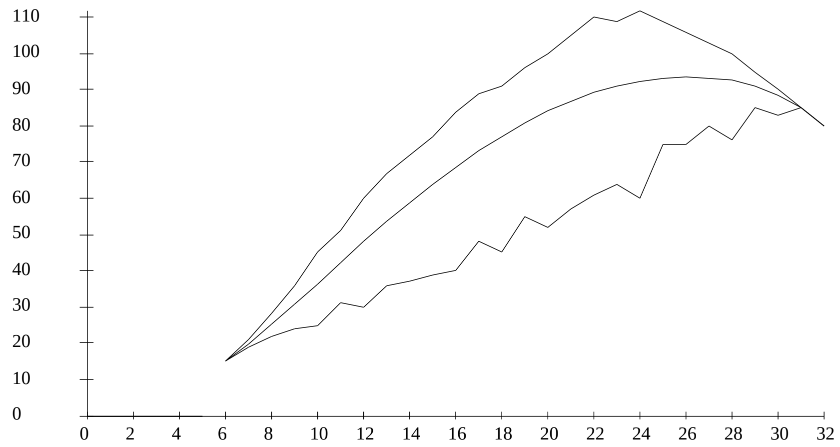


Figure 3.3: Minimal, average and maximal number of edges of 5-dimensional 0/1-polytopes over the number of vertices.

Using this approach we computed the number of edges and the maximal vertex degree of all 0/1-polytopes of dimension 5. A lower bound for the maximal number of edges is given by $d2^d$, the number of edges of the d-cube. Table 3.5 and Figure 3.3 show the minimal, average and maximal number of edges of 0/1-polytope of dimension 5. Table 3.5 also gives the maximal degree of an edge.

The maximal number of 112 edges is achieved by a set of 24 vertices. But also for smaller vertex sets the number of edges is rather large. For example for 10 vertices there still exists a 2-neighbourly polytope, which means that any pair of vertices is adjacent. Thus for small vertex sets the number of edges increases near quadratically with the number of vertices. This can also be seen from Figure 3.4 where the average number of edges for 10-dimensional 0/1-polytopes is given. Figure 3.4 also gives the upper and lower bounds attained by 20 random vertex sets for each set size. We thus conclude that if the dimension is very high, e.g. $d \geq 100$, the number of edges of the convex hull of a random vertex set of the d-cube is likely to be quadratic in the number of vertices, see also Section 3.5.

Let us now consider the degree of the vertices. Clearly the average degree is just one half of the average number of edges. Thus we ask here only for the maximal degree a vertex can achieve. A lower bound is given by 2^{d-1}: Take one half of the cube, say all vertices on the

k	number of edges min. / average / max.			max. degree
6	15	15.00	15	5
7	19	19.80	21	6
8	22	25.07	28	7
9	24	30.66	36	8
10	25	36.42	45	9
11	31	42.22	51	10
12	30	47.94	60	11
13	36	53.51	67	12
14	37	58.86	72	13
15	39	63.93	77	14
16	40	68.69	84	15
17	48	73.10	89	16
18	45	77.15	91	17
19	55	80.79	96	17
20	52	84.01	100	17
21	57	86.79	105	17
22	61	89.09	110	17
23	64	90.93	109	16
24	60	92.28	112	16
25	75	93.14	109	16
26	75	93.49	106	15
27	80	93.30	103	14
28	76	92.49	100	13
29	85	90.96	95	12
30	83	88.54	90	10
31	85	85.00	85	8
32	80	80.00	80	5

Table 3.5: Number of edges and maximal degree of 5-dimensional 0/1-polytopes with k vertices.

"left" side, and one additional vertex on the "right" side. Then obviously the vertex on the right side is adjacent to all other vertices, i.e. has degree 2^{d-1}.

It suggests itself to conjecture that the above lower bound is tight. By complete enumeration it is easy to verify that this is in fact the case for $d \leq 4$. But surprisingly the conjecture does not hold in general, cf. Table 3.5. Table 3.6 gives an example of 18 vertices of dimension 5, their degree and incident vertices. It can be seen that vertex v_{11} is adjacent to all other vertices of the set, i.e. has degree 17.

vertices	degree	incident vertices
$v_1 =$ (0,0,0,0,0)	8	$v_2, v_3, v_5, v_8, v_9, v_{10}, v_{11}, v_{12}$
$v_2 =$ (0,0,0,0,1)	7	$v_1, v_4, v_6, v_8, v_9, v_{11}, v_{13}$
$v_3 =$ (0,0,0,1,0)	7	$v_1, v_4, v_7, v_8, v_{10}, v_{11}, v_{14}$
$v_4 =$ (0,0,0,1,1)	7	$v_2, v_3, v_6, v_7, v_8, v_{11}, v_{15}$
$v_5 =$ (0,0,1,0,0)	7	$v_1, v_6, v_7, v_9, v_{10}, v_{11}, v_{16}$
$v_6 =$ (0,0,1,0,1)	7	$v_2, v_4, v_5, v_7, v_9, v_{11}, v_{17}$
$v_7 =$ (0,0,1,1,0)	7	$v_3, v_4, v_5, v_6, v_{10}, v_{11}, v_{18}$
$v_8 =$ (0,1,0,1,1)	11	$v_1, v_2, v_3, v_4, v_9, v_{10}, v_{11}, v_{12}, v_{13}, v_{14}, v_{15}$
$v_9 =$ (0,1,1,0,1)	11	$v_1, v_2, v_5, v_6, v_8, v_{10}, v_{11}, v_{12}, v_{13}, v_{16}, v_{17}$
$v_{10} =$ (0,1,1,1,0)	11	$v_1, v_3, v_5, v_7, v_8, v_9, v_{11}, v_{12}, v_{14}, v_{16}, v_{18}$
$v_{11} =$ (1,0,1,1,1)	17	$v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{12}, v_{13}, v_{14}, v_{15}, v_{16}, v_{17}, v_{18}$
$v_{12} =$ (1,1,0,0,0)	8	$v_1, v_8, v_9, v_{10}, v_{11}, v_{13}, v_{14}, v_{16}$
$v_{13} =$ (1,1,0,0,1)	7	$v_2, v_8, v_9, v_{11}, v_{12}, v_{15}, v_{17}$
$v_{14} =$ (1,1,0,1,0)	7	$v_3, v_8, v_{10}, v_{11}, v_{12}, v_{15}, v_{18}$
$v_{15} =$ (1,1,0,1,1)	7	$v_4, v_8, v_{11}, v_{13}, v_{14}, v_{17}, v_{18}$
$v_{16} =$ (1,1,1,0,0)	7	$v_5, v_9, v_{10}, v_{11}, v_{12}, v_{17}, v_{18}$
$v_{17} =$ (1,1,1,0,1)	7	$v_6, v_9, v_{11}, v_{13}, v_{15}, v_{16}, v_{18}$
$v_{18} =$ (1,1,1,1,0)	7	$v_7, v_{10}, v_{11}, v_{14}, v_{15}, v_{16}, v_{17}$

Table 3.6: 5-dimensional 0/1-polytope with 18 vertices and 75 edges. Vertex v_{11} achieves the maximal degree 17.

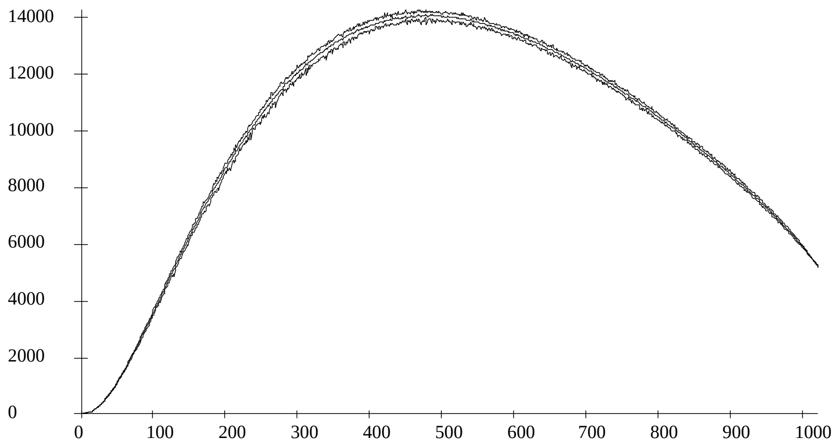


Figure 3.4: Average number of edges of 10-dimensional 0/1-polytopes over the number of vertices.

3.4 2-Neighbourly 0/1-Polytopes

As mentioned in the last section a 2-neighbourly 0/1-polytope is a polytope where any pair of vertices is adjacent. This means that the convex hull of a 2-neighbourly 0/1-polytope with k vertices has $\frac{k(k-1)}{2}$ edges, or, in other words, it represents a complete graph. Clearly 2-neighbourly 0/1-polytopes are worst case examples for algorithms based on the edge skeleton of their convex hull. Thus for fixed dimension d one may ask for the maximal number $N_2(d)$ of vertices of the d-cube for which a 2-neighbourly 0/1-polytope exists.

Obviously the d-simplex is a 2-neighbourly polytope, i.e. $N_2(d) \geq d+1$. On the other hand "one half" of a 2-neighbourly 0/1-polytope for the d-cube yields a 2-neighbourly 0/1-polytope for the $(d-1)$-cube. And adding one vertex to a 2-neighbourly 0/1-polytope for the $(d-1)$-cube gives a 2-neighbourly 0/1-polytope for the d-cube. Thus $N_2(d-1)+1 \leq N_2(d) \leq 2N_2(d-1)$.

dimension	1	2	3	4	5	6	7	8	9	10	11	12
vertices	2	3	4	6	10	13	≥ 18	≥ 20	≥ 25	≥ 32	≥ 38	≥ 46
edges	1	3	6	15	45	78	≥ 153	≥ 190	≥ 300	≥ 496	≥ 703	≥ 1035

Table 3.7: 2-neighbourly 0/1-polytopes with maximal number of vertices for dimensions 1 to 12.

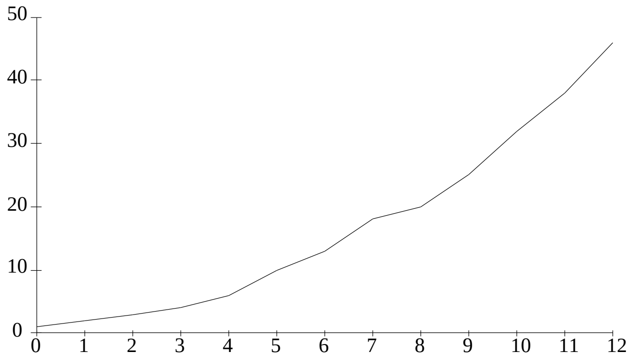


Figure 3.5: Maximal number of vertices for 2-neighbourly 0/1-polytopes for dimension 1 to 12. For $d \geq 7$ the curve shows the lower bound.

We have $N_2(d) = d+1$ for $d = 1 \ldots 3$, and complete enumeration gives the maximal value for $d = 4 \ldots 6$. To this end we utilize the results of the last section. This immediately gives $N_2(4) = 6$ and $N_2(5) = 10$, where for example only one equivalence class of $\mathcal{C}^5$ provides a 2-neighbourly 0/1-polytope with 10 vertices. For dimension $d = 6$ random vertex sets give a lower bound of $N_2(6) \geq 13$. From the above discussion it follows that any vertex set of $\mathcal{C}^6$ with $N_2(6) \geq 14$ must contain a subset of $\mathcal{C}^5$ with 7 or more vertices. We thus considered all 447 different 2-neighbourly vertex sets of $\mathcal{C}^5$ with at least 7 vertices and completed them with vertices on the "opposite half" of $\mathcal{C}^6$. A complete enumeration of all resulting 6-dimensional 2-neighbourly 0/1-polytopes (about half a million) showed $N_2(6) = 13$.

For higher dimensions we made incomplete enumerations. E.g. for dimension $d = 7$ we used a similar method as for dimension 6, getting the lower bound $N_2(7) \geq 18$. The investigations also showed that if there exists a vertex set of $\mathcal{C}^7$ with $N_2(7) \geq 19$ then no 6-dimensional subcube contains more than 12 vertices. This yields an upper bound of $N_2(7) \leq 24$.

Table 3.7 gives $N_2(d)$ for dimensions $d = 1 \ldots 12$, and Figure 3.5 shows that $N_2(d)$ seems to increase super linearly. This indicates that for higher dimensions there might exist large vertex sets of the d-cube which yield 2-neighbourly 0/1-polytopes. We state it as an open question to find general lower and upper bounds for the size of d-dimensional 2-neighbourly 0/1-polytopes.

3.5 Incomplete Enumeration for Higher Dimensions

There are several practical applications, like clustering binary data (see Chapter 7) or $\{0,1\}$-string searching (see Chapter 8), which are based on the edge skeleton of high dimensional 0/1-polytopes. Typical input parameters for these tasks might be $d = 100 \dots 200$ with $n = 1000 \dots 50000$ vertices. Thus the number of vertices is rather small compared to the number of "corners" of the d-cube. To get some insight of the structure of high dimensional edge skeletons with few vertices we investigated random vertex sets for dimension $d = 15$ of up to 240 vertices.

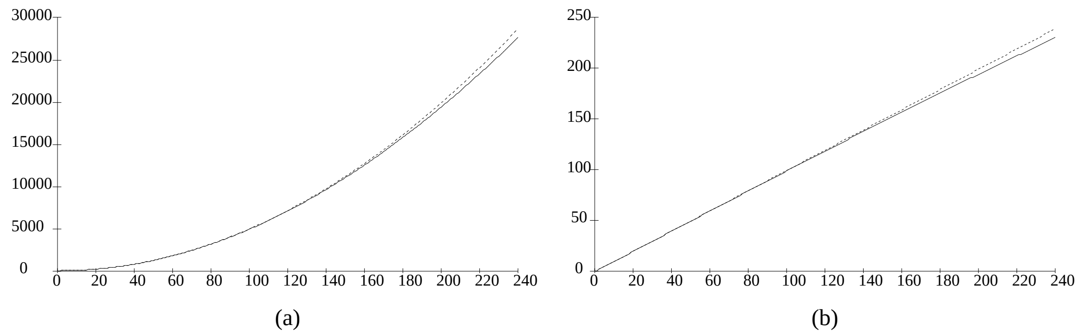


Figure 3.6: Edge skeleton of 0/1-polytopes for dimension $d = 15$ and $n = 1 \dots 240$ vertices: (a) average number of edges (solid) and upper bound $n(n-1)/2$ (dotted) (b) average edge degree (solid) and upper bound $n-1$ (dotted).

For dimension $d = 10$ Figure 3.4 already shows that for a small number of vertices the number of edges of the convex hull increases quadratically. Figure 3.6(a) shows the average size of the edge skeleton for dimension $d = 15$ and $n = 1 \dots 240$ vertices, compared to the upper bound of $n(n-1)/2$. It can be seen that the size of the edge skeleton almost matches the upper bound.

Figure 3.6(b) gives the average edge degree and the upper bound $n-1$. Again the curves are very similar, i.e. a random 0/1-polytope with the given parameters is very likely to be a good approximation of a 2-neighbourly 0/1-polytope with the same number of vertices.

This indicates that algorithms based on the edge skeleton of the convex hull of high dimensional 0/1-polytopes are most likely to attain their worst case behavior.

Figure 3.7 gives the maximal average edge degree and the maximal number of edges of d-dimensional 0/1-polytopes for dimension $d = 1 \dots 10$. To this end we considered random samples of 100 0/1-polytopes for each dimension and each number of vertices. In correspondence with Figure 3.6 these curves show that the number of edges and the average edge degree grows significantly with increasing dimensions.

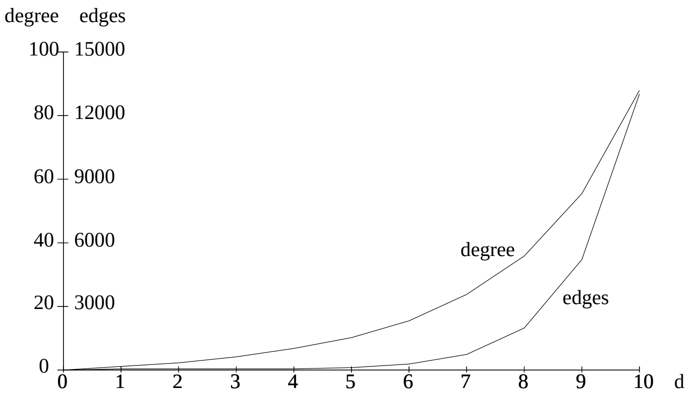


Figure 3.7: Maximal average edge degree and maximal number of edges of 0/1-polytopes for dimension $d = 1 \dots 10$.

Chapter 4

Covering the Hypercube

4.1 Introduction

Similar to Chapter 2 we are again interested in the set $\mathcal{H}(\mathcal{C}^d)$ of all hyperplanes that can be spanned by (d affinely independent) vertices of $\mathcal{C}^d$. In this chapter we concentrate on the following problems which are of interest in coding theory: What is the minimum number $C(d)$ of skew (non-axis-parallel) hyperplanes that cover all the 2^d vertices of the d-cube $\mathcal{C}^d$? More generally we ask: What is the maximum number $c_k(d)$, $1 \leq k \leq d$, of vertices of $\mathcal{C}^d$ that can be covered by k skew hyperplanes? The relation between these two questions is obvious: $C(d) = \min\{k | c_k(d) = 2^d\}$. Note that if we do not require hyperplanes to be skew the answer for the first problem is trivially 2 and thus $c_i(d) = 2^d$ for $i \geq 2$. Obviously $c_1(d) = 2^{d-1}$ since there is at least one axis to which a hyperplane is not parallel and therefore for every vertex covered by the hyperplane there exists at least one uncovered counterpart.

We give general bounds and the exact numbers for both problems if either $d \leq 6$ or $k \leq 2$. In addition, we can show that for $d \geq 5$ strictly less than d skew hyperplanes are sufficient but that at least d skew hyperplanes are needed for $d \leq 4$.

To indicate the connection to coding theory, vertices in the same hyperplane provide a set of binary strings (coding set) for which membership can be tested straightforwardly. $c_k(d)$ is the number of valid coding strings and k counts the number of tests for validity.

4.1.1 Notation

Recall that with $\mathcal{H}(\mathcal{C}^d)$ we denote the set of all hyperplanes that can be spanned by d affinely independent vertices of $\mathcal{C}^d$. In Section 4.6 for $d \geq 4$ we will show that for the covering problem it is sufficient to consider only hyperplanes of $\mathcal{H}(\mathcal{C}^d)$.

An important concept introduced in Chapter 2 will be symmetry, which is fulfilled by hyperplanes passing through the center $(\frac{1}{2}, \ldots, \frac{1}{2})$ of $\mathcal{C}^d$, i.e. halving the set of d-cube vertices they avoid. We will use symmetry to give a relation between the cover numbers of $\mathcal{C}^d$ and $\mathcal{C}^{d+1}$.

With $c_k(d)$ we denote the *k-cover number*, i.e. the maximum number of vertices of $\mathcal{C}^d$ that can be covered by k skew hyperplanes. The number covered by a single hyperplane, $c_1(d)$, is sometimes called the *rank* of $\mathcal{C}^d$ [92]. If we restrict the covering to skew symmetric hyperplanes we denote the k-cover number by $s_k(d)$ and similarly by $a_k(d)$ for skew asymmetric hyperplanes. The *cover number* $C(d)$ denotes the minimum number of skew hyperplanes that cover all 2^d vertices of $\mathcal{C}^d$, i.e. $C(d) = \min\{k|c_k(d) = 2^d\}$.

With *level* i we denote the set of vertices with exact i 1s. Clearly all vertices of level i can be covered by a single hyperplane $((1,\dots,1), i)$ perpendicular to the main diagonal. This type of hyperplane and its reflections will play an important role in deriving bounds for the k-cover number.

4.2 Related Work

In [92] Saks gave a nice survey on slicing (what is the minimum size of a set of hyperplanes that slices all edges of the cube?) and covering the hypercube. We first cite some relevant results from this work.

Using a generalization of the Sperner Theorem [14, 106] the following result is well known.

Lemma 4.1 *[76, 92] The maximum number of points covered by a skew hyperplane is* $c_1(d) = \binom{d}{\lceil d/2 \rceil}$.

Note that a hyperplane which covers level $\lceil d/2 \rceil$ attains this bound and this type is unique up to reflections and rotations.

With this lemma a lower bound for $C(d)$ is $\frac{2^d}{\binom{d}{\lceil d/2 \rceil}} = \Theta(\sqrt{d})$. On the other hand, a simple construction for $d \geq 2$ shows that d hyperplanes are sufficient: Each of $d-1$ hyperplanes h^i, $i = 1,\dots,d-1$, covers all vertices of level i and the remaining hyperplane h^d just needs to cover the two points $(0,\dots,0)$ and $(1,\dots,1)$. Note the special case of $C(1) = 2$.

Observation 4.1 $\frac{2^d}{\binom{d}{\lceil d/2 \rceil}} \leq C(d) \leq d,\ d \geq 2$

A lower bound on $c_k(d)$ now can be derived by summing over the k middle levels:

Lemma 4.2 *[92]* $c_k(d) \geq \sum_{i=\lceil \frac{d-k+1}{2} \rceil}^{\lceil \frac{d+k-1}{2} \rceil} \binom{d}{i},\ 1 \leq k \leq d$

Although the problem of determining $C(d)$ and $c_k(d)$ was open for $k > 1$ and $d > 3$, there are some special cases where the exact cover number is known. Let us call a hyperplane *monotone* if all elements of its normal vector are non-negative.

Theorem 4.1 *[92] The minimum number of monotone skew hyperplanes that cover all points of* $\mathcal{C}^d$ *is* $d+1$.

By reflections this applies to any set of hyperplanes which have a common orientation, i.e. they have the same sign for each element a_i of their normal vector a. Thus the theorem especially applies to a set of parallel hyperplanes.

Let a *±-hyperplane* be a hyperplane for which each coefficient of the normal vector is in $\{-1,+1\}$. In Chapter 2 we have shown that these are skew hyperplanes in $\mathcal{H}(\mathcal{C}^d)$ that do not (properly) cut d-cube edges. They might be called *hull-honest* hyperplanes as their intersection figure with $\mathcal{C}^d$ coincides with the convex hull of the vertices of $\mathcal{C}^d$ they contain. Using a result of Alon et. al. [8] Saks states the following:

Theorem 4.2 *[8, 92] To cover $\mathcal{C}^d$ by $\pm$–hyperplanes requires at least d hyperplanes if d is even and $d+1$ if d is odd, respectively.*

In [9] Alon and Füredi gave some surprising and useful results, if some of the vertices of $\mathcal{C}^d$ have to be missed (note that here we don't have the restriction to skew hyperplanes):

Lemma 4.3 *[9] Any set of $k \leq d$ hyperplanes that misses at least one vertex of $\mathcal{C}^d$ misses at least 2^{d-k} vertices.*

And thus

Theorem 4.3 *[9] Any set of hyperplanes which covers all vertices of $\mathcal{C}^d$ but one has at least size d.*

This result is especially surprising in the light of Theorem 4.13 in Section 4.4. To cover all vertices but one of $\mathcal{C}^d$, $d \geq 5$, by arbitrary hyperplanes is more difficult (in a quantitative sense) than to cover all vertices of $\mathcal{C}^d$ with the restriction to skew hyperplanes. This gives some evidence to the intrinsic complexity of the covering problem.

Let us call vertices which are only covered by exactly one hyperplane of a given set of hyperplanes as belonging to this hyperplane. As a consequence of Lemma 4.3 we immediately get

Lemma 4.4 *[9] If $C(d) = k$ then each hyperplane of a set of size k which covers $\mathcal{C}^d$ contains at least 2^{d+1-k} own vertices.*

4.3 Basic Relations

O'Neil [81] first stated the hypercube slicing problem: what is the size, $S(d)$, of a minimum set of hyperplanes that slices all $d2^{d-1}$ edges of the cube? Such a set of hyperplanes is called an *edge slicing set* and $S(d)$ is called the slicing number. See [92] for a nice survey on this topic and Chapter 5 for further investigations.

We give a simple relation between the cover number and the slicing number:

Theorem 4.4 $S(d) \leq 2C(d) - 2$

Proof. To prove this theorem it is sufficient to show that from any covering set of size k we can construct a slicing set of at most size $2k-2$. This can be done by introducing for every hyperplane h^i, $1 \leq i \leq k$, of the covering set two new hyperplanes h^i_l and h^i_u which are parallel to h^i and lie some small ϵ below and above h^i, respectively. Since h^i is skew, every edge emanating from a vertex covered by h^i is adjacent to a vertex above or below h^i. Thus by construction the two hyperplanes h^i_l and h^i_u cut all these edges, i.e. all edges emanating from points covered by h^i. Since we have a covering set of size k every vertex of $\mathcal{C}^d$ is covered by at least one hyperplane and we thus get a slicing set $\{h^i_l, h^i_u | i = 1, \ldots, k\}$ of size $2k$. By this set every edge of the hypercube is sliced at least twice, and therefore we may remove an arbitrary pair h^i_l, h^i_u to obtain a slicing set of size $2k-2$. □

For the slicing problem a simple method of adding two extended hyperplane sets for lower dimensions can be used to show its subadditivity, i.e. that $S(d_1 + d_2) \leq S(d_1) + S(d_2)$ [92].

Therefore improvements for specific low dimensions immediately yield better constructions for higher dimensions. The situation for covering the cube is more subtle: While k-parallelism doesn't play a role for slicing the cube (all hyperplanes there have sufficient freedom to be not parallel to any axis by at least some epsilon), the same idea is not applicable for covering: it would just lead to axes-parallel hyperplanes.

We now use the concept of symmetry to derive relations between $c_k(d)$ and $c_k(d+1)$. Recall that a symmetric hyperplane in $\mathcal{R}^d$ is one which passes through the center $(\frac{1}{2}, \ldots, \frac{1}{2})$ of $\mathcal{C}^d$, that is, a hyperplane $(v, b) \in \mathcal{R}^d$ with $\sum_{i=1}^d v_i = 2b$. For symmetric hyperplanes H, every two antipodal vertices $x, \overline{x} \in V^d$ either both lie in H or they lie on opposite sides of H. Thus the vertices of the d-cube which are avoided by the hyperplane are partitioned into equal-sized subsets. In [2, 7] it is shown that this is also a sufficient condition for symmetry, provided that the hyperplane covers at least one vertex of $\mathcal{C}^d$. Also recall that if we restrict the covering to asymmetric (symmetric) hyperplanes we denoted the k-covering number by $a_k(d)$ and $s_k(d)$, respectively.

Let us first give some simple observations and relations.

Observation 4.2 *a)* $c_k(d) \leq c_k(d+1)$
b) $c_k(d) \leq k c_1(d)$
c) $c_k(d) - c_{k-1}(d) \leq c_1(d)$
d) $c_k(d) \geq \max\{a_k(d), s_k(d)\}$

Lemma 4.5 *For $d \geq 3$ and $j \leq k \leq d$ we have*

a) $\frac{1}{2} c_k(d) + k \leq a_k(d) \leq c_k(d) \leq 2c_k(d-1)$
b) $k c_j(d) \geq j c_k(d)$

Proof.

a) To prove the left inequality we take a k-covering of $\mathcal{C}^d$ and split $\mathcal{C}^d$ perpendicular to an arbitrary axis into two $(d-1)$-cubes. Then for at least one $(d-1)$-cube $\frac{c_k(d)}{2}$ or more vertices are covered. By adding at least one vertex of the other $(d-1)$-cube to each hyperplane this k-covering can be extended to an asymmetric k-covering of $\mathcal{C}^d$, $d \geq 3$, which covers at least $\frac{c_k(d)}{2} + k$ vertices.

When adding these k vertices to the hyperplanes we have to take care that none of the hyperplanes becomes symmetric or parallel to an axis. Moreover every added vertex counts only for one hyperplane. By simple case analysis it is not hard to see that such an extension is possible for dimensions $d = 3 \ldots 5$, cf. also Table 4.5.

For general $d \geq 6$ we use the following simple construction. Assign an unique binary number b to every hyperplane covering vertices of the $(d-1)$-cube. Because of its skewness at least two of the four vertices $(0, 0, b), \ldots, (1, 1, b) \in \mathcal{C}^{d-1}$ are not covered by this hyperplane. It is not hard to see that at least one of them can be extended to a vertex of $\mathcal{C}^d$, such that the hyperplane which covers both, the 'old' vertices of the $(d-1)$-cube and this new vertex, is skew and asymmetric.

Since b is unique for every hyperplane we thus get the required k vertices. Since $k \leq d$ we need at most $\lceil ld\ d \rceil$ bits for b. Thus $\lceil ld\ d \rceil + 2 \leq d - 1$ and it follows that we can use the above construction for $d \geq 6$.

The second inequality is obvious. To prove the right inequality one can use the same idea as before of splitting the d-cube into two $(d-1)$-cubes. □

b) For a k-covering of $\mathcal{C}^d$ every hyperplane has an average number of $\frac{c_k(d)}{k}$ own vertices. If we sort the hyperplanes by the number of their own vertices in non increasing order, the first j hyperplanes clearly have an average of at least $\frac{c_k(d)}{k}$ own vertices per hyperplane. Thus these j hyperplanes cover at least $j\frac{c_k(d)}{k}$ vertices. □

For the special case $j = k-1$ Lemma 4.5 b) gives a bound on the difference between successive cover numbers and thus a bound on $c_k(d)$ in terms of $c_{k-1}(d)$.

Corollary 4.1 *$c_k(d) - c_{k-1}(d) \leq \frac{c_k(d)}{k}$, that is, $c_k(d) \leq \frac{k}{k-1}c_{k-1}(d)$ for $2 \leq k \leq d$*

Observation 4.2 a) to c), Lemma 4.5 b) and Corollary 4.1 are stated in terms of $c_k(d)$ but are also true for $s_k(d)$ and $a_k(d)$, respectively.

4.4 Main Results

In this section we summarize new results towards determining $C(d)$ and $c_k(d)$. In Chapter 2 we have shown that skew and asymmetric hyperplanes in $\mathcal{H}(\mathcal{C}^d)$ correspond to skew and symmetric ones in $\mathcal{H}(\mathcal{C}^{d+1})$, in a unique manner. The bijection used there proves useful for constructing similar relations for covering sets.

Theorem 4.5 *There exists a set of k asymmetric skew hyperplanes covering $\mathcal{C}^d$ iff there exists a set of k symmetric skew hyperplanes covering $\mathcal{C}^{d+1}$.*

Proof. The obvious bijection between asymmetric hyperplanes $(v,b) \in \mathcal{R}^d$ and symmetric hyperplanes $(w,b) \in \mathcal{R}^{d+1}$, is given by $w_i = v_i$ for $i = 1,\ldots,d$ and $w_{d+1} = 2b - \sum_{i=1}^{d} v_i$. Note that, under this correspondence, (v,b) is asymmetric and skew if and only if (w,b) is symmetric and skew. Furthermore, it is easy to check that a vertex x of $\mathcal{C}^d$ lies in (v,b) if and only if the antipodal vertex pair $(x,0)$ and $(\overline{x},1)$ of $\mathcal{C}^{d+1}$ lies in (w,b). Hence we have a set $v^1,\ldots,v^k$ of skew asymmetric hyperplanes covering $a_k(d)$ vertices of $\mathcal{C}^d$ iff we have a set $w^1,\ldots,w^k$ of skew symmetric hyperplanes covering $s_k(d+1) = 2a_k(d)$ vertices of $\mathcal{C}^{d+1}$ and thus the theorem follows. □

The idea of this construction can be generalized as follows.

Theorem 4.6 *If $C(d) = k$ then $C(d+1) = k$ iff there are two hyperplane sets of size k, each covering $\mathcal{C}^d$, such that there exists a one-to-one relation between parallel but non-identical hyperplanes of these two sets.*

Proof. If such sets exist then every pair of corresponding hyperplanes can be combined to a skew hyperplane in $\mathcal{R}^{d+1}$ using the same technique as in the previous proof. The union of these hyperplanes clearly covers $\mathcal{C}^{d+1}$. On the other hand, if we have k skew hyperplanes covering $\mathcal{C}^{d+1}$ we can split the $(d+1)$-cube into two d-cubes and get two sets of k hyperplanes, each covering $\mathcal{C}^d$. Clearly for each hyperplane in one set there is a corresponding parallel hyperplane in the other set and since the hyperplane in $\mathcal{R}^{d+1}$ this pair stems from was skew, they are not identical. $\square$

Corollary 4.2 *For $d \geq 4$ and $1 \leq k \leq d$ we have*

$$c_k(d) \geq s_k(d) = 2a_k(d-1) \geq c_k(d-1) + 2k$$

Proof. The proof of Theorem 4.5 provides the middle equality and Lemma 4.5 a) gives $2a_k(d-1) \geq c_k(d-1) + 2k$. $\square$

As an immediate consequence of Lemma 4.5 a) and Corollary 4.2 we get

Theorem 4.7 *If $c_k(d) = a_k(d)$ then $c_k(d+1) = s_k(d+1) = 2a_k(d) = 2c_k(d)$.*

The only previously known result on $c_k(d)$ for higher dimension was for $k = 1$. In the next theorem we give a simple construction to achieve the optimum for $k = 2$.

Theorem 4.8 *The maximum number of points covered by $k = 2$ skew hyperplanes is $c_2(d) = 2c_1(d) = 2\binom{d}{\lceil d/2 \rceil}$.*

Proof. Consider the two hyperplanes $h^1 = ((1, \ldots, 1), \lfloor \frac{d}{2} \rfloor) \in \mathcal{H}(\mathcal{C}^d)$ and $h^2 = ((1, \ldots, 1, -1), \lfloor \frac{d}{2} \rfloor - 1) \in \mathcal{H}(\mathcal{C}^d)$. It is easy to check that both of them cover the maximum of $\binom{d}{\lceil d/2 \rceil}$ vertices. Furthermore the vertex sets they cover are disjoint, since the inner product of any vertex $x \in V^d$ with the normal vector of h^1 is either identical to or differs exactly by 2 from the inner product of x with the normal vector of h^2. $\square$

Note that there are also other hyperplane pairs which achieve the maximum, but most of them exist only in even or odd dimensions. For even d, for example, $h^1 = ((1, \ldots, 1), \frac{d}{2}) \in \mathcal{H}(\mathcal{C}^d)$ and $h^2 = ((1, \ldots, 1_{\frac{d}{2}}, -1, \ldots, -1), 0) \in \mathcal{H}(\mathcal{C}^d)$ are only disjoint[1] hyperplanes for $d \equiv 2 \pmod 4$ ($x \in h^1$ has an odd number of $+1s$ while $x \in h^2$ has an even number of $+1s$).

Theorem 4.9 $c_3(d) \geq 2\left\{\binom{d}{\frac{d-1}{2}} + \binom{d-1}{\frac{d-3}{2}}\right\}$ *for $d \geq 3$ odd.*
$c_3(d) \geq \binom{d+1}{\frac{d}{2}} + \binom{d}{\frac{d-2}{2}}$ *for $d \geq 4$ even.*

Proof. We first proof the bound for odd $d \geq 3$. Consider the three hyperplanes $h^1 = ((1, \ldots, 1), \frac{d-1}{2}) \in \mathcal{H}(\mathcal{C}^d)$, $h^2 = ((1, \ldots, 1), \frac{d+1}{2}) \in \mathcal{H}(\mathcal{C}^d)$ and $h^3 = ((1, \ldots, 1, -2), \frac{d-3}{2}) \in \mathcal{H}(\mathcal{C}^d)$. h^1 and h^2 cover level $(d-1)/2$ and $(d+1)/2$, respectively, and thus $2\binom{d}{\frac{d-1}{2}}$ vertices. h^3 covers $\binom{d-1}{\frac{d-3}{2}}$ vertices of the type $(\{0,1\}^{d-1}, 0)$, from level $(d-3)/2$. Because of its symmetry, h^3 covers the same number of vertices from level $(d+3)/2$. Since all hyperplanes cover different levels the vertex sets they cover are disjoint and the given bound follows. For even $d \geq 4$ we only need to apply the right inequality from Lemma 4.5 a) to $c_3(d+1)$. $\square$

Note that although the bound for even d is derived in a different way it coincides with the bound of Lemma 4.2 whereas for odd dimensions the new bound is an improvement. This may give some evidence that the bounds are nearly optimal for 3-covering.

[1] We say that two hyperplanes are disjoint, if the vertex sets of V^d they cover are disjoint.

Theorem 4.10 $c_4(d) \geq a_4(d) \geq \frac{4d}{d+2} c_1(d)$ *for* $d \geq 4$ *even.*
$c_4(d) \geq s_4(d) \geq \frac{8(d-1)}{d+1} c_1(d-1)$ *for* $d \geq 5$ *odd.*

Proof. To prove the bound for even $d \geq 4$ consider the four asymmetric hyperplanes $h^1 = ((1, \ldots, 1), \frac{d}{2} - 1) \in \mathcal{H}(\mathcal{C}^d)$, $h^2 = ((1, \ldots, 1), \frac{d}{2} + 1) \in \mathcal{H}(\mathcal{C}^d)$, $h^3 = ((1, \ldots, 1, -1), \frac{d}{2}) \in \mathcal{H}(\mathcal{C}^d)$ and $h^4 = ((1, \ldots, 1, -1), \frac{d}{2} - 2) \in \mathcal{H}(\mathcal{C}^d)$. h^1 and h^2 cover levels $d/2-1$ and $d/2+1$, respectively, and thus $2\binom{d}{d/2-1}$ vertices. h^3 covers all vertices of level $d/2$ of the type $(\{0,1\}^{d-1}, 0)$ plus $\binom{d-1}{d/2+1}$ vertices of the type $(\{0,1\}^{d-1}, 1)$ from level $d/2 + 2$. Similarly h^4 covers all vertices of level $d/2$ of the type $(\{0,1\}^{d-1}, 1)$ plus $\binom{d-1}{d/2-2}$ vertices of the type $(\{0,1\}^{d-1}, 0)$ from level $d/2 - 2$. Thus Levels $d/2 - 1$, $d/2$, $d/2 + 1$ are fully covered, and levels $d/2 - 2$ and $d/2 + 2$ are partially covered. Since all hyperplanes cover different levels the vertex sets they cover are disjoint and we can sum up their numbers which results in $\frac{4d}{d+2}\binom{d}{\frac{d}{2}}$, the promised lower bound for even d. The bound for odd d now follows from Corollary 4.2. □

Together with the fact that $c_1(d-1) \geq \frac{1}{2} c_1(d)$ Theorem 4.10 shows that for $d \to \infty$ the 4-covering approaches the optimum of $4c_1(d)$. Moreover the bounds are tight as far as we could enumerate coverings numerically, see Table 4.4. Note that if we supplement the construction of the proof for even d by the symmetric hyperplane $h^5 = ((1, \ldots, 1, 5), \frac{d}{2} + 2)$ we get an alternative covering of levels $\frac{d}{2} - 2$ to $\frac{d}{2} + 2$ with 5 hyperplanes (cf. Lemma 4.2).

Next we give some lower bounds if we use almost d hyperplanes.

Theorem 4.11 *a)* $c_{d-2}(d)$ *is either* $2^d - 4$ *or* 2^d, $d \geq 4$.
b) either $2^d - 2d - 2 \leq c_{d-3}(d) \leq 2^d - 8$ *or* $c_{d-3}(d) = 2^d$, $d \geq 4$.

Proof. The bounds for $c_{d-3}(d)$ follow by combining Lemmas 4.2 and 4.3. From Lemma 4.3 we also know that $c_{d-2}(d)$ is either $\leq 2^d - 4$ or equals 2^d. It remains to be shown that $c_{d-2}(d) \geq 2^d - 4$ can always be obtained. To this end we use the following set of hyperplanes: the symmetric hyperplane $h^1 = ((1, \ldots, 1, 3-d), 1)$ and the parallel hyperplanes $h^i = ((1, \ldots, 1), i)$ for $i = 2, \ldots, d-2$, where h^i covers all vertices of level i. It is easy to verify that h^1 covers all but one vertices of level 1 and because of its symmetry all but one vertices of level $d-1$. Thus the only vertices not covered by this set are $(0, \ldots, 0)$, $(0, \ldots, 0, 1)$, $(1, \ldots, 1, 0)$ and $(1, \ldots, 1)$. □

The construction from the last proof can be used to replace a pair of two hyperplanes covering levels b and $d - b$ by a single symmetric hyperplane $h = ((1, \ldots, 1, 2b - d + 1), b)$ which misses $2\binom{d-1}{b-1}$ vertices from these two levels. Starting with a $(d-1)$–covering of levels $i = 1, \ldots, d-1$ with hyperplanes $h^i = ((1, \ldots, 1), i)$ we thus replace $d - k - 1 \leq \lfloor \frac{d-1}{2} \rfloor$ of the appropriate pairs to get the bound $c_k(d) \geq 2^d - 2 - 2\sum_{i=1}^{d-k-1} \binom{d-1}{i-1}$ for $k \geq \lfloor \frac{d}{2} \rfloor$. For $k = d - 2$ this gives the improved lower bound stated in the last theorem. For $k = d - 1$ and $k = d - 3$ this coincides with the bound from Lemma 4.2 and for the remaining $c_k(d)$ it provides a weaker bound.

Next we show that for $d \geq 5$ we can always cover the vertices of the d-cube by at most $d-1$ skew hyperplanes. To this end we utilize the relation between asymmetric and symmetric hyperplanes as given by Theorem 4.5.

		d even, $d \equiv 0 \bmod 4$	d even, $d \equiv 2 \bmod 4$	d odd
sets of hyperplanes $h = (v, b)$	h_α h_β h_γ	$((-1, 1, \ldots, 1), 2i)$ $((1, \ldots, 1), 2i + 1)$ for $i = 0, \ldots, \frac{d}{2} - 1$	$((-1, 1, \ldots, 1), 2i - 1)$ $((1, \ldots, 1), 2i)$ for $i = 1, \ldots, \frac{d}{2} - 1$ $((1, \ldots, 1, 2 - d), \{0, 1\})$	$((-1, 1, \ldots, 1), 2i)$ $((1, \ldots, 1, -1), 2i + 1)$ for $i = 0, \ldots, \frac{d-3}{2}$ $((2, 3 - d, 1, \ldots, 1), 1)$
covered levels and vertices	h_α h_β h_γ	levels $0, 2, \ldots, d$ levels $1, 3, \ldots, d - 1$	levels $1, 3, \ldots, d - 1$, except $(1, 0, \ldots, 0)$ and $0, 1, \ldots, 1)$ levels $2, 4, \ldots, d - 2$ $(0, \ldots, 0)$, $(0, 1, \ldots, 1)$, $b = 0$ $(1, 0, \ldots, 0)$, $(1, \ldots, 1)$, $b = 1$	levels $0, 2, \ldots, d - 1$ except $(0, 1, \ldots, 1)$ levels $1, 3, \ldots, d$ except $(0, \ldots, 0, 1)$ $(0, \ldots, 0, 1)$, $(0, 1, \ldots, 1)$
cardinality of hyperplane sets	h_α h_β h_γ	$\frac{d}{2}$ $\frac{d}{2}$	$\frac{d}{2} - 1$ $\frac{d}{2} - 1$ 2	$\frac{d-1}{2}$ $\frac{d-1}{2}$ 1
$\sum_{i=1}^{d} v_i$	h_α h_β h_γ	$d - 2$ d	$d - 2$ d 1	$d - 2$ $d - 2$ 3

Table 4.1: Sets of skew, asymmetric hyperplanes to cover C^d, $d \geq 4$.

Theorem 4.12 *For $d \geq 4$, $\mathcal{C}^d$ can be covered by d skew, asymmetric hyperplanes.*

Proof. To prove the theorem we consider three different cases, namely for d even the two cases $d \equiv 0 \bmod 4$ and $d \equiv 2 \bmod 4$ and thirdly d odd. For every case a set of skew and asymmetric hyperplanes which covers $\mathcal{C}^d$ is given in Table 4.1. For each case there is a column in this table which gives the sets of necessary hyperplanes, the covered vertices for each set, the cardinality of the sets and the sum over the entries of their normal vectors, respectively.

We first show that the given sets are indeed covering sets of $\mathcal{C}^d$. Let us start with the the case d even, $d \equiv 0 \bmod 4$. Every hyperplane of the set h_α covers all vertices of the form $(0, \{0, 1\}^{d-1})$ of level $2i$ and all vertices of the form $(1, \{0, 1\}^{d-1})$ of level $2i + 2$. Thus these $\frac{d}{2}$ hyperplanes cover together all vertices of even levels. The hyperplanes of the set h_β are normal to the main diagonal, each covering exactly all vertices of an odd level. Thus the two sets h_α and h_β cover together the whole d-cube with d hyperplanes.

The case d even, $d \equiv 2 \bmod 4$ can be treated very similarly, although the role of odd and even levels changes and thus the cardinality of the sets is $\frac{d}{2} - 1$. Moreover there are four remaining vertices which are not covered by the sets h_α and h_β, cf. Table 4.1. Thus we need two additional hyperplanes, the set h_γ, to cover these vertices. Note that the two hyperplanes of h_γ may cover some additional vertices which are already covered by other hyperplanes and thus are not given in the table. Together we again obtain a covering set of size d.

For odd d we use a slightly different construction. We cannot use hyperplanes which are normal to the main diagonal because they would either leave to many uncovered vertices or we would get a covering set of cardinality $d+1$. Thus we build the set h_β similar to the set h_α, but rotate and 'shift' it accordingly, see Table 4.1. This gives us two sets of cardinality $\frac{d-1}{2}$, covering all even and odd, respectively, levels, each set leaving one uncovered vertex. These two vertices are covered by the last hyperplane (set h_γ), again resulting in a covering set of size d.

That all hyperplanes of the given sets are in fact asymmetric follows from the property that every symmetric hyperplane $h = (v, b)$ fulfills the relation $\sum_{i=1}^{d} v_i = 2b$ (cf. Section 4.3). The last row of Table 4.1 gives the sum over the entries of the normal vector for each set. Let us consider the case d even, $d \equiv 0$ mod 4. The remaining cases can be treated in a similar way. For the hyperplanes of the set h_α we have to ensure that $\frac{d-2}{2}$ is odd, since every 'b' of h_α is even. But this is clearly fulfilled for every $d \equiv 0$ mod 4. In contrary for every hyperplane of the set h_β $\frac{d}{2}$ has to be even. Again this holds for every $d \equiv 0$ mod 4.

Note that the restriction to $d \geq 4$ comes from the entries $2 - d$ and $3 - d$ in some of the normal vectors. For $d = 2$ or $d = 3$ these entries would vanish and thus cause parallelism of the respective hyperplanes. □

Theorem 4.13 *For $d \geq 5$, $\mathcal{C}^d$ can be covered by $d - 1$ skew, symmetric hyperplanes, i.e. $C(d) \leq d - 1$ and $c_{d-1}(d) = s_{d-1}(d) = 2^d$.*

Proof. Theorem 4.12 shows for $d \geq 5$ that $\mathcal{C}^{d-1}$ can be covered by $d-1$ skew and asymmetric hyperplanes. It follows by Theorem 4.5 that $\mathcal{C}^d$ can thus be covered by $d - 1$ skew and symmetric hyperplanes. □

Corollary 4.3 *For $d \geq 4$*

$$a_{d-1}(d) \geq \begin{cases} 2^d - d & for \quad d \equiv 0 \ mod\ 4 \\ 2^d - 2 & else \end{cases}$$

$$a_{d-2}(d) \geq \begin{cases} 2^d - 2d & for \quad d \equiv 0 \ mod\ 4 \\ 2^d - 4 & for \quad d \equiv 2 \ mod\ 4 \\ 2^d - d - 2 & else \end{cases}$$

$$a_{d-3}(d) \geq \begin{cases} 2^d - 3d & for \quad d \equiv 0 \ mod\ 4 \\ 2^d - \frac{d(d-1)}{2} - 4 & for \quad d \equiv 2 \ mod\ 4 \\ 2^d - 2d - 2 & else \end{cases}$$

Proof. To prove the inequalities we consider the hyperplanes given in Table 4.1 with the lowest number of covered vertices. Removing these hyperplanes in order of non-decreasing number of covered vertices then directly gives the lower bounds.

For $d \equiv 0$ mod 4 and for either $i = 0$ or $i = \frac{d}{2} - 1$ there are two hyperplanes of each set h_α and h_β, respectively, which cover d vertices.

For $d \equiv 2$ mod 4 the two hyperplanes of h_γ cover two vertices each. For $i = 1$ the first hyperplane of h_β covers $\binom{d}{2}$ vertices.

For odd d the hyperplane of h_γ covers 2 vertices. For $i = 0$ the first hyperplane of h_α covers d vertices and for $i = \frac{d-3}{2}$ the last hyperplane of h_β covers d vertices, too. □

Table 4.5 shows that the bounds of Corollary 4.3 are tight for $3 \leq d \leq 7$ with only two exceptions. For $a_5(7)$ we found a set which covers 122 vertices and for $a_6(7)$ it is not known whether there exists a set which covers the whole 7-cube. If no such set exists the given lower bound would also be tight in this case.

Corollary 4.4 *For $d \geq 5$*

$$c_{d-2}(d) \geq s_{d-1}(d) \geq \begin{cases} 2^d - 2d + 2 & for \quad d \equiv 1 \ mod\ 4 \\ 2^d - 4 & else \end{cases}$$

$$c_{d-3}(d) \geq s_{d-2}(d) \geq \begin{cases} 2^d - 4d + 4 & for \quad d \equiv 1 \ mod\ 4 \\ 2^d - 8 & for \quad d \equiv 3 \ mod\ 4 \\ 2^d - 2d - 2 & else \end{cases}$$

$$c_{d-4}(d) \geq s_{d-3}(d) \geq \begin{cases} 2^d - 6d + 6 & for \quad d \equiv 1 \ mod\ 4 \\ 2^d - (d-1)(d-2) - 8 & for \quad d \equiv 3 \ mod\ 4 \\ 2^d - 4d & else \end{cases}$$

Proof. By Corollary 4.2 we have $c_k(d) \geq s_k(d) = 2a_k(d-1)$. Together with Corollary 4.3 the bounds follow. □

Again the bounds for $s_k(d)$ are tight for $4 \leq d \leq 7$ as can be seen from Table 4.5.

4.5 Low Dimensions

Although it is easy to find the cover number and the k-cover number for $d \leq 3$ they were unknown for $d > 3$ and $k > 1$. Here we give the results of dimension 4 and 5. In Section 4.6 we then give some numerical results for dimensions up to 10.

Theorem 4.14 *To cover $\mathcal{C}^4$ requires four skew hyperplanes, i.e. $C(4) = 4$.*

Proof. Suppose a set of three hyperplanes covers $\mathcal{C}^4$. Then using Theorem 4.6 there must exist two sets of three hyperplanes, each covering $\mathcal{C}^3$, such that there exists a one-to-one relation between parallel but not identical hyperplanes of the two sets. But the only way to cover $\mathcal{C}^3$ with three skew hyperplanes is unique up to reflections and rotations. The first hyperplane covers level 1, the second level 2, and the antipodal pair (0,0,0),(1,1,1) is covered by the third hyperplane. It is easy to figure out that there is no rotated or reflected copy of this set, which fulfills the required correspondence. Thus with Observation 4.1 it follows that $C(4)$ is indeed 4. □

Lemma 4.6 $c_3(4) = 14$ *and* $c_4(4) = 16$.

Proof. The last theorem showed $C(4) = 4$, thus $c_4(4) = 16$. From Lemma 4.3 we get $c_3(4) \leq 2^d - 2^{d-k} = 14$ and the following set of hull-honest hyperplanes covers 14 vertices: $h^i = ((1,1,1,1), i)$ for $i = 1, \ldots, 3$. (Note that the example coincides with the lower bound of Lemma 4.2.) □

Corollary 4.5 *To cover $\mathcal{C}^5$ requires exactly four skew hyperplanes, i.e. $C(5) = 4$.*

Proof. That 4 hyperplanes are necessary follows trivially from Theorem 4.14 and the fact that $C(d+1) \geq C(d)$. From Theorem 4.13 it follows that 4 hyperplanes are sufficient. $\square$

It is interesting to note that the set of hyperplanes used to cover $\mathcal{C}^4$ in the proof of Theorem 4.13 consists of four disjoint hyperplanes each covering exactly 4 vertices. We therefore got a covering of $\mathcal{C}^5$ with four disjoint hyperplanes, each covering 8 vertices. In fact we observed that all covering sets for $\mathcal{C}^5$ of size 4 entirely consist of symmetric hyperplanes.

Lemma 4.7 *$c_3(5) = 28$ and $c_4(5) = s_4(5) = 32$.*

Proof. $c_4(5) = s_4(5) = 32$ follows from the last Theorem. From Lemma 4.5 a) we get $c_3(5) \leq 2c_3(4) = 28$ and thus, by Theorem 4.11 a) or Theorem 4.9, $c_3(5) = 28$. $\square$

4.6 Numerical Investigations

Let $\mathcal{H}_s(\mathcal{C}^d)$ be the subset of skew (non-axis-parallel) hyperplanes of $\mathcal{H}(\mathcal{C}^d)$. Note that $\mathcal{H}_s(\mathcal{C}^d)$ does not contain all skew hyperplanes which contain vertices of the d-cube. For example to cover the 3-cube we need two hyperplanes $h^1 = ((1,1,1),1)$ and $h^2 = ((1,1,1),2)$ which cover 3 vertices each, and one hyperplane h^3 which covers the remaining antipodal pair $(0,0,0)$ and $(1,1,1)$. But now it is impossible to entirely determine h^3 with another vertex from V^d without making it parallel to an axis. Thus from the definition of $\mathcal{H}(\mathcal{C}^d)$ this hyperplane does not occur in our set $\mathcal{H}_s(\mathcal{C}^d)$. Let us denote such skew hyperplanes which are not fully determined by the vertices they cover and which by adding any other vertex of $\mathcal{C}^d$ would become parallel to an axis, as *pending* hyperplanes.

Clearly, for any pending hyperplane h which covers a set $V \subseteq V^d$ there exists a hyperplane $h' \in \mathcal{H}(\mathcal{C}^d)$ covering a superset of V. But since h' may now be axes-parallel there might be no hyperplane in $\mathcal{H}_s(\mathcal{C}^d)$ covering the set V, as was pointed out by the example above. Although we need pending hyperplanes to cover $\mathcal{C}^3$ with the minimum number of three hyperplanes, they do not play a role in higher dimensions, as we will show with the next two lemmas.

Theorem 4.15 *For any skew hyperplane h in $\mathcal{R}^d$, $d \geq 4$, there exists a vertex $x \in V^d$ which has Hamming distance at least two to any vertex $x' \in V^d$ covered by h.*

Proof. Let $h = (v, b)$ and w.l.o.g. arrange the entries of v according to their sign, i.e. $v = (v_1, v_2, \ldots, v_k, v_{k+1}, \ldots, v_d)$ with $v_i > 0$ for $1 \leq i \leq k < d$ and $v_i < 0$ for $k < i \leq d$. Note that we assume that v has at least one negative entry, else we take the inverse normal vector of h. Let $x = (1, \ldots, 1_k, 0, \ldots, 0)$ and $\overline{x}$ its complement. If either $x \in h$ or $\overline{x} \in h$ then obviously no other vertex is covered by h and for $d \geq 2$ nothing remains to be proven. So assume $x \notin h$ and $\overline{x} \notin h$. Suppose now the lemma is wrong. Then there must be neighbours (i.e. vertices with Hamming distance one) of x and $\overline{x}$ covered by h. There are two types of neighbours of x, namely i) $y' = (1, \ldots, 1_k, 0, \ldots, 0, 1_l, 0, \ldots, 0)$, $k < l \leq d$, and ii) $y' = (1, \ldots, 1, 0_l, 1, \ldots, 1_k, 0, \ldots, 0)$, $1 \leq l \leq k$, and similarly two types for $\overline{x}$, namely iii) $y'' = (0, \ldots, 0, 1_m, 0, \ldots, 0_k, 1, \ldots, 1)$, $1 \leq m \leq k$, and iv) $y'' = (0, \ldots, 0_k, 1, \ldots, 1, 0_m, 1, \ldots, 1)$, $k < m \leq d$.

At least one combination of vertices from i) & iii), i) & iv), ii) & iii) or ii) & iv), respectively, has to be covered by h to provide neighbours for both, x and $\overline{x}$. We will show that this can not be done without getting another vertex $z \in V^d$ which is not covered by h and has no neighbour covered by h.

1. i) & iii): From their construction y' and y'' have the two entries at position l and m in common and differ for all remaining indices. While for this indices y' 'collects' all positive entries of v, y'' does the opposite and thus for $d \geq 2$ we have $v \cdot y' > v \cdot y''$ witnessing that h can not cover both.

2. i) & iv): If $l \neq m$ then again y' and y'' have only the two entries at position l and m in common and we conclude as in the first case. So let $l = m$ and therefore $y'' = \overline{y'}$. We know that there is at least one negative entry in v, i.e. $k < d$, and thus w.l.o.g. we may assume that $l = m = k + 1$. We now have the following structure of y' and y'':

$$\begin{array}{rcl} y' & = & (\overbrace{1_1, \ldots, 1_k}^{v_i > 0,\ i \leq k}, \overbrace{1_l\ , 0, \ldots, 0_d}^{v_i < 0,\ i > k}) \\ y'' & = & (\underbrace{0_1, \ldots, 0_k, 0_m}_{\sum_{i \leq k+1} v_i = b}, \underbrace{1, \ldots, 1_d}_{\sum_{i > k+1} v_i = b}) \end{array}$$

Of course both, y' and y'', must lie in h. Therefore the first $k+1$ as well as the last $d-k-1$ elements of v must sum up to $b < 0$. We have to consider four different cases depending on the value of k, the number of positive entries of v. For each case we give a vertex $z \in V^d$, $d \geq 4$, which is not covered by h and also has no neighbour in V^d covered by h, thus proving the correctness of the lemma.

We give the vertices to be considered for all four cases but show only the first one in detail. The remaining cases can be proven in a very similar way, using the scheme given above, and are left to the interested reader.

(a) $k = 0$: Let $z = (1, \ldots, 1, 1, 0)$. Since in this case v has only negative entries and $v_1 = v_l = b$ we get $v \cdot z = b + \sum_{i=2}^{d-1} v_i < b$, hence $z \notin h$. There are three different types of neighbours of z, non covered by h: First let $z' = (0, 1, \ldots, 1, 0)$. Then $v \cdot z' = b - v_d > b$, hence $z' \notin h$. Second let $z'' = (1, \ldots, 1, 0_j, 1, \ldots, 1, 0)$, $2 \leq j \leq d-1$. Then $v \cdot z'' = b + \sum_{i=2}^{j-1} v_i + \sum_{i=j+1}^{d-1} v_i < b$, hence $z'' \notin h$. And third let $z''' = (1, \ldots, 1)$. Then $v \cdot z''' = 2b \neq b$ since $b < 0$, and hence $z''' \notin h$. This proves that for $k = 0$ neither z nor a neighbour of z is covered by h.

(b) $1 \leq k \leq d-3$: Let $z = (1, \ldots, 1, 1, 1)$.

(c) $k = d-2$: Let $z = (1, \ldots, 1, 0, 1)$.

(d) $k = d-1$: Let $z = (1, \ldots, 1, 0, 0)$.

The combinations ii) & iii) as well as ii) & iv) are similar to the two former cases and can be handled in the same way. Thus we have shown that in all cases at least one of the vertices

dimension d	m	example hyperplane $h = (a, b)$
1	1	((1),0)
2	1	((-1,1),0)
3	1	((-2,1,1),0)
4	2	((-2,1,1,1),0)
5	2	((-3,1,1,1,1),0)
6	2	((-4,1,1,1,1,1),0)
7	2	((-5,2,1,1,1,1,1),0)
8	2	((-4,-2,1,1,1,1,1,1),0)
9	2	((-5,-2,1,1,1,1,1,1,1),0)
10	≤ 3	((-7,2,1,1,1,1,1,1,1,1),0)

Table 4.2: Hamming distance minimizing hyperplanes.

x, $\overline{x}$ or z, respectively, has Hamming distance at least two to any vertex covered by h, which completes the proof of the lemma. □

It is interesting to note that up to dimension $d = 9$ one can always find a hyperplane $h \in \mathcal{H}_s(\mathcal{C}^d)$ such that any non covered vertex of $\mathcal{C}^d$ has Hamming distance not more than two to the 'closest' vertex covered by h. Thus Theorem 4.15 is tight in this sense for $d \leq 9$. Determining the minimum possible value for general dimensions, i.e. computing

$$m = \min_{h \in \mathcal{H}_s(\mathcal{C}^d)} \left\{ \max_{v \in V^d} \left\{ \min_{x \in h} \{ d_H(v, x) \} \right\} \right\},$$

where $d_H(v, x)$ is the Hamming distance between v and x, seems to be an open problem. Table 4.2 shows the hyperplanes minimizing m for dimensions 1 to 10. Dimension 10 was the first where we could not find a hyperplane guaranteeing $m \leq 2$ although we were testing over 3 million hyperplanes with 'reasonable' normal vectors, suggested by the vectors for dimensions 1 to 9 shown in the table.

Theorem 4.16 *For $d \geq 4$ there are no pending hyperplanes.*

Proof. Suppose to the contrary that h is an pending hyperplane in $\mathcal{R}^d$, $d \geq 4$. From Theorem 4.15 it follows that there exists a vertex $x \in V^d$, not covered by h, which has at least Hamming distance two to any other vertex of V^d covered by h. Therefore adding this vertex to h does not cause parallelism of h to an axis which, by definition of pending hyperplanes, leads to a contradiction. □

Theorem 4.16 shows that if a vertex set $V \subseteq V^d$, $d \geq 4$, can be covered by a skew hyperplane, then there exists a hyperplane $h \in \mathcal{H}_s(\mathcal{C}^d)$ which covers this set (and possibly some additional vertices). Thus it allows us to use the results of [6, 2] and restrict the numerical investigations to the set $\mathcal{H}_s(\mathcal{C}^d)$. On the other hand the huge number of hyperplanes in $\mathcal{H}_s(\mathcal{C}^d)$ for $d \geq 6$ (234688 for $d = 6$ and e.g. about 85 billion for $d = 8$, cf. Chapter 2) shows that it is impossible to use a straightforward implementation for reasonable dimensions. Therefore we give the next lemma which allows us to skip parts of the exhaustive search of 'hopeless' hyperplane combinations if we consider the hyperplanes in non increasing order of the number of vertices they cover.

Lemma 4.8 *For a fixed set of $C(d) = m$ hyperplanes, denote with $c'_k(d)$ the maximum number of vertices of $\mathcal{C}^d$ covered by a subset of size $k \leq m$. Then*

$$a)\ c'_{k+1}(d) - c'_k(d) \leq c'_k(d) - c'_{k-1}(d) \text{ for } k = 2, \dots, m-1$$
$$b)\ c'_{k+1}(d) - c'_k(d) \geq \frac{2^d - c'_k(d)}{m-k} \qquad \text{for } k = 1, \dots, m-1$$

Proof. Starting with a $(k+1)$-covering remove the hyperplane with the lowest number of own vertices from the set. This clearly leads to the best k-covering of this set. Of course the number of vertices belonging to a remaining hyperplane thus can not decrease. Again removing the hyperplane with the lowest number of it's own vertices leads to the best $(k-1)$-covering. Since this hyperplane has at least as much vertices as the previous one, inequality a) follows immediately. Thus the number of vertices covered by a single (additional) hyperplane for its own is not increasing. After having a set of k hyperplanes the $2^d - c'_k(d)$ remaining vertices have to be covered by $m-k$ hyperplanes and b) follows as a straight forward consequence. □

Note that it is an interesting open problem if the lemma holds for general $c_k(d)$ since $c_k(d)$ and $c_{k-1}(d)$ are often attained by different sets. For example the only type of a 4-covering of $\mathcal{C}^5$ consists of four disjoint hyperplanes, each covering 8 vertices, see Section 4.4. Of course no subset of three of these hyperplanes can realize $c_3(5) = 28$. If we restrict the numerical investigations for dimension 7 to certain subsets of $\mathcal{H}_s(\mathcal{C}^7)$, namely each set consisting of one arbitrary hyperplane of $\mathcal{H}_s(\mathcal{C}^7)$ and the remaining hyperplanes covering 35 vertices each, we get $c_3(7) = 100$, $c_4(7) = 109$ and $c_5(7) = 120$, respectively. Therefore for this subset $c_5(7) - c_4(7) > c_4(7) - c_3(7)$, again indicating that a generalization of Lemma 4.8 may not hold.

We now use the previous results to determine $C(d)$ and $c_k(d)$ for dimension 6 and 7. To this end we did some numerical investigations. From the huge number of hyperplanes in $\mathcal{H}_s(\mathcal{C}^d)$ and their combinations, it is clear that we have to exploit theoretical results. Especially Lemmas 4.3, 4.4, 4.8 b) and Theorems 4.7 and 4.5 proved useful for this purpose.

Trivial but still useful is in addition the following observation. From Lemma 4.4 we can w.l.o.g. assume that the origin is only covered by exactly one hyperplane. Moreover for each different type we need only to consider one hyperplane through the origin, since if there exists a covering including a certain type of hyperplane then for any specific hyperplane of the same type there exists a similar covering obtained by rotation and reflection. Thus we may split $\mathcal{H}_s(\mathcal{C}^d)$ into two subsets. For each type of hyperplane the first set contains one representative which covers the origin. The second set contains all hyperplanes from $\mathcal{H}_s(\mathcal{C}^d)$ which avoid the origin. Thus for any covering of $\mathcal{C}^d$ by k hyperplanes an equivalent covering can be obtained by combining one hyperplane from the first set with $k-1$ hyperplanes from the second set. This method avoids most of the multiple computations of equivalent (but rotated or reflected) coverings and thus speeds up the process. Furthermore the distinction into these two sets will also prove useful for the case analysis in the next section.

A full enumeration of all combinations can be done in reasonable time only for dimensions up to 5. For $d \geq 6$ only certain subsets can be examined by the program. The following section shows how this interaction between theory and computation can be used to obtain results for dimensions 6 and 7.

no. of vertices	skew	cover origin	different types	avoid origin
20	32	10	1	22
15	448	105	3	343
14	480	105	2	375
12	2240	420	4	1820
11	2688	462	7	2226

Table 4.3: Hyperplanes $\in \mathcal{H}_s(\mathcal{C}^6)$ covering 11 to 20 vertices.

4.6.1 Dimension 6

Table 4.3 summarizes results from [2] on hyperplanes in $\mathcal{H}_s(\mathcal{C}^6)$ covering 11 to 20 vertices. The table lists their number, the number of hyperplanes which cover the origin, the number of their different types and the number of hyperplanes which avoid the origin.

We now use the result listed in Table 4.3 to derive $c_k(6)$ and $C(6)$. To this end we give a case analysis to exclude irrelevant hyperplane sets. The remaining combinations can easily be checked in reasonable time by the program mentioned above.

Lemma 4.9 *$c_3(6) = 50$, $c_4(6) = 60$, $c_5(6) = 64$.*

Proof. We first show $c_3(6) = 50$. A lower bound is given by Theorem 4.9 thus suppose $c_3(6) \geq 51$. From Table 4.3 we know that only sets of two hyperplanes which cover 20 vertices each and a hyperplane which covers at least 11 vertices can cover more than 50 vertices of $\mathcal{C}^6$. It remains to check all combinations of a hyperplane which covers at least 11 vertices including the origin (17 possible hyperplanes with unique orientation) and two hyperplanes which cover 20 vertices avoiding the origin (22 possible hyperplanes each). Thus we have a total of 8228 combinations which easily can be checked in reasonable time. None of them covers more than 49 vertices witnessing that $c_3(6) = 50$.

We next show $c_4(6) = 60$. A lower bound is e.g. provided by Theorem 4.11 a) or by Theorem 4.10. It is interesting to note that the two bounds provide different sets of hyperplanes. Especially the set derived from Theorem 4.10 consists of 4 asymmetric skew hyperplanes, each covering 15 own and thus a total of 60 vertices of $\mathcal{C}^6$.

From Lemma 4.3 we know that if a 4-covering of $\mathcal{C}^6$ covers more than 60 vertices it must cover the whole 6-cube. Suppose we can cover $\mathcal{C}^6$ by 4 skew hyperplanes. Since $c_3(6) = 50$ the fourth hyperplane and thus by Lemma 4.8 all hyperplanes of the set have to cover at least 14 own vertices. Therefore only hyperplanes which cover 14, 15 and 20 vertices are of interest. Moreover we can conclude that at least one hyperplane must cover 20 and one hyperplane ≥ 15 vertices, respectively. From Table 4.3 we have 22 hyperplanes covering 20 vertices and avoiding the origin, 365 hyperplanes covering ≥ 15 vertices and avoiding the origin, 740 hyperplanes covering ≥ 14 vertices and avoiding the origin and 6 hyperplanes covering ≥ 14 vertices including the origin. This gives a total of 35653200 combinations to check. The results of the program show that non of these combinations covers more than 60 vertices, thus $c_4(6) = 60$.

The result for $c_5(6)$ follows from the next theorem. $\square$

$d \setminus k$	1	2	3	4	5	6	7	8	9
1	1	cc							
2	2	cc							
3	3	6	cc						
4	6	12	14	cc					
5	10	20	28	cc					
6	20	40	50	60	cc				
7	35	70	100	120	124 *or* 128	cc			
8	70	140	182	224	≥ 244 ≤ 256	252 *or* 256	cc		
9	126	252	364	448	≥ 464 ≤ 512	≥ 492 ≤ 512	508 *or* 512	cc	
10	252	504	≥ 672 ≤ 728	≥ 840 ≤ 896	≥ 912 ≤ 1024	≥ 984 ≤ 1024	≥ 1002 ≤ 1024	1020 *or* 1024	cc

Table 4.4: k-cover numbers $c_k(d)$ (cc = complete coverage, that is $c_k(d) = 2^d$).

Theorem 4.17 $C(6) = 5$.

Proof. A lower bound follows from the previous lemma and Theorem 4.13 provides $C(6) \leq 5$. □

4.6.2 Dimension 7

Lemma 4.10 $c_3(7) = 100$, $c_4(7) = 120$.

Proof. An upper bound is given by Lemma 4.5 a) and the results for $d = 6$. For $c_3(7)$ the upper bound coincides with the lower bound of Theorem 4.9, and similarly for $c_4(7)$ it coincides with the lower bound of Theorem 4.10. □

Theorem 4.13 shows that $C(7) \leq 6$. Therefore the only remaining question for dimension 7 is to determine the value of $c_5(7)$, either 124 or 128, and thus whether $C(7)$ is 5 or 6.

Table 4.4 shows the known results for $c_k(d)$ for $d = 1, \ldots, 10$. For $d \leq 5$ these results are based on the presented lemmas and theorems, while for $d = 6, \ldots, 10$ we give some bounds derived from either theoretical or numerical results.

4.6.3 Asymmetric and Symmetric Coverings

To allow the restriction of numerical investigations to asymmetric and symmetric hyperplanes we give a special version of Theorem 4.16.

Corollary 4.6 *For $d \geq 4$ there are no symmetric pending hyperplanes in $\mathcal{R}^d$. For $d \geq 3$ there are no asymmetric pending hyperplanes in $\mathcal{R}^d$.*

$d \setminus k$	1		2		3		4		5		6		7	
1	1		cc											
	1	–	cc											
2	2		cc											
	1	2	2	cc	3		cc							
3	3		6		cc									
	3	2	6	4	7	6	cc	cc						
4	6		12		14		cc							
	4	6	8	12	12	14	cc	cc						
5	10		20		28		cc							
	10	8	20	16	25	24	30	cc	cc					
6	20		40		50		60		cc					
	15	20	30	40	45	50	60	60	62	cc	cc			
7	35		70		100		120		124 or 128		cc			
	35	30	70	60	91	90	112	120	$\geq$122	124	$\geq$126	cc	cc	

Table 4.5: k-cover numbers $c_k(d)$ compared to coverings with asymmetric ($a_k(d)$) and symmetric ($s_k(d)$) hyperplanes.

Proof. Suppose there exists a symmetric pending hyperplane $h \in \mathcal{R}^d$ for $d \geq 4$. Following the lines of the proof of Theorem 4.16 we can add vertices of V^d to h to attain a non pending hyperplane h'. Since h was symmetric it covered at least one antipodal pair $x, \overline{x}$ which is still covered by h'. Thus by definition h' is a symmetric non pending hyperplane covering all vertices of h, a contradiction.

From the relation between symmetric and asymmetric hyperplanes given in Chapter 2 (cf. the remarks in Section 4.4 and especially the proof of Theorem 4.5) we can conclude that if there does not exist a symmetric pending hyperplane in $\mathcal{R}^d$ there can not exist an asymmetric pending hyperplane in $\mathcal{R}^{d-1}$. □

In Table 4.5 we list the k-cover numbers for asymmetric ($a_k(d)$) and symmetric ($s_k(d)$) hyperplanes. Entries there have the form $\begin{array}{|c|c|}\hline \multicolumn{2}{|c|}{c_k(d)} \\ \hline a_k(d) & s_k(d) \\ \hline \end{array}$. As mentioned in Section 4.3 many of the derived results for the k-cover number $c_k(d)$ also hold in the restricted case of asymmetric and symmetric coverings.

4.7 Open Problems

There are several open questions concerning the presented topic. The most challenging one is to determine the asymptotic behaviour of $C(d)$. Directly related is the question of whether there exists some $d \geq 5$ such that all vertices of $\mathcal{C}^d$ can be covered by at most $d-2$ skew hyperplanes. In Theorem 4.13 we have shown that for $d \geq 5$ strictly less than d skew hyperplanes are sufficient to cover $\mathcal{C}^d$. We gave the exact value of $C(d)$ for d up to 6 but $C(7)$ it is still unsettled. Similar questions are open when the covering sets are restricted to either asymmetric or symmetric hyperplanes, respectively. For example it also remains to determine

$a_5(7)$ and $a_6(7)$.

Chapter 5

Slicing the Hypercube

5.1 Slicing Hyperplanes

In the previous chapters we were mainly interested in hyperplanes spanned by the vertices of $\mathcal{C}^d$. We even considered sets of hyperplanes which avoided the interior of edges of $\mathcal{C}^d$, the hull-honest hyperplanes. To the contrary in this chapter we investigate hyperplanes which avoid the vertices of the d-cube and we are only interested in the edges of $\mathcal{C}^d$ they intersect. Motivation for considering such hyperplanes mainly stems from learning theory, where they are used to linearly separate subsets of hypercube vertices.

A *slicing* hyperplane $h = (a, b)$ is a hyperplane in $\mathcal{R}^d$ which properly intersects the d-dimensional hypercube $\mathcal{C}^d$ but avoids to cover any vertices of $\mathcal{C}^d$. Therefore h splits the vertex set of $\mathcal{C}^d$ into two parts, one lying "above" and the other "below" h. We call h a $m : n$ slicing hyperplane if m and n, respectively, denote the cardinality of these subsets. Clearly there are infinitely many hyperplanes which split the vertices into the same subparts. For any such set we will only consider one representative and call this the slicing hyperplane. If we are only interested in the way how the edges of the hypercube are intersected or how the vertex set is partitioned, it does not matter which hyperplane of the set we choose.

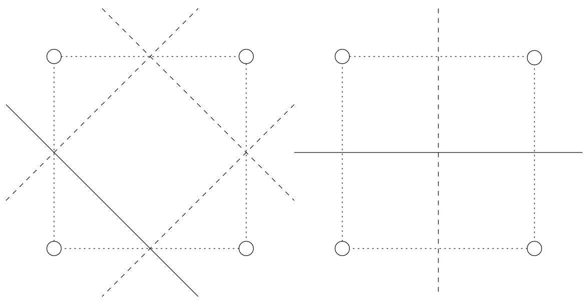


Figure 5.1: Six "hyperplanes" of two different types, slicing $\mathcal{C}^2$.

From Figure 5.1 it can be seen that there are six slicing hyperplanes for $\mathcal{C}^2$. By rotation and reflection of the hypercube (square in this case) there are only two different types of hyperplanes, namely splitting the vertices $1 : 3$ and $2 : 2$, respectively.

In this section we will give the number of slicing hyperplanes and different classes for $d = 1, \ldots, 8$. To this end we denote with $H(d)$ the number of slicing hyperplanes in $\mathcal{C}^d$,

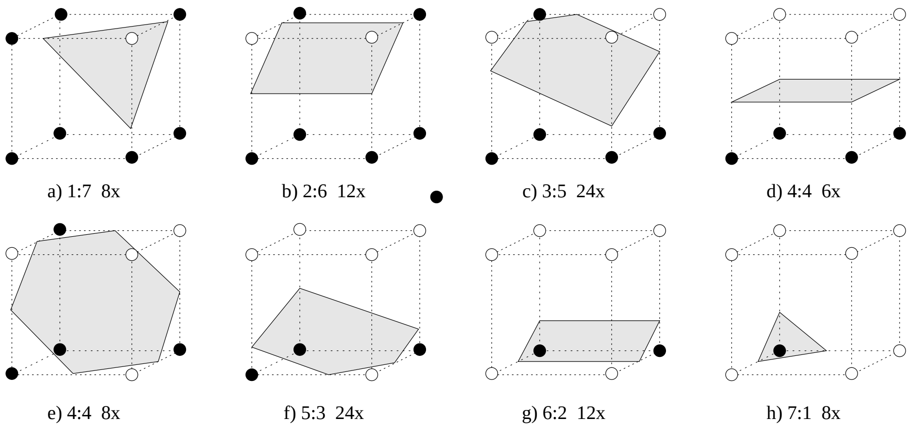


Figure 5.2: Different ways to linearly separate the vertices of the 3-cube.

e.g. $H(2) = 6$. A *class* of slicing hyperplanes includes all hyperplanes which intersect the same hypercube edges, up to reflections and rotations of the hypercube (see e.g. Chapter 3 for a similar definition of equivalence classes and a formally description of the reflection and rotation operations). Thus the number $N(d)$ of classes reflects the number of combinatorially different slicing hyperplanes. Note that therefore a hyperplane splitting the vertex set $m : n$ might belong to the same class as a $n : m$-hyperplane. If the vertices of $\mathcal{C}^d$ are partitioned symmetrically, i.e. $2^{d-1} : 2^{d-1}$, then we call a class symmetrical. Their number is denoted by $N_s(d)$. For the 2-dimensional case we get $N(2) = 2$ and $N_s(2) = 1$.

In addition we consider the number of different ways to get a subset of hypercube vertices by a linear separator, i.e. a hyperplane. This number is equivalent to two times the number of non-symmetric slicing classes, plus the number of symmetric slicing classes, plus two for the sets containing no or all vertices, respectively. We call these classes *monotone* classes and denote their number by $M(d)$.

To clarify the notation let us consider Figure 5.2 which shows the classes for $\mathcal{C}^3$. For every class the number of vertices "above" and "below" the hyperplanes are given. In addition the last number shows how many realizations of such a hyperplane exist.

There are two different types of symmetric hyperplanes, d) and e), and three asymmetric types, splitting the cube 1:7 (a) and h)), 2:6 (b) and g)) and 3:5 (c) and f)), respectively. Thus $N(3) = 5$ and $N_s(3) = 2$. To determine $H(d)$ we have to take into account that for example Figure 5.2a) and f) give the same sets of slicing hyperplanes. Moreover for the symmetrical cases d) and e) we take only half of the hyperplanes, since we do not care about their orientation. Thus we take the hyperplanes of the classes a) to c) and half of the hyperplanes of d) and e), resulting in a total number of $H(d) = 51$ slicing hyperplanes.

Together with the empty set and the set of all vertices of $\mathcal{C}^3$ the eight hyperplanes of Figure 5.2 give the 10 monotone classes, i.e. $M(3) = 10$. (Note that to simplify the visualization we moved the origin to the upper right vertex in front.)

In the literature of threshold logic and threshold gates $M(d)$ is called the "number of representative positive threshold functions of d or fewer variables", see e.g. Chapter 7 in [76]. In Table 7.5.1 of this book $M(d)$ is given for $d = 1, \ldots, 5$. Every slicing hyperplane gives rise to two linear threshold functions, since either the vertices above or below the hyperplane

d	classes $N(d)$	symm. classes $N_s(d)$	monotone classes $M(d)$	No. of slicing hyperplanes $H(d)$	threshold functions with d variables $T(d)$
1	1	1	3	1	4
2	2	1	5	6	14
3	5	2	10	51	104
4	14	3	27	940	1 882
5	62	7	119	47 285	94 572
6	566	21	1 113	7 514 066	15 028 134
7	14 754	135	29 375	4 189 035 431	8 378 070 864
8	1 366 317	2 470	2 730 166	8 780 769 776 472	17 561 539 552 946

Table 5.1: Cardinality of different classes of slicing hyperplanes, their number and the number of threshold functions.

may be "accepted" by the function. In addition there are two threshold functions without corresponding slicing hyperplane, namely either accepting all vertices of $\mathcal{C}^d$, or none of them. We denote the number of threshold functions by $T(d)$ and get $T(d) = 2H(d) + 2$. Figure 5.2 shows the 102 'non-trivial' threshold functions, and therefore $T(3) = 104$. See for example Chapter 9 in [76] for a discussion on asymptotic upper and lower bounds on $T(d)$.

For dimension $d = 1, \ldots, 8$ Table 5.1 gives the cardinality of different classes, the number of slicing hyperplanes, and the number of threshold functions. The last numbers can also be found in Table 9.4.3 of [76]; see also [78] for related tables in the context of threshold functions. To compute the table we used observations and methods similar to the one described for hyperplanes spanned by vertices of $\mathcal{C}^d$ (see Chapter 2) and 0/1-polytopes (Chapter 3). Computing the table for $d = 1, \ldots, 8$ takes about one hour on a standard Personal Computer. Nevertheless a complete enumeration for higher dimensions seems currently not to be possible within reasonable time.

5.2 Slicing Sets

Similar to the covering problems of Chapter 4 we now consider the question of slicing numbers, which was first stated by O'Neil [81]: what is the size, $S(d)$, of a minimum set of hyperplanes that slices all $d2^{d-1}$ edges of the hypercube? Such a set of hyperplanes is called an *edge slicing set* and $S(d)$ is called the *slicing number*. Similar to the k-cover numbers we also ask how many edges of $\mathcal{C}^d$ are sliced by k hyperplanes and denote this number by $s_k(d)$, called the *k-slicing number*. Obviously $S(d) = \min\{k | s_k(d) = d2^{d-1}\}$. See [92] for a nice survey on this topic and Section 4.3 of Chapter 4 for a relation to the Covering number $C(d)$. For a complete characterization of all possible slicing hyperplanes of $\mathcal{C}^4$ see [39], which also gives some pointers to the literature. In addition this paper shows $S(4) = 4$, which coincides with our numerical observations. For a single hyperplane O'Neil [81] showed that $s_1(d) = (d - \lfloor \frac{d}{2} \rfloor)\binom{d}{\lfloor \frac{d}{2} \rfloor}$; confer this to the maximal number of covered vertices given in Chapter 4.

For the slicing problem a simple method of adding two extended hyperplane sets for lower

$d \setminus k$	1	2	3	4	5
1	1				
2	2	4			
3	6	10	12		
4	12	24	30	32	
5	30	54	70...78	78..80	80

Table 5.2: k-slicing numbers $s_k(d)$.

dimensions can be used to show its subadditivity, i.e. that $S(d_1 + d_2) \leq S(d_1) + S(d_2)$ [92]. Therefore improvements for specific low dimensions immediately yield better constructions for higher dimensions. In fact a slicing set for $\mathcal{C}^6$ of size 5 can be found in Section 3.3 of [92]. Therefore it follows that $S(d) \leq \lceil \frac{5d}{6} \rceil$, $d \geq 1$. On the other hand the huge number of slicing hyperplanes of $\mathcal{C}^d$ for $d \geq 5$ makes it very difficult to compute $S(d)$ and $s_k(d)$. Table 5.2 gives some results for $d = 1, \ldots, 5$. Although we use a similar approach as for covering sets (cf. Section 4.6 of Chapter 4) it seems not be be possible to perform a complete enumeration of all slicing sets for dimensions higher than $d = 5$. The most challenging open problem here is to determine whether $S(5) = 4$, since this would immediately improve the upper bound. However, from our numerical investigations we conjecture that $s_4(5) = 78$ and thus $S(5) = 5$.

5.3 Stratified Depth-2 Threshold Gates

Analogous to the relation between slicing hyperplanes and threshold functions there is a relation between slicing sets and simple "neural" networks which we will point out in this section. We will give only an informal definition of a simple version of a neural network here. For further details we refer the interested reader to the literature, e.g. [76, 82, 85, 101].

A *stratified depth-2 threshold gates* network consists of $k+1$ linear threshold gates, arranged in two layers. All k gates of the first layer are connected to the input lines. The second layer consists only of a single threshold gate and its input is just the output of the first layer. The output of the second layer reflects the "yes" or "no" answer of the net. Note that there are no direct connections from the input to the output gate.

A prominent problem for neural networks is to compute the parity of d binary inputs. It is well known that even for $d = 2$ this can not be done by a single linear threshold gate. Therefore a stratified depth-2 threshold gates network might be used to solve the parity problem. Clearly the gates of the first layer have to separate inputs with different parity. In terms of hyperplanes and hypercubes this means that the threshold functions of the input layer correspond to a slicing set for $\mathcal{C}^d$. Thus the cardinality of slicing sets is a lower bound for the number of gates in the first layer.

A simple slicing set of size d can be obtained by taking hyperplanes normal to the main diagonal, each separating the vertices of different levels. If we invert the "above-below" orientation of every other plane then it is not hard to see that the parity of the vertices can be computed by a single 'output gate'. We just have to compute the difference of the numbers of hyperplanes a vertex is above and below, respectively. Observe that this difference is either one or zero. Thus $d+1$ linear threshold gates are sufficient to solve the parity problem by a stratified depth-2 threshold gates network. The question is now, whether it is possible to use a slicing set with strictly less than d hyperplanes for the first layer. Unfortunately it turned out that the example for $d=6$ of size 5 given by Saks [92] is not suitable. There are vertices with different parity which are not separated by this slicing set. This happens to be because these vertices are e.g. at Hamming distance 3 from each other. Moreover even if such vertices are correctly separated it might not be possible to "interpret" the output of the first layer by a single gate of the second layer, i.e. the resulting "parity-assignment" of the slicing set might not be linearly separable.

Although we investigated a huge number of slicing sets of size $d-1$ for $d=6,\ldots,7$ numerically we could not find a suitable set. As far as we know it thus remains an open problem whether a stratified depth-2 threshold gates network with less than $d+1$ gates can compute the parity function of d binary inputs.

Note that if we use a non-stratified network, i.e. we allow connections from the input to the output node, or if we use more than two layers, then we can reduce the number of necessary gates, see e.g. [60, 101, 102].

5.4 Hypercube Intersection Figures

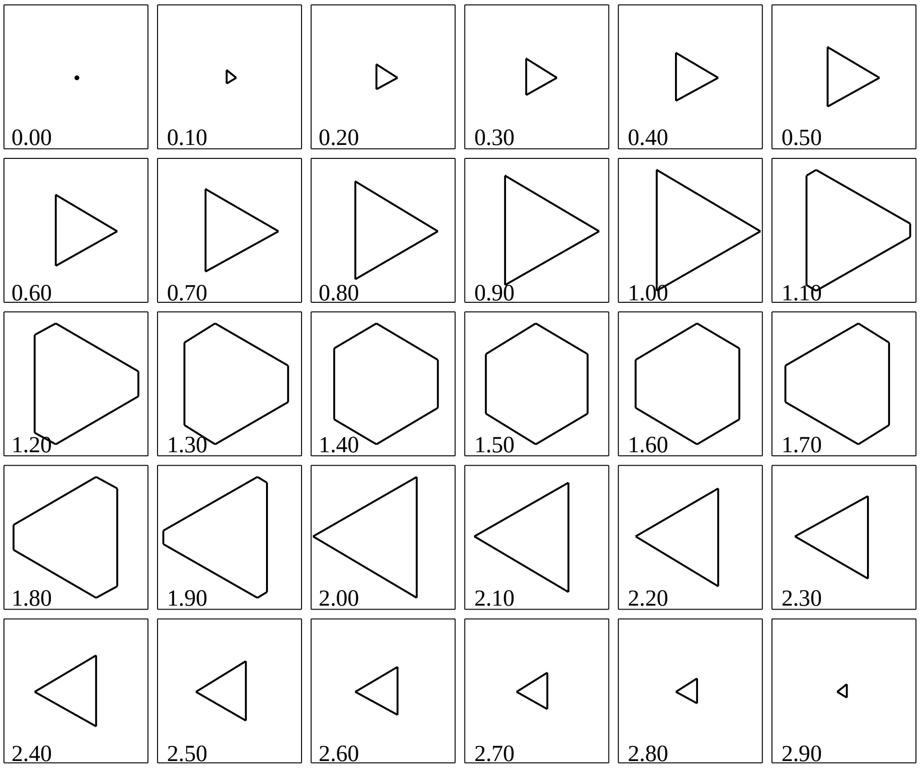


Figure 5.3: Slices of the 3-cube obtained from planes normal to the main diagonal.

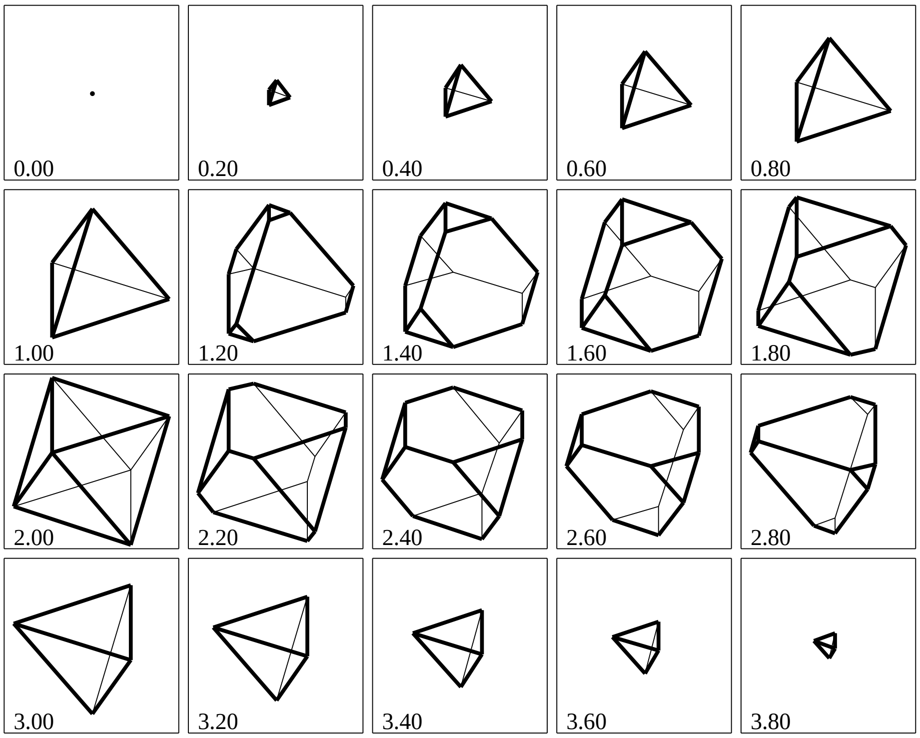


Figure 5.4: Slices of the 4-cube obtained from hyperplanes normal to the main diagonal.

For low dimensional hypercubes the intersection figures of cutting hyperplanes (in contrary to a slicing hyperplane a cutting hyperplane might also cover vertices) yield sometimes beautiful geometric figures. A sequence of intersections, obtained from moving the hyperplane along its normal vector, gives some insight into the geometry of such slices. Figures 5.3 and 5.4 give slices obtained from cutting hyperplanes normal to the main diagonal of $\mathcal{C}^3$ and $\mathcal{C}^4$, respectively. The hyperplanes are of the form $h = ((1, \ldots, 1), b)$ where the numbers below the figures give the value of b.

Half way through the cube we obtain very interesting intersection figures, namely a regular hexagon for the 3-cube and a regular octahedron for the 4-cube. A nice animation of different projections and intersections of the 4-cube can be seen in the Video of Banchoff and Strauss [18]. Similar sequences as in Figure 5.4 are also given in [19] for educational purposes.

A theorem by Naumann [80] shows that slicing the hypercube can yield various different polytopes: Every $(d-1)$-polytope is obtainable as the intersection of a $(d-1)$-dimensional hyperplane and a hypercube of sufficiently high dimension. This also gives some evidence for the intrinsic complexity of enumerating all slicing hyperplanes or slicing sets.

An interesting other question asks whether there always exists a hyperplane that meets the interior of every facet of the d-cube. Contrary to the impression for $d = 1$ and $d = 2$ the answer is “yes” for $d \geq 2$, [62]. An example for such a hyperplane is $h = ((1, \ldots, 1)\frac{d}{2})$, cf. the central slicing of Figures 5.3 and 5.4.

Let us finally mention two related, simple looking questions, which were only recently solved. The first was settled in 1986 by Ball [15] (see also [16]): Which hyperplane intersects the d-cube in a $(d-1)$-polytope of maximum volume? The answer is that every

$(d-1)$-dimensional section of $\mathcal{C}^d$ has volume at most $\sqrt{2}$. An obvious example achieving this maximum is the $(d-2)$-parallel hyperplane $h = ((1,1,0,\ldots,0),1)$.

A hyperplane going through the center of the hypercube generates a so called *central* slice of the d-cube. The second question now asks for a lower bound on the volume of such a central slice. For $d=2$ the length of any central intersecting line is clearly at least one, but even for the 3-cube the answer is not obvious. Actually it can be shown that every central slice of the d-dimensional unit cube has a volume of at least 1, cf. [55]. See also [17] for a relation to central sections of d-dimensional Euclidean balls. There the surprising result that all central sections of $\mathcal{C}^d$ have smaller volume than those of the Euclidean ball of volume 1 is proven for $d \geq 10$. This is related to the well known paradoxon of the decreasing volume of the unit ball for dimensions greater than 5; see also the discussion about AESA in Section 8.1.2.

Chapter 6

Distance Problems on the Hypercube

6.1 Introduction

In this Chapter we consider various problems concerning the Hamming distance between vertices of the hypercube. Recall that the *Hamming distance* $d_H(x, y)$ of two vertices $x, y \in V^d$ is the number of coordinates where x differs from y. Also recall that with $\mathcal{H}(\mathcal{C}^d)$ we denote the set of all hyperplanes that can be spanned by d affinely independent vertices of $\mathcal{C}^d$. As mentioned in the introduction it follows from linear algebra (Cramer's rule) that there is always an integer normal vector v provided $h = (v, b) \in \mathcal{H}(\mathcal{C}^d)$. Moreover let $\mathcal{H}_s(\mathcal{C}^d)$ be the subset of skew (non-axis-parallel) hyperplanes of $\mathcal{H}(\mathcal{C}^d)$.

The following problems are investigated:

EMH **EXTERNAL MINIMIZING HYPERPLANE**
QUESTION: Find a skew hyperplane $h \in \mathcal{H}_s(\mathcal{C}^d)$ such that the maximum of all minimum Hamming distances from any vertex $v \in V^d$ to a vertex x covered by h is minimized.

That is computing $\min_{h \in \mathcal{H}_s(\mathcal{C}^d)} \{\max_{v \in V^d} \{\min_{x \in h} \{d_H(v, x)\}\}\}$.

IMH **INTERNAL MAXIMIZING HYPERPLANE**
QUESTION: Find a hyperplane $h \in \mathcal{H}(\mathcal{C}^d)$ such that the minimum Hamming distance between vertices covered by h is maximized.

That is computing $\max_{h \in \mathcal{H}(\mathcal{C}^d)} \{\min_{x,y \in h, x \neq y} d_H(x, y)\}$

MVS **MAXIMIZING VERTEX SET**
QUESTION: Find a vertex set $S \subseteq V^d$ of size n such that the minimum Hamming distance between vertices of S is maximized.

That is computing $\max_{S \subseteq V^d, |S|=n} \{\min_{x,y \in S, x \neq y} d_H(x, y)\}$

A variant might ask for computing the largest set $S \subseteq V^d$ such that the minimum Hamming distance between vertices of S is not smaller than a given constant h, i.e. to compute $\max_{S \subseteq V^d} \{|S| : \min_{x,y \in S, x \neq y} d_H(x, y) \geq h\}$.

MNN **MAXIMUM NEAREST NEIGHBOUR**
QUESTION: For how many vertices of $\mathcal{C}^d$ can a vertex $v \in \mathcal{C}^d$ be simultaneously the (unique) nearest neighbour?

That is computing $\max_{v \notin S \subseteq V^d} \{|S| : \forall x, y \in S, x \neq y : d_H(x, v)[\leq \mid <]d_H(x, y)\}$.

Problem EMH was considered in Chapter 4 in Section 4.6. In Theorem 4.15 we showed that for any skew hyperplane h in $\mathcal{R}^d$, $d \geq 4$, there exists a vertex $v \in V^d$ which has Hamming distance at least 2 to any vertex $x \in V^d$ covered by h. Moreover we showed that for $d \leq 9$ such hyperplanes always exist, i.e. for $4 \leq d \leq 9$ the optimum is 2, see Table 4.2.

Note that the restriction to hyperplanes from $\mathcal{H}_s(\mathcal{C}^d)$ is no limitation to the optimum, since clearly no pending hyperplane can achieve a better minimum.

The distance function of this problem can also be seen as a binary version of the directed *Hausdorff distance* $\tilde{\delta}_H(A, B) := \max_{a \in A} \min_{b \in B} \|a - b\|$, see e.g. [87]. Here A is the set of all vertices of $\mathcal{C}^d$ and B is the set of the vertices covered by a hyperplane $h \in \mathcal{H}(\mathcal{C}^d)$. The underlying metric is the L_1 distance, i.e. $\|a - b\| = d_H(a, b)$.

The inverse problem, finding a hyperplane which maximizes the minimum Hamming distance between vertices of $\mathcal{C}^d$ and covered vertices, is obviously trivial. We might just take a hyperplane which covers only one vertex. Even with the restriction to the set $\mathcal{H}_s(\mathcal{C}^d)$ we get the maximum Hamming distance of $d - 1$ e.g. by the hyperplane $((1, \ldots, 1), 1)$, $d \geq 3$.

We will investigate the remaining three problems in the following sections.

6.2 Hyperplanes Maximizing the Hamming Distance Between Covered Vertices

In this section we consider the minimal Hamming distance between vertices of $\mathcal{C}^d$ lying in a given hyperplane, i.e. the problem

IMH **INTERNAL MAXIMIZING HYPERPLANE**

QUESTION: Find a hyperplane $h \in \mathcal{H}(\mathcal{C}^d)$ such that the minimum Hamming distance between vertices covered by h is maximized.

That is computing $\max_{h \in \mathcal{H}(\mathcal{C}^d)} \{\min_{x,y \in h, x \neq y} d_H(x, y)\}$

To this end, for a hyperplane h let $min_H(h) := \min_{x,y \in h, x \neq y} d_H(x, y)$. Clearly $min_H(h) = 1$ if and only if h is an axis-parallel hyperplane. In the following discussion we give some results on $min_H(h)$ for skew hyperplanes.

Lemma 6.1 *For a skew hyperplane $h = (a, b) \in \mathcal{R}^d$ let $x, y \in h$ be two vertices covered by h with $d_H(x, y) = k$. Then there exists an index set $I \subseteq \{1, \ldots, d\}$ of size k and a vector $s \in \{0, 1\}^d$ such that $\sum_{i \in I} (-1)^{s_i} a_i = 0$.*

Proof. The vector $z = x - y \in \{-1, 0, 1\}$ has exactly k non-zero entries and since $x \cdot a = b = y \cdot a$ we get $z \cdot a = (x - y) \cdot a = x \cdot a - y \cdot a = 0$. Let now I contain all indices of non-zero entries of z and set $s_i = 1$ if and only if $z_i = -1$. Thus we have $\sum_{i \in I} (-1)^{s_i} a_i = z \cdot a = 0$. □

Note that the conversion of this lemma is not true as can be seen by the hyperplane $h = ((1, 2, 3, 4), 5)$. Although with the vector (-1,-1,1,0) the first three elements sum up to zero, h contains only the vertices $x = (1, 0, 0, 1)$ and $y = (0, 1, 1, 0)$ with $d_H(x, y) = 4$.

Clearly with the same construction we can always obtain a hyperplane in $\mathcal{R}^d$ with $min_H(h) = d$. On the other hand this hyperplanes seem rather "unnatural" since they contain only two vertices of C^d. Thus we will restrict further investigations to the more interesting subset of hyperplanes in $\mathcal{H}(\mathcal{C}^d)$.

Theorem 6.1 *Let $h = (a, b) \in \mathcal{H}(\mathcal{C}^d)$ be a skew hyperplane.*

$$min_H(h) = 2 \iff \exists\, i, j, 1 \leq i < j \leq d : |a_i| = |a_j|$$

Proof. The "only if" part follows immediately from Lemma 6.1 for k=2. It remains to show that if $|a_i| = |a_j|$ for some i and j, $1 \leq i < j \leq d$, we can always find two vertices x and y covered by h with $d_H(x, y) = 2$.

W.l.o.g. and to simplify notation let $i = 1$ and $j = 2$, i.e. $|a_1| = |a_2|$. Moreover we consider only hyperplanes through the origin, i.e. $h = (a, 0)$. Any hyperplane $h = (a, b)$ can be transformed to such a hyperplane by simple reflections without changing the absolute value of the normal vector entries or the Hamming distance between covered vertices of C^d. (For details on reflections see also the previous sections.)

If $a_1 = -a_2$ then the two vertices $(0, \ldots, 0)$ and $(1, 1, 0, \ldots, 0)$ are both covered by h and we have $min_H(h) = 2$. Suppose now $a_1 = a_2$. If h covers at least one vertex of the form $(0, 1, x_3, \ldots, x_d)$ it clearly also covers a vertex $(1, 0, x_3, \ldots, x_d)$ and again we are done.

Thus assume h covers only vertices of the form $(0, 0, x_3, \ldots, x_d)$ and $(1, 1, y_3, \ldots, y_d)$ from C^d. We will show that in this case h can not belong to $\mathcal{H}(\mathcal{C}^d)$.

Set $h' = ((-a_3, 2a_1 + a_3, a_3, \ldots, a_d), 0)$. All vertices from C^d of the form $(0, 0, x_3, \ldots, x_d)$ covered by h are also covered by h' and since $-a_3 + (2a_1 + a_3) = a_1 + a_2$ the same holds for all vertices of the form $(1, 1, y_3, \ldots, y_d)$. But in addition h' covers the vertex $(1, 0, 1, 0, \ldots, 0)$ which by assumption was not covered by h. Thus we conclude that h was not fully determined by the vertices it covered, i.e. $h \notin \mathcal{H}(\mathcal{C}^d)$, completing the proof. □

A simple example shows that the "if part" of Theorem 6.1 is not true for $h \notin \mathcal{H}(\mathcal{C}^d)$. Let $h = ((1, 1, -2), 0)$ covering the vertices $(0, 0, 0)$ and $(1, 1, 1)$ of C^3. Although $|a_1| = |a_2|$ we get $min_H(h) = 3$.

Another consequence of Lemma 6.1 is the following.

Theorem 6.2 *Let $h = (a, b) \in \mathcal{H}(\mathcal{C}^d)$ be a skew hyperplane. If for $1 \leq i < j \leq d$ $|a_i| \neq |a_j|$ and $|a_i|$ is odd for $i = 1, \ldots, d$, then $min_H(h) \geq 4$.*

Proof. From Theorem 6.1 we know that $min_H(h) \geq 3$. Lemma 6.1 states that $min_H(h) = 3$ only if there is a way to choose the signs of three entries of a such that they sum up zero. But this is not possible since a contains only odd entries. □

Numerical investigations have shown that the type of hyperplanes with only odd entries, as used for Theorem 6.2, appears the first time for C^8. (Recall that we consider only hyperplanes from $\mathcal{H}(\mathcal{C}^d)$). An example for C^9 shows that the conversion of Theorem 6.2 is not true: the hyperplane $h = ((-13, -15, -36, 1, 3, 5, 7, 9, 11), 0) \in \mathcal{H}(\mathcal{C}^d)$ contains 9 vertices with $min_H(h) = 4$.

Table 6.1 shows the results on the maximum possible $min_H(h)$ for hyperplanes from $\mathcal{H}(\mathcal{C}^d)$ for dimensions 2 to 10. For $d = 2, \ldots, 8$ we tested all hyperplanes in $\mathcal{H}(\mathcal{C}^d)$, as given in

dimension d	2	3	4	5	6	7	8	9	10
$\max_{h\in\mathcal{H}(\mathcal{C}^d)}\{min_H(h)\}$	2	2	2	2	2	3	4	4	4 or 5

Table 6.1: Maximum possible $min_H(h)$ for hyperplanes from $\mathcal{H}(\mathcal{C}^d)$ for dimensions 2 to 10.

Chapter 2. The lower bounds for $d = 9$ and $d = 10$ are provided by hyperplanes we found by testing subsets of $\mathcal{H}(\mathcal{C}^d)$. On the other hand, to span a hyperplane in $\mathcal{R}^d$ we need at least d vertices of $\mathcal{C}^d$. But Table 6.2 in Section 6.3 gives an upper bound on the minimum Hamming distance between a fixed number n of vertices from $\mathcal{C}^d$. Although Table 6.2 concerns arbitrary vertex sets of $\mathcal{C}^d$ it clearly gives an upper bound on the minimum Hamming distance between vertices covered by a hyperplane, too. It can be seen that the results given in Table 6.1 are tight for $d \leq 9$. For dimension 10 we found a hyperplane $h \in \mathcal{H}(\mathcal{C}^d)$ with $min_H(h) = 4$ and get the upper bound 5 from Table 6.2 (for 10 or more vertices of $\mathcal{C}^{10}$).

For higher dimensions it is an open problem to give tight upper bounds for $min_H(h)$, $h \in \mathcal{H}(\mathcal{C}^d)$. One reason why this seems to be a very hard task is the irregularity and the dramatically increasing cardinality of the sets $\mathcal{H}(\mathcal{C}^d)$, see e.g. Chapter 2. For $d \geq 9$ the cardinality is even unknown. Moreover the exact maximum of normal vector entries of hyperplanes from $\mathcal{H}(\mathcal{C}^d)$ are also unknown; Chapter 2 provides some upper bounds on both quantities. But Theorems 6.1 and 6.2 give rise to the conclusion that the size of the normal vector entries plays an important role in determining upper bounds for $min_H(h)$.

6.3 Hamming Distance Maximizing Vertex Sets

Let $S \subseteq V^d$ be a set of n vertices of the d-dimensional hypercube. With $min_H(S) := \min_{x,y\in S}\{d_H(x,y)|x \neq y\}$ we denote the minimum Hamming distance between vertices of S. In this section we consider the best possible arrangement of these n vertices in S such that no two vertices are to close together, i.e. that they maximize $min_H(S)$.

More formally we consider the following problem:

MVS **MAXIMIZING VERTEX SET**

QUESTION: Find a vertex set $S \subseteq V^d$ of size n such that the minimum Hamming distance between vertices of S is maximized, i.e. computing $\max_{S\subseteq V^d,|S|=n} min_H(S)$, where $|S|$ denotes the cardinality of S.

In a variant we ask for computing the largest set $S \subseteq V^d$ such that the minimum Hamming distance between vertices of S is not smaller than a given constant h, i.e. to compute $\max_{S\subseteq V^d}\{|S| : \min_H(S) \geq h\}$.

For the first problem we denote the maximum Hamming distance with $H(d,n) = \max_S\{min_H(S) : S \subseteq V^d, |S| = n\}$. For the second problem the maximum number of vertices S may contain is denoted by $N(d,h) = \max_S\{|S| : S \subseteq V^d, min_H(S) \geq h\}$.

A first result can be found by summing over all distances between vertices of S.

$d \setminus n$	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1														
2	2	1	1												
3	3	2	2	1	1	1	1								
4	4	2	2	2	2	2	2	1	1	1	1	1	1	1	1
5	5	3	3	2	2	2	2	2	2	2	2	2	2	2	2
6	6	4	4	3	3	3	3	2	2	2	2	2	2	2	2
7	7	4	4	4	4	4	4	3	3	3	3	3	3	3	3
8	8	5	5	4	4	4	4	4	4	4	4	4	4	4	4
9	9	6	6	5	5	4	4	4	4	4	4	4	4	4	4
10	10	6	6	6	6	5	5	5	5	5	5	4	4	4	4

Table 6.2: $H(d,n)$: Maximal $min_H(S)$ for a set of n vertices of $\mathcal{C}^d$.

Lemma 6.2 *Let $S \subseteq V^d$ be a set of size $n \geq 2$. The mean Hamming distance between vertices of S is bounded from above by $\frac{n}{2(n-1)}d$ for n even, and $\frac{n+1}{2n}d$ for n odd.*

Proof. Consider the i-th bit of every vertex of S, for some fixed $i \in \{1,\ldots,d\}$. With $n_0(i)$ we denote the number of vertices in S with a "0" on the i-th bit. Then the sum of all bit differences for the i-th bit is $n_0(i)(n - n_0(i))$, which is maximal for $n_0(i) = \lfloor \frac{n}{2} \rfloor$. Summing over all bit positions $i = 1 \ldots d$ we get $d\lfloor \frac{n}{2} \rfloor \lceil \frac{n}{2} \rceil$ for the maximum sum of Hamming distances in S. Together with the fact that we have $\binom{n}{2}$ distances the lemma follows immediately. □

Clearly the minimum distance between two points in S cannot exceed the mean distance and thus we get

Corollary 6.1 *Let $S \subseteq V^d$ be a set of size $n \geq 2$.*

$$H(d,n) \leq \begin{cases} \lfloor \frac{n}{2(n-1)} d \rfloor & \text{for } n \text{ even} \\ \lfloor \frac{n+1}{2n} d \rfloor & \text{for } n \text{ odd} \end{cases}$$

Corollary 6.1 is obviously tight for $n = 2$ and the following simple example shows that it is also tight for $n = 3, 4$. Note that for an odd number of vertices the upper bound coincides with the bound of the succeeding even number. Consider the four vertices $v_1 = (0,\ldots,0)$, $v_2 = (1,\ldots,1_{\lfloor \frac{2}{3}d \rfloor},0,\ldots,0)$, $v_3 = (0,\ldots,0,1_{\lfloor \frac{d}{3} \rfloor+1},\ldots,1)$, and $v_4 = (1,\ldots,1_{\lfloor \frac{d}{3} \rfloor},0,\ldots,0,1_{\lfloor \frac{2}{3}d \rfloor+1},\ldots,1)$, where x_i denotes the bit on the $i-th$ position. It is not hard to see that their pairwise Hamming distance is at least $\lfloor \frac{2}{3}d \rfloor$ and we thus have

Observation 6.1 *$H(d,2) = d$, $H(d,3) = H(d,4) = \lfloor \frac{2}{3}d \rfloor$.*

The bounds of Corollary 6.1 tend to $\frac{d}{2}$ when n increases, thus it may only provide good estimations for relatively small n (compared to d). E.g. for dimension $d = 5$ and a set of $n = 5$ or $n = 6$ vertices the upper bound for $H(5,5) = H(5,6)$ is 3, but checking all possible arrangements shows that $H(5,5) = H(5,6) = 2$.

Table 6.2 shows some numerical results for $d = 1,\ldots,10$ and $n \leq 16$.

Theorem 6.3 *Let $S \subseteq V^d$ be a subset of hypercube vertices and $h = min_H(S)$ their minimum Hamming distance. Then for $h > \frac{d}{2}$ the size of S is bounded by*

$$N(d,h) \leq 2\lfloor \frac{h}{2h-d} \rfloor$$

Proof. Since we have $\frac{d}{2} < h \leq H(d,n)$ we can rearrange the inequality for even n of Corollary 6.1 to $n \leq \frac{2h}{2h-d}$. With the restriction for $N(d,h)$ to be even we thus get $N(d,h) \leq 2\lfloor \frac{h}{2h-d} \rfloor$. Similarly we get $N(d,h) \leq 2\lfloor \frac{d-h}{2h-d} \rfloor + 1$ for odd cardinality.

Moreover we see that for every set S with an odd number n of vertices and fulfilling the inequality of Corollary 6.1 there also exists an set S' of size $n+1$ which fulfills it. Thus the derived bound will be tight only for even cardinality of S, and Theorem 6.3 follows. □

It is interesting to observe that the last theorem seems to be tight for h even, cf. the numerical results given in Table 6.4. We will consider some special cases below.

The next lemata give some relations between different dimensions.

Lemma 6.3 *Let $S \subseteq V^d$ and $h = \frac{c_1}{c_2}d$, where h, c_1, c_2 are integer and c_2 is a divisor of d. Then with $n = N(c_2, c_1)$*

$$N(d,h) \geq n = N(c_2, c_1)$$

$$H(d,n) \geq \frac{d}{c_2} H(c_2, n)$$

Proof. From $n = N(c_2, c_1)$ it follows that in dimension c_2 we can generate a set $S \subseteq V^{c_2}$ of n vertices such that $min_H(S) \geq c_1$. Since c_2 is a divisor of d we can duplicate every bit of a vertex of S $\frac{d}{c_2}$ times, obtaining a set $S' \subseteq V^d$ with $min_H(S') = \frac{d}{c_2} min_H(S) \geq c_1 \frac{d}{c_2} = h$. Thus $N(d,h) \geq n = N(c_2, c_1)$ and $H(d,n) \geq \frac{d}{c_2} H(c_2, n)$. □

Lemma 6.4 $N(d+1,h) \geq N(d,h) \geq N(d+1,h+1) \geq N(d,h+1)$

Proof. The left and right inequalities are obviously correct since any point set from V^d is also contained in V^{d+1}. For the middle inequality consider a point set $S \subseteq V^{d+1}$ with $min_H(S) = h+1$. For every vertex in S remove its last bit and let $S' \subseteq V^d$ be the set of these vertices. Since the distance between two vertices can not decrease by more than 1 we have $min_H(S') \geq h$ and thus the inequality follows. □

Lemma 6.5 $N(d+1,h) \leq 2N(d,h)$

Proof. Consider a point set $S \subseteq V^{d+1}$, $|S| = n = N(d+1,h)$ with $min_H(S) = h$. Split $\mathcal{C}^{d+1}$ into two halves by fixing the first bit of every vertex to either 0 or 1. Then at least one half of the cube must contain at least $\lceil \frac{n}{2} \rceil$ vertices of S and thus $N(d,h) \geq \lceil \frac{n}{2} \rceil$. □

We now consider the special cases $h = \frac{d}{2}$ for d even, and $h = \frac{d+1}{2}$ for d odd.

Lemma 6.6 *Let d be even. Then $N(d, \frac{d}{2}) \leq 2d$.*

$v_1 = (0,0,0,0,0,0,0,0)$
$v_2 = (1,1,1,1,1,1,1,1)$
$v_3 = (0,0,0,0,1,1,1,1)$
$v_4 = (1,1,1,1,0,0,0,0)$
$v_5 = (0,0,1,1 \mid 0,0,1,1)$
$v_6 = (0,0,1,1 \mid 1,1,0,0)$
$v_7 = (1,1,0,0 \mid 0,0,1,1)$
$v_8 = (1,1,0,0 \mid 1,1,0,0)$
$v_9 = (0,1 \mid 0,1 \mid 0,1 \mid 0,1)$
$v_{10} = (0,1 \mid 0,1 \mid 1,0 \mid 1,0)$
$v_{11} = (0,1 \mid 1,0 \mid 0,1 \mid 1,0)$
$v_{12} = (0,1 \mid 1,0 \mid 1,0 \mid 0,1)$
$v_{13} = (1,0 \mid 0,1 \mid 0,1 \mid 1,0)$
$v_{14} = (1,0 \mid 0,1 \mid 1,0 \mid 0,1)$
$v_{15} = (1,0 \mid 1,0 \mid 0,1 \mid 0,1)$
$v_{16} = (1,0 \mid 1,0 \mid 1,0 \mid 1,0)$

Table 6.3: 16 vertices of $\mathcal{C}^8$ with pairwise Hamming distance at least 4.

Proof. Let $d' = d-1$ and $h' = \frac{d}{2} = \frac{d'+1}{2}$. Then by Theorem 6.3 we have $N(d',h') \le 2\lfloor \frac{h'}{2h'-d'} \rfloor = d'+1 = d$. And from Lemma 6.5 we conclude $N(d,\frac{d}{2}) \le 2N(d-1,\frac{d}{2}) = 2N(d',h') \le 2d$. □

Theorem 6.4 *Let $d = 2^k$, $k \ge 1$. Then $N(d,\frac{d}{2}) = 2d$.*

Proof. Let $d = 2^k$, $k \ge 1$. The upper bound is given by the last lemma. We prove the lower bound by constructing a vertex set S with $min_H(S) = \frac{d}{2} = 2^{k-1}$ and $|S| = 2d = 2^{k+1}$.

Start with $v_1, v_2 \in S$, $v_1 = (0,\ldots,0)$ and $v_2 = (1,\ldots,1)$. Since $min_H(S) \ge \frac{d}{2}$ all additional vertices of S must have precisely $\frac{d}{2}$ '1'-bits. So let $v_3 = (0,\ldots,0_{d/2},1,\ldots,1) \in S$ and $v_4 = (1,\ldots,1_{d/2},0,\ldots,0) \in S$. This 'splits' the bit positions into two halves and it is easy to see that any additional vertex must have $\frac{d}{4}$ '1'-bits for each half.

We now iterate this process for $i = 2\ldots k$ (the case $i=1$ just generated v_3 and v_4). Before step i the bit positions were split into 2^{i-1} parts and every additional vertex of S must have exactly $\frac{d}{2^i}$'1'-bits for each part. We again only consider vertices where for all parts of the previous subdivision all '1'-bits are either shifted to the right or to the left. For iteration $i=2$ we get for example $v_5 = ((0,\ldots,0_{d/4},1,\ldots,1_{d/2},0,\ldots,0_{3d/4},1,\ldots,1) \in S$ and v_6 to v_8 are defined analogously. See Table 6.3 for an example for dimension 8.

Since $min_H(S) \ge \frac{d}{2}$ any pair of vertices added during iteration i must differ in at least half of these parts, i.e. 2^{i-2} out of 2^{i-1} parts have to be different. Thus there are at least $N(2^{i-1},2^{i-2})$ such vertices. (Note that by construction all vertices added during step i have a Hamming distance of exact $\frac{d}{2}$ to all previously added vertices of S.)

We thus get the relation $N(2^k,2^{k-1}) \ge 2 + 2 + \sum_{i=2}^{k} N(2^{i-1},2^{i-2}) = N(2^{k-1},2^{k-2}) + 4 + \sum_{i=2}^{k-1} N(2^{i-1},2^{i-2}) \ge 2N(2^{k-1},2^{k-2})$. With $N(2,1) = 4$ it follows by induction that $N(2^k,2^{k-1}) \ge 2^{k+1} = 2d$. □

Lemma 6.7 *Let $d = 2^k - 1$, $k \geq 1$. Then $N(d, \frac{d+1}{2}) \geq d + 1$.*

Proof. With $d' = d + 1 = 2^k$ we get $N(d', \frac{d'}{2}) \leq 2N(d' - 1, \frac{d'}{2}) = 2N(d, \frac{d+1}{2})$ from Lemma 6.5. By Theorem 6.4 we have $N(d', \frac{d'}{2}) = 2d'$ and therefore $N(d, \frac{d+1}{2}) \geq d' = d + 1$. □

The last lemma shows that for dimensions $d = 2^k - 1$, $k \geq 1$ we can find $d+1$ vertices of $\mathcal{C}^d$ with pairwise Hamming distance $\frac{d+1}{2}$. More precisely only the lower bound for the Hamming distance follows from the last lemma. But the construction used to prove Theorem 6.4 together with the fact that we take, for example, only vertices with a leading '0'-bit to obtain a set of dimension $2^k - 1$, shows the regularity of the set, i.e. that in fact all pairwise Hamming distances are $\frac{d+1}{2}$. Thus these $d+1$ vertices build the 'corners' of a regular d-simplex inscribed into the hypercube.

There exists a well known relationship between the existence of a regular d-simplex with vertices from $\mathcal{C}^d$ and the existence of Hadamard matrices of dimension $d + 1$, see e.g. [71]. A *Hadamard matrix* M is an $n \times n$ matrix with entries from $\{-1, 1\}$ which satisfies $MM^T = M^T M = nI_n$, see e.g. Chapter 18 in [68]. In [50] it is shown that there always exists a Hadamard matrix when $n = 2^k$, and its existence is conjectured for every $n = 4k$. (This conjecture has been verified for $n < 428$; see [33, 50]).

In fact the construction used in the proof of Theorem 6.4 is very similar to the construction of the related Hadamard matrix. See also Chapter 2 for other relationships between Hadamard matrices and properties of the hypercube.

So far the results we obtained for $N(d, h)$ are for $h \geq \frac{d}{2}$. So let us now consider the case of small (nearly constant) values of h. To this end we denote with $\tilde{N}(d, h)$ the maximum number of vertices $x \in \mathcal{C}^d$ with pairwise Hamming distance of at least h and between 1 and $h - 1$ '1' bits (i.e. vertices in or between levels 1 and $h-1$). In other words $\tilde{N}(d, h)$ gives the maximum number of vertices within distance at most $h - 1$ around a given vertex v, which are at least h bit apart from each other. Because of the symmetry of the hypercube we can w.l.o.g. set the fixed vertex v to the origin.

Theorem 6.5

$$\left\lceil \frac{2^d}{\sum_{i=0}^{h-1} \binom{d}{i}} \right\rceil \leq N(d, h) \leq \left\lfloor \frac{2^d}{1 + \frac{\sum_{i=1}^{h-1} \binom{d}{i}}{\tilde{N}(d,h)}} \right\rfloor$$

Proof. Let $S \subseteq V^d$ realize $N(d, h)$. Since the minimum Hamming distance between vertices of S is h, every vertex of $x \in S$ blocks all vertices of $\mathcal{C}^d$ within distance at most $h - 1$ around x, i.e. exactly $\sum_{i=0}^{h-1} \binom{d}{i}$ vertices (including x itself). As long as there are vertices of $\mathcal{C}^d$ which are not blocked we might add them to S and thus the lower bound follows immediately.

For the upper bound observe that a vertex not included in S can be blocked several times. The number of 'blocking' vertices from S for a given vertex $v \in \mathcal{C}^d$ is bounded from above by $\tilde{N}(d, h)$. Thus the number of vertices not in S but blocked by the vertices of the set S is at least $b = \frac{\sum_{i=1}^{h-1} \binom{d}{i}}{\tilde{N}(d,h)} |S|$. Clearly $b + |S| \leq 2^d$ and the upper bound follows. □

Lemma 6.8 *$\tilde{N}(d, 2) = d$, $\tilde{N}(d, 3) = \lceil \frac{d}{2} \rceil$, $\tilde{N}(d, 4) \leq \lfloor \frac{d(d-1)}{6} \rfloor$.*

Proof. There are exactly d vertices in $\mathcal{C}^d$ at Hamming distance one from the origin, all having a pairwise Hamming distance of 2. Thus $\tilde{N}(d,2) = d$.

For $\tilde{N}(d,3)$ observe that two vertices of level 2 can achieve a Hamming distance of at least 3 only if they have their '1'-bits for different directions, i.e. for different positions in their binary presentation. The same holds if one vertex has level 1 and the other level 2. Moreover we can at most include one vertex of level 1 in our set. Thus for even dimension d we can either take a set of $\frac{d}{2}$ vertices from level 2 or a set of $\frac{d}{2} - 1$ vertices from level 2 and a single vertex of level 1. Similarly for odd d we get a set of $\frac{d-1}{2}$ vertices from level 2 and a single vertex of level 1, and thus $\tilde{N}(d,3) = \lceil \frac{d}{2} \rceil$.

For $\tilde{N}(d,4)$ we have to consider different cases. For the first case let us assume that we only take vertices from level 3. Every such vertex v blocks $3(d-3)$ other vertices of level 3, namely all vertices where exactly two positions of '1'-bits coincide with the positions of '1'-bits in v. On the other hand for a given vertex x of level 3 there are at most 3 blocking vertices which realize a pairwise Hamming distance of at least 4. This comes from the fact that they have to share two positions of '1'-bits with x but at most one such position with each other. Thus if $\tilde{n}(d)$ is the size of our set we block at least $\frac{3(d-3)}{3}\tilde{n}(d) = (d-3)\tilde{n}(d)$ other vertices of level 3. Clearly we have $\tilde{n}(d) + (d-3)\tilde{n}(d) \leq \binom{d}{3}$ and thus we get $\tilde{n}(d) \leq \frac{d(d-1)}{6}$.

Let us now consider the case where we also choose vertices of levels 1 and 2. If we take a vertex from level 1 then all other vertices have to come from level 3 to achieve the minimum Hamming distance of 4. Moreover none of these vertices from level 3 can have a '1'-bit at the same position as the vertex from level 1. Thus the maximum number of vertices we may obtain is $1 + \tilde{n}(d-1) = 1 + \frac{(d-1)(d-2)}{6}$ which, for $d \geq 4$, is not greater than $\frac{d(d-1)}{6}$.

It remains to consider the case where we take vertices from level 2 and 3. Observe that a level 2 vertex can not share a '1'-bit positions with another vertex of level 2 or 3. Thus, taking k level 2 vertices, $0 \leq k \leq \frac{d}{2}$, $d \geq 4$, the cardinality of our set is bounded by $k + \tilde{n}(d-2k)$, which is maximized for $k = 0$. This completes the proof. □

Although we could only prove the upper bound for $\tilde{N}(d,4)$ this bound seems to be tight for at least some values of d. For an example of 12 vertices from $\mathcal{C}^9$ with at most 3 '1'-bits and pairwise Hamming distances of at least 4 see the next section (Table 6.6).

Theorem 6.6

$$N(d,1) = 2^d$$

$$N(d,2) = 2^{d-1}$$

$$\left\lceil \frac{1}{d^2+d+2} 2^{d+1} \right\rceil \leq N(d,3) \leq \begin{cases} \lfloor \frac{1}{d+2} 2^d \rfloor & \textit{for } d \textit{ even} \\ 2\lfloor \frac{1}{d+1} 2^{d-1} \rfloor & \textit{for } d \textit{ odd} \end{cases}$$

$$\left\lceil \frac{3}{d^3+5d+6} 2^{d+1} \right\rceil \leq N(d,4) \leq \begin{cases} 2\lfloor \frac{1}{d} 2^{d-2} \rfloor & \textit{for } d \textit{ even} \\ \lfloor \frac{1}{d+1} 2^{d-1} \rfloor & \textit{for } d \textit{ odd} \end{cases}$$

Proof. The value of $N(d,1)$ is obvious. The upper bound for $N(d,2)$ follows from Theorem 6.5 and Lemma 6.8. A set of all vertices with an odd number of '1's shows $N(d,2) \geq 2^{d-1}$.

The lower bounds for $N(d,3)$ and $N(d,4)$ are provided by Theorem 6.5, as well as, in conjunction with Lemma 6.8, the upper bound of $N(d,3)$ for even d. The bound for odd d is

$h \setminus d$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768	65536
2	–	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768
3		–	2	2	4	8	16	20..25	32..50	64..85	128..170	256..292	512..584	1024	2048	2048..3640
4			–	2	2	4	8	16	20..25	32..50	64..85	128..170	256..292	512..584	1024	2048
5				–	2	2	2	4	6	12	24	25..48	32..96	64..192	128..384	256..768
6					–	2	2	2	4	6	12	24	25..48	32..96	64..192	128..384
7						–	2	2	2	2	4	4	8	16	32	32..64
8							–	2	2	2	2	4	4	8	16	32
9								–	2	2	2	2	2	4	4	6
10									–	2	2	2	2	2	4	4
11										–	2	2	2	2	2	2
12											–	2	2	2	2	2
13												–	2	2	2	2
14													–	2	2	2
15														–	2	2
16															–	2

Table 6.4: Maximal number $N(d, h)$ of vertices of $\mathcal{C}^d$ with pairwise Hamming distance at least h.

now obtained by Lemma 6.5: $N(d, 3) \leq 2N(d-1, 3)$. Note that this gives a slightly stronger bound than we would get directly from Theorem 6.5.

Similarly, applying Lemma 6.8 to Theorem 6.5 for $N(d, 4)$ would give an upper bound of $\lfloor \frac{d-1}{d^2+d+4} 2^d \rfloor$. But with Lemma 6.4 we get for $d \geq 4$ $N(d, 4) \leq N(d-1, 3)$ and thus the stronger bounds stated in the theorem. $\square$

Generalizing the above argumentation used for $N(d, 4)$ we get

Corollary 6.2 *For $d \geq h \geq 3$*

$$N(d, h) \leq \begin{cases} 2\lfloor \frac{1}{d-h+4} 2^{d-h+2} \rfloor & \text{for } d-h \text{ even} \\ \lfloor \frac{1}{d-h+5} 2^{d-h+3} \rfloor & \text{for } d-h \text{ odd} \end{cases}$$

Table 6.4 summarizes the results on $N(d, h)$ from the previous theoretical results and a huge number of exhaustive computer searches. The most challenging open problem for small dimensions is to determine $N(8, 3)$. We conjecture that here the given lower bound 20 is tight. This would, by the previous theoretical results, also improve some upper bounds of Table 6.4.

Finally let us give two applicational examples of the above results to coding theory. Detailed background on error correcting codes can e.g. be found in [103] or Chapter 20 of [68]. For the first example assume that we want to transmit a message consisting of 10 different characters, e.g. a file of decimal numbers. Moreover we want use a fixed number of bits for every character and we also require to be able to correct 2-bit errors. Thus the Hamming distance between two code words must be at least 5. From Table 6.4 we see that we must use code words of at least length 10.

For the second example we use the extended ASCII set (256 characters) and want to be able to correct 1-bit errors and to detect 2-bit errors (but not to correct them). Thus we need a Hamming distance of 4 between code words and conclude from Table 6.4 that we need at least a word length of 13 bit.

d	1	2	3	4	5	6	7	8	9	10
$\mathcal{N}(d)$	1	2	3	6	10	15	21	28	≥ 36	≥ 45
$\mathcal{N}_u(d)$	1	2	3	4	5	6	7	8	12	13

Table 6.5: Maximal numbers $\mathcal{N}(d)$ and $\mathcal{N}_u(d)$ having the origin as (unique) nearest neighbour.

6.4 Counting Nearest Neighbours in the Hypercube

Algorithms on the hypercube are often based on the concept of nearest neighbour vertices. Examples are the construction of minimum spanning trees or various hierarchical clustering strategies, see Chapter 7 for details and further examples. Note that we again take the Hamming distance as the distance function.

Two different questions appear in this context. The first asks how many nearest neighbours a fixed vertex $v \in \mathcal{C}^d$ has in a given subset $v \notin S \subset V^d$ of hypercube vertices. W.l.o.g. we may assume that v is the origin and thus all nearest neighbours of v have to lie in the same level (the lowest 'occupied' level). Thus it is well known that there are up to $\binom{d}{\lfloor d/2 \rfloor} = \Theta(2^d/\sqrt{d})$ possible nearest neighbours, namely all vertices of level $\lfloor d/2 \rfloor$, see Chapter 2.

The reverse question seems not to be as easy to solve. It can be stated as follows:

MNN **MAXIMUM NEAREST NEIGHBOUR**
QUESTION: For how many vertices of $\mathcal{C}^d$ can a vertex $v \in \mathcal{C}^d$ be simultaneously the (unique) nearest neighbour?

This question arises for example during the construction of hierarchical cluster trees. How often a fixed vertex appears as the nearest neighbour of other vertices affects the number of necessary updates in a union-find structure and thus it also affects the running time (cf. Chapter 7).

We consider two variants of the problem. For the first version we take for a vertex $x \in S$ the vertex v as the nearest neighbour, even if there are other vertices in our subset at the same distance from x. We denote the maximal cardinality of such a set by $\mathcal{N}(d) = \max_{v \notin S \subseteq V^d} \{|S| : \forall x, y \in S, x \neq y : d_H(x,v) \leq d_H(x,y)\}$.

In the second variant v has to be the unique nearest neighbour of every vertex in the set, that is $\mathcal{N}_u(d) = \max_{v \notin S \subseteq V^d} \{|S| : \forall x, y \in S, x \neq y : d_H(x,v) < d_H(x,y)\}$.

Clearly $\mathcal{N}(d)$ is an upper bound for $\mathcal{N}_u(d)$. W.l.o.g. let the vertex $v \in \mathcal{C}^d$ be the origin. Lower bounds can now be obtained by simple observations. All vertices of level 2 are at least at Hamming distance 2 from each other and at Hamming distance 2 from the origin. And all vertices at level 1 are at Hamming distance 1 from the origin and at Hamming distance 2 from each other. Thus we get

Observation 6.2 *$\mathcal{N}(d) \geq \frac{d(d-1)}{2}$ for $d \geq 3$, and $\mathcal{N}_u(d) \geq d$ for $d \geq 1$.*

Although this lower bounds are rather trivial they are tight for low dimensional hypercubes. For dimensions $d = 1, \ldots, 10$ Table 6.5 gives the numerical results for $\mathcal{N}(d)$ and $\mathcal{N}_u(d)$, respectively. While for $d = 1, \ldots, 8$ we computed all possible vertex sets we got only lower

$$
\begin{aligned}
v_1 &= (000000111) \text{ a} \\
v_2 &= (001110000) \text{ a} \\
v_3 &= (110001000) \text{ a} \\
v_4 &= (000011001) \text{ b} \\
v_5 &= (010100100) \text{ b} \\
v_6 &= (101000010) \text{ b} \\
v_7 &= (000101010) \text{ c} \\
v_8 &= (011000001) \text{ c} \\
v_9 &= (100010100) \text{ c} \\
v_{10} &= (001001100) \text{ d} \\
v_{11} &= (010010010) \text{ d} \\
v_{12} &= (100100001) \text{ d}
\end{aligned}
$$

Table 6.6: 12 vertices of $\mathcal{C}^9$ with the origin as their unique nearest neighbour.

bounds for $\mathcal{N}(9)$ and $\mathcal{N}(10)$. It can be seen that the lower bound for $\mathcal{N}(d)$ seems to be tight, at least for small dimensions.

The numerical examples of Table 6.5 also show that the lower bound for $\mathcal{N}_u(d)$ is tight for $d = 1, \ldots, 8$. However, the example of Table 6.6 shows that $\mathcal{N}_u(9) = 12$. The twelve vertices are all from level 3. They build four triples (a to d) with pairwise Hamming distance 6. All remaining Hamming distances between vertices of different triples are 4. Thus the origin is the unique nearest neighbour to all twelve vertices.

Observe also the relationship to the value $\tilde{N}(d, h)$ from the last section. The example of Table 6.6 shows that $\tilde{N}(9, 4) \geq 12$ and therefore the upper bound of Theorem 6.5 is tight in this case.

Although the example for $\mathcal{N}_u(9)$ is highly symmetric it seems to be a hard task to give the exact values of $\mathcal{N}_u(d)$ for general dimension d. An indication for this is also the maximal number of vertices for the next higher dimension, namely $\mathcal{N}_u(10) = 13$.

Based on the example of Table 6.6 we derive an improved lower bound for $\mathcal{N}_u(d)$ in the next theorem.

Theorem 6.7 $\mathcal{N}_u(d) \geq 12 \cdot 4^{\lfloor \log_2 d/9 \rfloor} + d - 9 \cdot 2^{\lfloor \log_2 d/9 \rfloor}$, $d \geq 9$

Proof. We prove the lower bound by constructing sets including only vertices from level 3. Starting with the example of Table 6.6 we show by induction that we can generate similar vertex sets for higher dimensions.

Thus assume we have a vertex set V of dimension d and of size n. All vertices of V are from level 3 with pairwise Hamming distance of at least 4. We will show how to construct a vertex set V' of dimension $2d$ and of size $4n$, such that all vertices of V' are from level 3 and with pairwise Hamming distance of at least 4, too.

To this end for every vertex of V double every bit, such that a '0'-bit becomes '00' and a '1'-bit becomes either '01' or '10' in the following way. For every vertex from V we choose one of the four different assignments for its three '1'-bits: $(1, 1, 1) \mapsto (01, 01, 01)$, $(01, 10, 10)$, $(10, 01, 10)$ or $(10, 10, 01)$. Thus for a vertex from V we can construct four vertices of V' with pairwise Hamming distance of 4. Vertices of V' which are generated from different vertices of V have by induction a pairwise Hamming distance of at least 4. Thus V' indeed shows that $\mathcal{N}_u(2d) \geq 4n$.

Together with the example of Table 6.6 we conclude that for dimensions $d = 9 \cdot 2^k$, $k \geq 0$, we can construct vertex sets of size $12 \cdot 4^k$.

For dimensions not of the form $d = 9 \cdot 2^k$ let $l = \lfloor \log_2 d/9 \rfloor$ and we get a set of $12 \cdot 4^l$ vertices of dimension $9 \cdot 2^l$ as shown above. Moreover observe that $\mathcal{N}_u(d+1) \geq \mathcal{N}_u(d) + 1$, since we only need to add a vertex with a '1' for the new dimension and '0' else. This vertex obviously has the origin as its nearest neighbour and is farther apart from all other vertices than the origin is. Therefore we can in addition include $d - 9 \cdot 2^l$ vertices from level 1 to our set, and the theorem follows. $\square$

From the last Theorem we get an asymptotic lower bound of $\Omega(d^2)$ for $\mathcal{N}_u(d)$. Thus we conjecture that both numbers, $\mathcal{N}(d)$ and $\mathcal{N}_u(d)$, respectively, increase quadratically in the dimension.

Chapter 7

Clustering the Hypercube

7.1 Introduction and Notation

In this chapter we consider various clustering methods for objects represented as binary strings of fixed length d. The dissimilarity of two given objects is their Hamming distance. Clustering these objects can be seen as clustering a subset of the vertices of a d-dimensional hypercube, and thus is a geometric problem in d dimensions. We give algorithms for various agglomerative hierarchical methods (including single linkage and complete linkage) as well as for two-clusterings and divisive methods.

Recall that the set $V^d = \{0,1\}^d$ is the set of vertices of the d-cube, and can be seen as the set of all binary strings of length d. The vertices of the d-cube, (elements of V^d) represent the objects to be clustered, and we will use the terms 'object' and 'vertex of $\mathcal{C}^d$' synonymously. The goal of this chapter is to give various methods to cluster a subset $P \subseteq V^d$ of objects.

Also recall that the *Hamming distance* $d_H(x, y)$ of two vertices $x, y \in P$ is the number of coordinates where x differs from y. In the language of clustering, the distance function d_H is called symmetric and invariant. Invariance means that d_H is not affected by permuting coordinates (each bit in a string is equally important), and symmetry means that the probability of being '1' is 1/2 for all bits in a string randomly chosen from P. See e.g. Chapter 4 in [21] for a more detailed classification of binary distance functions.

We will use the term 'distance' to describe the dissimilarity of two objects, whereas the term dissimilarity itself is taken to measure the (non-)quality of a cluster. Thus a cluster containing very close objects has high quality and thus a low dissimilarity. For a single object its dissimilarity is zero, for two objects it coincides with their distance. The dissimilarity of a cluster may be its diameter (e.g. for C-link) or the longest edge in its minimum spanning tree (S-link) or the mean distance between points (a version of A-link) and so on.

By symmetry of $\mathcal{C}^d$, for each vertex $x = (x_1, x_2, \dots, x_d) \in V^d$ there is a vertex $\overline{x} = (1 - x_1, 1 - x_2, \dots, 1 - x_d) \in V^d$ which we call the *antipode* of x in $\mathcal{C}^d$. Note that x and $\overline{x}$ represent two of the most disagreeing objects of a set. We will use antipodal pairs when constructing worst case configurations.

To make the description of the algorithms simpler we assume that the objects of P are pairwise different, i.e. their Hamming distance is greater or equal one. If multiple copies of an object are required in an application, we just weight an object by the number of its appearances. Where necessary the computation of the cluster quality has to be modified accordingly.

All algorithms we present in this chapter use only linear space. In particular, we do not store any kind of a distance matrix. Thus, unlike most other clustering methods, we do *not* assume that the distance of two objects can be computed in constant time. We rather have to spend $O(d)$ time for any such computation, and we will take this into account in the time analysis. The time requirements thus will be given in terms of the number of objects *and* the dimension.

For $P \subset V^d$ we denote with $n = |P|$ the number of objects to be clustered (i.e., the problem size), which may range from a few thousand to tens or hundreds of thousands. Typical values for the number d of bits (the problem dimension) are in the range from 32 (a 4 byte long integer) up to several hundreds (e.g. 256 or 512). We have $n > d$ and since a d-cube has 2^d vertices we get $d > \log_2 n$. Moreover we assume $\log_2 d \leq 31$ and $\log_2 n \leq 31$ so that we can handle ordinary variables and indices of objects in constant time (e.g. as long integers on current computers). Clearly this is not a serious limitation to applications since it allows about 10^9 objects to be represented.

For typical applications the space needed to store the input can range from some kilobytes up to several megabytes. A large example may have $n = 100000$ objects with $d = 200$ bits each. Thus we need 25 byte to store an object and about 2.5 MB to store the whole set. Therefore to be practicable the overhead in space required by the clustering algorithms is to be at most linear in the input size, i.e. $O(dn)$ with some small constant. All algorithms given in this chapter fulfill this requirement. Thus we suppose that all data can be kept in the main memory, and that an object and its augmenting information can be accessed in constant time.

A previous version of this chapter appeared as [3].

7.2 Agglomerative Hierarchical Clustering

Agglomerative hierarchical clustering methods build a full cluster tree in the "bottom up" way, which means that leaves (and fruits?) grow first, then branches and the trunk come up, and finally the root appears. (Well, somehow that's top down, but we are all used to see trees up side down, so it's bottom up again.)

More formally an agglomerative hierarchical clustering method starts with n single objects, all seen as their own clusters. Step by step two specific clusters (objects at the beginning) are merged to a new cluster until, after $n-1$ steps, all objects form in a single final cluster, the root of the tree. Thus a hierarchical cluster tree is a family of clusters such that any two clusters are disjoint iff they lie on different paths (seen from the root), or one includes the other iff they are on the same path.

These cluster trees are sometimes visualized in the form of *dendrograms*: On the y-axis single objects and the agglomeration steps are shown, whereas the x-axis gives the dissimilarity of the resulting clusters, increasing from left to right. For an example of a dendrogram see Figure 7.9.

The two clusters to be merged in the next step are chosen in a way that the dissimilarity of the resulting cluster is minimized among all possible pairs. The criterion how to compute the dissimilarity of a (resulting) cluster determines the different methods. This criterion should fulfill the property that merging two clusters results in a new cluster with a dissimilarity at least the maximum of the two dissimilarities of the merged clusters. Thus the maximum dissimilarity of all clusters in a certain subtree is realized by the last cluster built (the subtree root). It is non-decreasing and can be seen as the measure of this subtree. Criteria fulfilling this property and cluster trees resulting from them are called *monotonous*. The closer a cluster lies to the root the more objects it will contain and the larger its dissimilarity will be. Therefore a monotonous tree gives us n levels of increasing maximum dissimilarity and decreasing number of clusters (seen from leaves to root).

The advantage of hierarchical clustering is now this trade-off between the number of clusters of a certain partition and their maximum dissimilarity. For applications where it is a priori not known how many, i.e. few, clusters are required, the best suitable combination of number and dissimilarity can easily be chosen once the cluster tree has been computed. Though needing only linear storage for the whole cluster tree, hierarchical cluster methods are more flexible than other clustering methods.

For a more detailed description of hierarchical clustering see e.g. Chapter 9 in [21] or Chapter 8 in [52]. Various methods for the dissimilarity computation of combinatorial agglomerative hierarchical clustering methods are given in Table 2 in [34]. Note however, that most of the dissimilarities there can only be computed in constant time if $O(n^2)$ space is available to store the distance matrix of the clusters. In the following sections we will use dissimilarity functions which turn out to be suitable for the d-cube case, needing only $O(dn)$ space.

An application of the hierarchical clustering of hypercubes given in this chapter, can be found in Chapter 8. There the structure of the cluster tree is utilized to give a new approach to the $\{0,1\}$-stringsearch problem: Given a set $P \subseteq V^d$ of $\{0,1\}$-strings of fixed length d, find for a query string $x \in V^d$ the "most similar" element $p \in P$. Most similar here means to minimize the Hamming distance between x and p. The cluster hierarchy now enables them to find the optimum vertex p without considering the whole set P by skipping irrelevant subclusters.

The widely used *single linkage* method (S-link) takes the two clusters which realize the smallest inter-cluster distance (i.e., the min-pair of elements in different clusters). The advantage of this method is its fast computability, but the quality of the resulting clusters may be poor due to chaining-effects; see e.g. [51]. To overcome this deficiency, *complete linkage* (C-link) uses a different criterion to chose the next two clusters: it minimizes the diameter (maximum distance of objects in one cluster) of the resulting cluster. Thus the resulting clusters are more "sphere-like" and produce better results for practical problems like document retrieval [51]. The clear disadvantage of C-link is the more difficult and thus time consuming computation.

Another variant is *average linkage* (A-link) where the mean (or sum) of the distances between all pairs of objects of a cluster is taken as the cluster dissimilarity. There are several variants of this methods, where e.g. a "central" representative of the cluster is chosen (either the center of gravity or a "typical" object of the set) and only the distances to this object are taken into account.

In the following subsections we will give algorithms for all mentioned methods.

7.2.1 Single Linkage Clustering

The single linkage clustering (S-link, also called *nearest-neighbour clustering*) is one of the oldest methods of cluster analysis; it was first suggested by Florek et al. [40] and later on by various other authors. The first efficient general implementation of S-link was given by Sibson [98].

When building up the cluster tree S-link takes the two clusters which realize the smallest inter-cluster distance (min-pair with elements of different clusters). Intuitively speeking this approach tries to merge clusters having objects very close to each other. Thus the dissimilarity of the resulting clusters is defined as the longest edge in their minimum spanning tree (MST, cf. the next subsection for a definition).

The advantage of this method is its fast computability, because of its close relation to the well known MST of the objects. Thus, especially in low dimensions, it can be computed very fast, e.g. in $O(n \log n)$ time for general point sets for $d = 2$, with the help of the Delaunay triangulation (see e.g. [12]).

By definition of S-link, sorting the edges of a MST by length in non-decreasing order reflects the sequence of merging steps of S-link. Thus, once given the MST of n points, we can compute the S-link tree in $O(n \log n)$ time by sorting and inserting the $n - 1$ edges of the MST. For the d-cube case we have to take care that comparing two elements for sorting can be done in constant time. To this end the MST algorithm has to store the lengths of the edges of the MST. Of course this does not affect the time analysis of the MST algorithms and needs only $O(n)$ additional space.

The MST algorithm for $\mathcal{C}^d$ we present in the following subsection runs in $O(dn^2)$ time. Together with the above remarks we have the following:

Theorem 7.1 *For a set $P \subseteq V^d$ with n elements, the S-link tree of P can be computed in $O(dn^2)$ time and $O(dn)$ space.*

The Minimum Spanning Tree in $\mathcal{C}^d$

Let $P \subseteq V^d$ be a set of vertices of $\mathcal{C}^d$. Then its *minimum spanning tree*, $MST(P)$, is a tree of P which minimizes the sum of its edge lengths. The length of an edge is defined as the distance of its two endpoints.

In the plane the Euclidean MST of a point set can be computed in time $O(n \log n)$ by using the fact that it is a subgraph of the Delaunay triangulation. Although in higher dimensions the MST is still a subset of the edges of the Delaunay triangulation this does not lead to an efficient algorithm since there the time to compute the Delaunay triangulation is $\Omega(n^{\lceil d/2 \rceil})$ [97]. Vertices of $\mathcal{C}^d$ are highly degenerate in the sense that there are many linear dependences between them. This would cause a lot of numerical troubles in computing the Delaunay triangulation for $\mathcal{C}^d$, and in fact most of the existing software packages fail on this task (compare e.g. [26]). In addition, vertices of P may also be cocircular (consider the hyper sphere with the same center as the cube and going through all vertices of $\mathcal{C}^d$) and thus the Delaunay triangulation may not even be unique.

Another way to compute the MST is the greedy approach of the graph algorithms of Prim and Kruskal. They insert the edges into an initially empty graph in a certain length order, adding only edges which do not close a circle. While the algorithm of Kruskal first sorts the

1. Let $MST(p)$ contain one arbitrary vertex $p \in P$. Let Q be an empty priority queue.
2. For all remaining vertices of P compute the distance to p, i.e. their priority, and insert them into Q.
3. **WHILE** Q is not empty **DO**
   (a) Remove the first element q of Q and add it to MST.
   (b) Restructure Q.
   (c) **FOR** all remaining elements b in Q **DO**
      i. Compute the distance from q to b.
      IF this distance is smaller than the priority of b **THEN**
      ii. update the priority and nearest neighbour pointer of b
      iii. update the position of b in Q

Figure 7.1: Prims MST algorithm for $\mathcal{C}^d$, running in $O(dn^2)$ time and $O(dn)$ space.

edges in non-decreasing length to get a "global" order, the method of Prim always "locally" takes the shortest edge between two (not yet linked) components. Its not hard to see that both approaches indeed produce a MST of the given point set. But since Kruskal's approach needs to sort the edges it would require $O(n^2)$ space for a complete graph and is thus impractical in the d-cube case.

For both methods the time to compute the MST is $O(m \log n)$ where m is the number of edges in the given graph, i.e. $m = O(n^2)$ in our case. Moreover they assume that the distance between two points can be computed in $O(1)$ time. In our case, we have to spend $O(d)$ time because, due to space limitations, we cannot simply store the distance matrix. On a first glance this gives a time of $O(dn^2 \log n)$.

A more careful look on Prim's algorithm shows that it can be made to run in $O(dn^2)$ time and $O(dn)$ space. The algorithm is given in Figure 7.1. For the correctness of the algorithm see any basic text book an algorithms, e.g. [30].

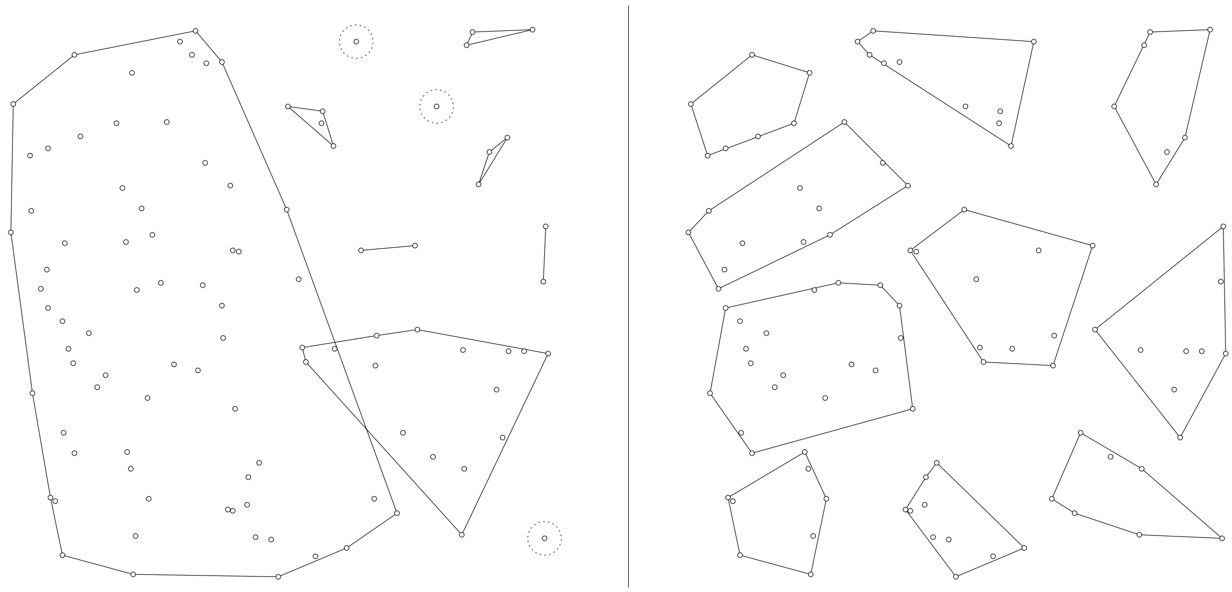
The basic data structure we use is a simple priority queue Q containing all vertices not yet in the partially built MST. For any vertex in Q we store the minimum distance (=priority) to a vertex in the MST and a pointer to this vertex. Whenever a point is moved from Q to the MST this information is stored in the MST. The additional space needed for the algorithm is $O(n)$. As mentioned above S-link can then use this information to efficiently complete the cluster tree.

For the time analysis recall that we have $d > \log_2 n$. The initialization of steps 1 and 2 can be done in $O(dn)$ time. The while loop of step 3 is repeated $n-1$ times where step 3a and 3b can be done in constant and $O(\log n)$ time, respectively. Again the for loop of step 3c repeats $O(n)$ times and requires $O(d + 1 + \log n) = O(d)$ time per iteration. This gives a running time for the algorithm of $O(dn + n \log n + dn^2)$ which simplifies to $O(dn^2)$.

Lemma 7.1 *Given a set $P \subseteq V^d$ with n elements the MST of P can be computed in $O(dn^2)$ time and $O(dn)$ space.*

Note that since we have only d different distances we could also use, instead of the priority queue Q, d linked lists to organize the priorities. Thus the updates in step 3b and 3ciii could be done in constant time instead of $O(\log n)$. But since $d > \log_2 n$ this would not affect the asymptotic running time of the algorithm. It might simplify the implementation, however.

7.2.2 Generic Agglomerative Hierarchical Clustering Algorithms



(a) (b)

Figure 7.2: Planar point set of size $n = 100$ clustered by (a) S-link and (b) C-link. Both cluster hierarchies are shown at the level of 10 clusters. For S-link single-point clusters are marked by a dotted circle.

Unfortunately the quality of the resulting clusters of S-link is often poor due to the chaining-effect of S-link [21]. It tends to build very uneven clusters, i.e. very long and tiny chains may contain nearly the whole set, while there are still non-clustered single objects. For an example of a planar point set see Figure 7.2(a).

We therefore give two generic hierarchical algorithms that run in $O(d^2n^2)$ and $O(dn^3)$ time, respectively, and require $O(dn)$ space. With the help of these algorithms we then can simply implement various criteria to choose the next pairs of clusters to be merged. We give some examples of criteria in the next subsections.

For a cluster C let $d(C)$ be its dissimilarity function or *δ-function* for short. Moreover let C be the agglomeration of the two subclusters C_i and C_j. To fit into the general framework, the dissimilarity function has to fulfill the following properties to which we will refer as *δ-function properties* in the sequel:

1. Put every object of P into a single-element cluster C_i, $1 \leq i \leq n$
2. **FOR** $diss = 1$ **TO** d **DO**
   (a) Set $i = 1$
   (b) **WHILE** i is less than the actual number of cluster **DO**
      i. Set $j = i + 1$
      ii. **WHILE** j is less than or equal to the actual number of clusters **DO**
         A. **IF** the dissimilarity of the union of C_i and C_j equals $diss$ **THEN**
            - Let C_i be the union of C_i and C_j
            - Replace C_j by the cluster with the highest index, changing its index to j
            - Reduce the number of clusters by one
            - (Set j = i+1) *; see text*
         B. **ELSE**
            - Set $j = j + 1$
   (c) Set $i = i + 1$

Figure 7.3: Generic hierarchical clustering Algorithm GHC1 for $\mathcal{C}^d$ with running time $O(d^2n^2)$.

1. For a single-object cluster C, $d(C) = 0$.
2. $d(C) \geq d(C')$ for any $C' \subseteq C$ in the cluster tree.
3. $d(C_i \cup C_j)$ can be computed with $O(|C_i| \cdot |C_j|)$ distance computations of single objects, where $|C|$ denotes the cardinality of C.
4. $d(C) \in \{0, 1, \ldots, d\}$.

Property 1) is not really necessary for the correctness of the algorithms but seems to be quite natural and is implicitly fulfilled by all reasonable δ-functions known to the author. Property 2) ensures that the dissimilarity of the clusters does not decrease with increasing level (i.e. towards the root) of the cluster tree. Note that, by transitivity, property 2) is equivalent to $d(C) = d(C_i \cup C_j) \geq \max\{d(C_i), d(C_j)\}$, i.e. the levels of the cluster tree are monotonous.

If property 2) is violated then the relation between the number of clusters of a level and their maximal dissimilarity is not longer monotonous. Thus reducing the number of clusters may give an even smaller maximum dissimilarity which is called an *inversion*. Of course this hardly makes sense in interpreting the given data and we thus only study δ-functions fulfilling property 2).

Properties 3) and 4) will be used to give good time bounds for the algorithms. Some of the presented δ-functions will violate property 4) and thus result in slower algorithms.

Related to property 3) we give a rather trivial but useful observation in the next lemma.

Lemma 7.2 *Given k integers $n_i \geq 1$, $i = 1, \ldots, k$ with $n_1 + \ldots + n_k = n$. Then*

$$\sum_{i=1}^{k} \sum_{j=i+1}^{k} n_i n_j \leq n^2$$

Proof. $\sum_{i=1}^{k} \sum_{j=i+1}^{k} n_i n_j = \sum_{i=1}^{k} \left(n_i \sum_{j=i+1}^{k} n_j \right) = \sum_{i=1}^{k} \left(n_i \left(n - \sum_{j=1}^{i} n_j \right) \right) =$
$= n \sum_{i=1}^{k} n_i - \sum_{i=1}^{k} \sum_{j=1}^{i} n_i n_j = n^2 - \sum_{i=1}^{k} n_i^2 - \sum_{i=1}^{k} \sum_{j=1}^{i-1} n_i n_j =$
$= n^2 - \sum_{i=1}^{k} n_i^2 - \sum_{j=1}^{k} \sum_{i=j+1}^{k} n_i n_j = n^2 - \sum_{i=1}^{k} n_i^2 - \sum_{i=1}^{k} \sum_{j=i+1}^{k} n_i n_j$
Rearranging the equality gives $\sum_{i=1}^{k} \sum_{j=i+1}^{k} n_i n_j = \frac{1}{2} \left(n^2 - \sum_{i=1}^{k} n_i^2 \right) \leq n^2$. □

We will use this lemma in the time analysis when the dissimilarity of every pair C_i, C_j of clusters has to be computed.

The scheme of the first generic hierarchical clustering algorithm *Algorithm GHC1* is given in Figure 7.3. In Step 1 every object is made to a single-object cluster. In the for loop of step 2 the dissimilarity to be considered is preset, starting from 1 and going up to d. In the body of this loop now all pairs of clusters are merged which produce a cluster with the given dissimilarity. Clearly from the definition of a δ-function the dissimilarity cannot decrease during the iterations. But we have to be careful if there exists a triple C_i, C_j, C_k of clusters, such that merging any two out of these three gives the same dissimilarity of the resulting cluster. In this case e.g. C_i may be involved in two merge steps within one iteration of the for loop. This is the reason why we give the handling of the indices explicitly.

The outer while loop of step 2b controls the first cluster C_i of the pair to be considered. Note that the actual number of clusters may decrease during the iteration. The inner while loop (2bii) controls the second cluster C_j such that any possible pair of clusters is considered exactly once. If two clusters are merged then the resulting cluster gets the index i. Cluster C_j is replaced by the cluster with the highest index and the number of clusters is reduced.

To handle the above mentioned special case, that one cluster may be involved in several agglomeration steps during one iteration of the FOR loop, we have to distinguish between two different methods to compute the dissimilarity of a cluster. For the first method we assume that adding points (i.e. another cluster) to a cluster C can never decrease the dissimilarity of C. This is e.g. the case for C-link, see Section 7.2.3. In this case no cluster with an index smaller than j can have a dissimilarity with C_i smaller or equal to *diss* after merging C_i and C_j. Thus it is sufficient to not increase the index j (which already points to a new cluster) after an agglomeration step.

In the second case adding points to a cluster C may reduce its dissimilarity. This is e.g. the case for centroid clustering if the centroid point is restricted to the set of cluster points, see Section 7.2.5. Note that this is no contradiction to δ-function property 2), since there the monotonicity is only required between clusters within the same path of the cluster tree. For example we may have three clusters C_i, C_j and C_k, $i < j < k$, where C_k contains a vertex which is the minimizing centroid point for C_i and C_j (cf. Section 7.2.5). Thus we first compute the dissimilarity of $C_i \cup C_j$ which may be too large. In the next step we merge C_i and C_k to C_i', since C_k provides a centroid point which gives a dissimilarity small enough for $C_i' = C_i \cup C_k$. But after this step the dissimilarity of $C_i' \cup C_j$ may be also small enough to be merged. Note that by δ-function property 2) it can not be smaller than the dissimilarity of C_i', but might be equal, which means that we might have to merge the two clusters within the same iteration of the FOR loop. Therefore it is necessary to reconsider all clusters C_j, $j > i$, after an agglomeration step, i.e. to set $j = i + 1$ as the last step in 2biiA.

In this way we guarantee that within one iteration of the FOR loop all possible pairs are merged and the correctness of the algorithm follows.

From the above discussion it also follows that in steps 2a up to 2c every possible pair of clusters is considered exactly once during an iteration of *diss* as long as there is no agglomeration step. If the objects of a cluster are stored in a linked list then by δ-function property 3) and Lemma 7.2 a single iteration of step 2 takes $O(dn^2)$ time, resulting in time $O(d^2n^2)$ for the whole Algorithm GHC1. If, as mentioned above, we have to reconsider all clusters C_j, $j > i$, after every agglomeration step, we add $n - 1$ times $O(dn)$ time, thus the overall time of the algorithm remains $O(d^2n^2)$.

Instead of using a FOR-loop in step 2 we can increase *diss* (after step 2c) to the smallest dissimilarity larger than *diss* which occurred during the actual iteration. Thus we may skip several steps, and experimental results show that this significantly speeds up the algorithm (cf. Section 7.4).

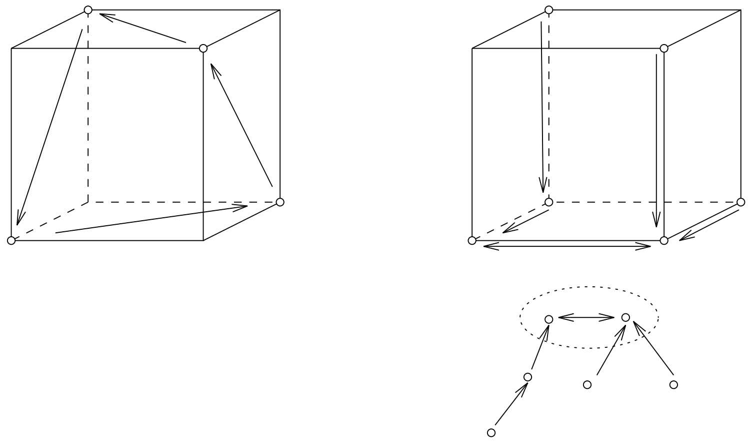
Essential for this first approach is that the dissimilarity of a cluster can only be an integer in the range $\{0, \ldots, d\}$ (δ-function property 4). For δ-functions which do not fulfill this property we give another approach, *Algorithm GHC2*, as a generic hierarchical clustering algorithm. GHC2 has been observed in several experiments to be faster than GHC1 despite its larger worst case running time. The main feature of GHC2 is its generality.

---

1. Put every object of P into a single-element cluster C_i, $1 \leq i \leq n$
2. For all clusters C_i compute the best merge cluster $BMC(C_i)$.
3. **WHILE** there is more than one cluster **DO**
   (a) **FOR** all clusters C_i **DO**
      i. **IF** $BMC(BMC(C_i)) = C_i$ **THEN**
         A. Let C_i be the union of C_i and $BMC(C_i)$ and mark the resulting cluster as new.
         B. Remove $BMC(C_i)$
   (b) Compute the best merge cluster $BMC(C_i)$ for all clusters C_i marked as new or with $BMC(C_i)$ marked as new or having been removed. *; see also text*
   (c) Delete all new markings

Figure 7.4: Generic hierarchical clustering Algorithm GHC2 for $\mathcal{C}^d$ with worst case running time $O(dn^3)$.

When merging a cluster C_i with another cluster we define a *best merge cluster of* C_i, $BMC(C_i)$, as a cluster which minimizes the resulting dissimilarity among all clusters C_k, $k \neq i$. For single points this usually coincides with the well known nearest neighbour of a point. If the best merge cluster of C_i has itself C_i as its best merge cluster, i.e. $BMC(BMC(C_i)) = C_i$, we call this a *merge pair*.



(a) (b)

Figure 7.5: Cycle of best merge clusters (a) and best merge cluster tree (b) with merge pair as root (dotted ellipse).

Note the special case if $BMC(C_i)$ is not unique, i.e. where C_k and C_l, $k \neq l$, both give the same dissimilarity when being merged with C_i. This may cause cyclic relations; for an example with a 4-cycle in $\mathcal{C}^3$ see Figure 7.5(a). Due to the symmetry of the δ-function all dissimilarities in a non-trivial cycle (a trivial cycle is just a merge pair) must of course be equal. In this case for any cluster C_i the cluster with the lowest index of all elements in $BMC(C_i)$ is taken to avoid non-trivial cycles where no merge pair may exist. Thus the graph of best merge cluster relations can be seen as a forest, where the roots of the trees consist of the merge pairs (cf. Figure 7.5(b)). Moreover, since the graph is free of non-trivial cycles, at least one merge pair always exists.

From the fact that the δ-function is monotonous (property 2) it follows that all existing merge pairs can be merged in parallel. After merging two clusters the BMC-information of a third cluster C_k can change if and only if the old $BMC(C_k)$ was one of the merged clusters or the new $BMC(C_k)$ is the new cluster. Thus after merging all merge pairs we have to update the information about best merge clusters accordingly. This means that we recompute the BMC-information of every new cluster symmetrically, i.e. while searching for the BMC of the new cluster we may also update the information for all considered clusters (for implementation details see below).

The new approach now follows these lines. It first computes all best merge clusters. Then it merges all merge pairs, updates the information about best merge clusters only where necessary and repeats this until only one final cluster is left. The detailed algorithm is given in Figure 7.4. From the above discussion it is straight forward to see that Algorithm GHC2 indeed produces the required cluster tree.

Note that the construction sequence of the cluster tree is not monotonous for Algorithm GHC2, i.e. if necessary for an application we have to sort the levels of the tree by non-decreasing dissimilarity. Since the tree has only linear size this can clearly be done in $O(n \log n)$ time.

The worst case running time of Algorithm 2 is $O(dn^3)$: The initialization in step 2 requires to compute the distance function between all vertices, which takes $O(dn^2)$ time. The for-loop of step 3a repeats $O(n)$ times and the body of the loop can be done in constant time. Although in most cases in step 3b only a small number of clusters is concerned, in the worst case we have to update the information about the best merge clusters for a linear number of clusters. Thus step 3b needs $O(dn^2)$ time (cf. the analysis of the previous algorithm).

If the dissimilarity of a cluster has to be from $\{0, \ldots, d\}$ (e.g. for the diameter or radius of a cluster) and if $BMC(C_i)$ is unique, then every iteration of the while loop in step 3 increases the minimum dissimilarity by at least one. Since the maximum dissimilarity is d, we have at most d iterations and thus an overall running time of $O(d^2n^2)$. If $BMC(C_i)$ is not unique we may have only one merge pair per iteration, i.e. the while loop must repeat O(n) times. The same is true if the δ-function does not fulfill δ-function property 4). This gives the worst case time of $O(dn^3)$.

Let us point out some implementation details of step 3b of Algorithm GHC2. When computing the new $BMC(C)$ for a new cluster C, we have to distinguish between two kinds of dissimilarity functions. If for any agglomeration step $C = C_i \cup C_j$ we have

$$\min\{d_H(C_k, C_i), d_H(C_k, C_j)\} < d_H(C_k, C) \tag{7.1}$$

for any third cluster C_k, $k \neq i, j$, we clearly need not to update $BMC(C_k)$ as long as $BMC(C_k) \neq C_i, C_j$. Since $BMC(C_k)$ achieved a smaller dissimilarity with C_k than with the "better" one of C_i and C_j, C can never be the new $BMC(C_k)$.

But if inequality 7.1 is violated then $d_H(C_k, C)$ might be less than $d_H(C_k, BMC(C_k))$ even when $BMC(C_k) \neq C_i, C_j$. Another possibility is that $d_H(C_k, C) = \min\{d_H(C_k, C_i), d_H(C_k, C_j)\}$. If in this case the index of C is now smaller than the index of the old $BMC(C_k)$ we have to update the $BMC(C_k)$ to avoid non-trivial cycles, though the value of $BMC(C_k)$ remains unchanged (see above for a discussion on cycles).

In both cases we therefore have to recompute the BMC information in step 3b in a symmetric way, i.e. whenever we compute the dissimilarity between a cluster C marked as new and an arbitrary cluster C_k, we not only have to check whether this provides a new $BMC(C)$ but also if we get a new $BMC(C_k)$. Clearly this does not affect the running time of the algorithm, since it requires only a simple constant time comparison and there are no additional distance computations necessary. We thus implemented the additional symmetric test for all methods based on Algorithm GHC2.

Note that inequality 7.1 does not follow from the monotonicity of a dissimilarity function, since δ-function property 2 only has to hold for the relations given by the "edges" of the cluster tree. For example A_{mean}-link (see Section 7.2.4) is shown to be monotonous (cf. Lemma 7.6) but obviously violates inequality 7.1. In fact A_{mean}-link and centroidal clustering minimizing the mean distance (see Section 7.2.5) are the only methods we will present for GHC2, where this is the case and thus the symmetric test is required.

Experiments show that in step 3b only the information of a small number of clusters has to be updated and the while loop of step 3 repeats much less than $O(n)$ times. Thus Algorithm GHC2 is expected to be very efficient for applications. For more details see Section 7.4.

7.2.3 Complete Linkage Clustering

In this subsection we consider the Complete-Linkage Clustering method (C-link) which was first mentioned by Sørensen [105] and also very early used in [63, 69, 73] (for a historical survey see e.g. Chapter 1.6 in [36]). To overcome the chaining effect of S-link, C-link uses a different criterion to choose the next two clusters: it minimizes the diameter (maximum distance of objects in one cluster) of the resulting cluster. Therefore the δ-function of C-link for a cluster C is $d(C) = \max_{x,y \in C} d_H(x, y)$. The resulting clusters are more 'sphere-like' and are expected to produce better results for practical problems [51, 52]. Experiments on this method show that it produces rather even, i.e. equal sized, clusters, which may be suitable for many applications. For an example of a planar point set see Figure 7.2(b).

The clear disadvantage of C-link is the more difficult and thus time consuming computation. Another reason which caused C-link to be unpopular was given by the paper of Defays [35]. He gave an $O(n^2)$ implementation of C-link. Unfortunately this implementation does not really compute C-link but only a variant of it (although this was not pointed out explicitly in the paper). It is shown in [51] that this variant lacks some of the nice properties of C-link and thus produces worse results. In fact, while C-link is reported to be one of the best methods there (beside A-link, see Section 7.2.4), the results obtained by Defays' implementation are the worst. Unfortunately the fast running time proposed by Defays prevented other researcher for considering new implementations of C-link and therefore only straightforward and thus slow codes exist.

The best linear space implementation for general objects given in the literature so far had a worst case time bound of $O(n^4)$, the fastest known implementation was the version used by Voorhees [111] (see also [51]), running in time $O(n^3)$, but using space $O(n^2)$. In a recent paper [4] practical implementations, with running time $O(n^2 \log n)$ for quadratic space, and $O(n^3)$ time for linear space, are given: The $O(n^2 \log n)$ time solution is based on n heaps, one for every subcluster. Each heap manages the dissimilarities of its cluster to all other clusters. Thus we can find the next two clusters to merge in linear or even logarithmic time, if an additional heap is used to store the minima. An update step for the dissimilarity information can be done in time $O(n \log n)$, which gives the promised bound.

For linear space there are two algorithms. The first one is very similar to the previously known $O(n^4)$ version, except that from the new implementation it is obvious to see that one merge step needs at most $O(n^2)$ distance computations. Thus the $O(n^3)$ time bound follows immediately. The second approach uses the concept of nearest neighbour pairs, very similar to the best merge clusters of Algorithm GHC2 (cf. Section 7.2.2). It thus has a worst case running time of $O(n^3)$, but experimental results show that it most likely runs in $O(n^2 \log n)$ time for practical problems (see [4], cf. also Section 7.4).

Most recently Krznaric and Levcopoulos gave an $O(n \log^2 n)$ time C-link algorithm for the special case of planar point sets, see [66].

Note that all mentioned implementations assume that the distance between two objects can be computed in constant time. To this end either the distance matrix has to be given explicitly, where then already the input has quadratic size, and we thus do not need to consider linear space solutions. Or, if the objects (points) lie e.g. in the real space $\mathcal{R}^d$, we can compute the distance from their coordinates. But then the dimension d must be of constant size. Considering arbitrarily high dimensions we clearly have to multiply the given time bounds by a factor $O(d)$. If we may use quadratic space we can precompute the distance matrix in $O(dn^2)$ time, and get thus an $O(n^2(\log n + d))$ algorithm.

For the special case of binary presentable objects we can now use the generic algorithms developed in the last section.

The diameter of a cluster can simply be computed by taking the maximum of the distances between all pairs of points. Thus it is straight forward to see that the diameter fulfills δ-function properties 1), 2) and 4) of our framework. The next lemma shows that it fulfills also property 3).

Lemma 7.3 *If a cluster C_k is build by a merge step in the C-link tree from two clusters C_i and C_j then there exists a pair of vertices $v_i \in C_i$, $v_j \in C_j$ which realize the diameter of C_k. That is the diameter of C_k can be computed with $O(|C_i| \cdot |C_j|)$ distance computations.*

Proof. Suppose that no such pair exists. Let r be the diameter of C_k. Then w.l.o.g. r is realized by vertices of C_i which means that the diameter of C_i is r. Looking at the history of C_i in the C-link tree there was a merge step where C_i or a subset of it achieved the diameter r the first time. At this stage the two subclusters merged had both a diameter of less than r. Thus merging one of them with C_j (or any at this time existing subcluster of C_j with diameter less than r) would have given a cluster with diameter strictly less than r. But then C-link would have merged these two subcluster instead, which is a contradiction to the current existence of C_i and C_j. $\square$

From the above results we conclude that we can use Algorithm GHC1. Note that, as mentioned in Section 7.2.2, we do not need to include the line "j=i+1" in step 2biiA.

Theorem 7.2 *Given a set $P \subseteq V^d$ with n elements. The C-link tree of P can be computed in $O(d^2n^2)$ time and $O(dn)$ space.*

7.2.4 Average Linkage Clustering

In this subsection we consider hierarchical clustering methods where the dissimilarity of a cluster depends on the sum or mean of distances between vertices. For general implementations and applications of this method see e.g. Voorhees [111]. We first show that these dissimilarities can be computed very efficiently in the d-cube case.

Lemma 7.4 *For all clusters $C, C_i, C_j \subseteq V^d$ arising during the hierarchical clustering of n objects from V^d the sum (mean) of distances between points within C or between points of C_i and C_j can be computed in $O(d)$ time, using $O(dn)$ additional space for the whole cluster tree.*

Proof. For all clusters C_i existing during the hierarchical clustering process we store for every bit position b, $b = 1, \ldots, d$, the number $n_0(C_i, b)$ of elements from C_i with a '0' on position b and the number $n_1(C_i, b)$ of elements from C_i with a '1' on position b. To do this we need $O(d)$ additional space per cluster and thus $O(dn)$ additional space for the whole cluster tree.

Clearly an update for a merge step takes only $O(d)$ additional time. Moreover we also know the number of elements of a cluster, thus once we have the sum of distances we can compute the mean of them in constant time. (In fact any reasonable implementation will only store the number n_i of elements in a cluster C_i and $n_0(C_i, b)$ since $n_1(C_i, b) = n_i - n_0(C_i, b)$.)

To compute the sum of distances between the elements of two clusters C_i and C_j one usually has to compare all pairs of elements and sum up the number of differing bits per pair. We now do this separately for every bit position b and then sum over all d positions. Clearly a pair of elements contributes one to the sum iff they have a different bit on position b. This means that any element of C_i with '0' on position b adds one to the sum whenever compared with an element of C_j with '1' on position b. Of course the same is true if we exchange '0' and '1' and thus summing over all positions b we get $\sum_{b=1}^{d} (n_0(C_i, b)n_1(C_j, b) + n_1(C_i, b)n_0(C_j, b))$ for the sum of all distances between C_i and C_j. This sum can easily be computed in $O(d)$ time using the information stored in the cluster tree. For the sum of distances between the elements of a single cluster C we similarly get $\sum_{b=1}^{d} n_0(C, b)n_1(C, b)$. □

The Average-linkage method (A-link, sometimes also called Group Average Linkage, [51, 79]) uses the mean distance between the points of two clusters as their cluster distance, i.e. as the dissimilarity of the resulting cluster after a merge step. Note that this δ-function does not only depend on a cluster but also on the way it was built. Thus we can show properties of this δ-function only when used for hierarchical clustering. For δ-function property 1) we just have to define the mean distance of a single point as zero. Property 3) follows from Lemma 7.4 and property 4) is obviously violated, thus we may only use Algorithm GHC2 of the last section. It remains to prove property 2), i.e. that A-link builds a monotonous cluster tree. It is well known (see e.g. [21]) that A-link is monotonous, not only in the geometric or hypercube case. Nevertheless, we give the proof of the following lemma for the sake of completeness and because some of the subsequent proofs rely on the same cluster hierarchy.

Lemma 7.5 *For all merge steps $C = C_i \cup C_j$ of A-link we have $d(C) \geq \max\{d(C_i), d(C_j)\}$.*

Proof. W.l.o.g. assume that $d(C_i) \geq d(C_j)$ and suppose that $d(C) < d(C_i)$. We prove the lemma by induction over the merge steps. Let the merge step $C = C_i \cup C_j$ be the first step where an inversion occurs. Then by induction the last merge operation in this subtree was the agglomeration of $C_i = C_i' \cup C_i''$. Since $d(C) < d(C_i)$ the mean distance between points of C_i and C_j is less than the mean distance between points of C_i' and C_i''. But then either the mean distance between points of C_i' and C_j or C_i'' and C_j is also less than $d(C_i)$. Thus A-link would merge either C_i' and C_j or C_i'' and C_j instead of C_i' and C_i'', a contradiction to the existence of C_i. □

Using Algorithm GHC2 we thus get

Theorem 7.3 *Given a set $P \subseteq V^d$ with n elements. The A-link tree of P can be computed in $O(dn^3)$ time and $O(dn)$ space.*

It is quite unsatisfactory that the δ-function of A-link depends on the history of a cluster. One reason for the way A-link is defined comes from the attempt to use the generic SAHN clustering methods (SAHN stands for sequential, agglomerative, hierarchical, nonoverlapping, see e.g. [34]). We now give two modified versions of A-link which do not fit into the scheme of SAHN but seem to be more natural at least to the author.

For A_{sum}*-link* (A_{mean}*-link*) we define the dissimilarity of a cluster C as the sum (mean) of all distances between points from C. Clearly the δ-functions do not longer depend on the history of a cluster. From the above discussion it follows that we need only to consider the δ-function property 2). The dissimilarity of A_{sum}-link obviously fulfills property 2) since adding points to a cluster always increases the sum of Hamming distances. For the mean distance this is in general not true as can be seen by the following example. Consider the two clusters $C_1 = ((0,0,0,0),(1,1,1,0))$ and $C_2 = ((0,0,0,1),(1,1,1,1))$ of $\mathcal{C}^4$ with mean distance 3 each. Merging them to $C = ((0,0,0,0),(0,0,0,1),(1,1,1,0)),(1,1,1,1))$ gives a cluster C with mean distance of $2\frac{2}{3}$. Clearly clusters C_1 and C_2 would not occur during a hierarchical clustering process of these four objects. Still A_{mean}-link defines a monotonous cluster tree, i.e., forces property 2.

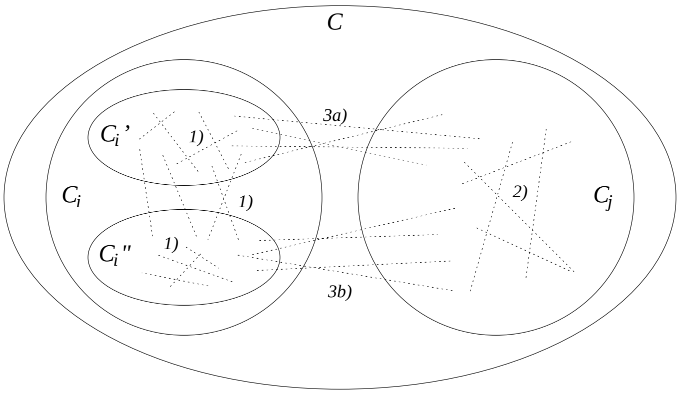


Figure 7.6: Cluster hierarchy for the proof of Lemma 7.6

Lemma 7.6 *Let the dissimilarity* $d(C)$ *be the mean distance between the vertices of the cluster* C*. Then for the hierarchical clustering method* A_{mean}*-link the dissimilarity is monotonous, i.e. for any cluster* $C = C_i \cup C_j$ *we have* $d(C) \geq \max\{d(C_i), d(C_j)\}$*.*

Proof. We prove the lemma by induction over the merge steps. It is clear that merging single points can not violate the monotonicity. So let the merge step $C = C_i \cup C_j$ be the first step where an inversion occurs, i.e. $d(C) < d(C_i)$ where we assume w.l.o.g that $d(C_i) \geq d(C_j)$. Then by induction the last merge operation in this subtree was the agglomeration of $C_i = C_i' \cup C_i''$, refer to Figure 7.6. Note that this step was monotonous.

The distances in C consist of three types: 1) the distances within C_i, 2) the distances within C_j and 3) the distances between vertices from C_i and vertices from C_j. Since the mean of all distances of C is less than $d(C_i)$ the mean of the distances of type 2 and 3 must be less than $d(C_i)$, too. Now split the distances of type 3 into two parts: 3a) distances between C_i' and C_j, 3b) distances between C_i'' and C_j. W.l.o.g we assume that the mean of the distances of type 3a is less or equal the mean of the distances of type 3b. Then clearly the mean of the distances of type 2 and type 3a is less than $d(C_i)$ (because $d(C_j) \leq d(C_i)$).

On the other hand we have $d(C_i') \leq d(C_i)$ since by induction this agglomeration step was monotonous. Therefore the mean of all distances within C_j (type 2), between C_j and C_i' (type

3a) and within C_i' is strictly less than $d(C_i)$. But thus $d(C_j \cup C_i') < d(C_i)$, i.e. merging C_j and C_i' results in a smaller dissimilarity than merging C_i' and C_i'', a contradiction to the current existence of C_i in the cluster tree. Thus the mean distance function is indeed monotonous for hierarchical clustering. □

It therefore follows that we can use Algorithm GHC2 to compute the A$_{sum}$-link and A$_{mean}$-link trees.

Theorem 7.4 *Given a set $P \subseteq V^d$ with n elements. The A_{sum}-link and A_{mean}-link tree of P can be computed in $O(dn^3)$ time and $O(dn)$ space.*

Note that for all presented versions of hierarchical average linkage clustering the dissimilarity of a cluster can be computed very efficiently. Taking only $O(d)$ time it is much faster than required by δ-function property 3). Unfortunately the true bottleneck of Algorithm GHC2 is the number of updates of the best merge cluster information. Thus in the worst case analysis the fast dissimilarity computation has no effect on the running time. But the experiments given in Section 7.4 show that for applications the number of necessary updates is far below the worst case. Therefore Algorithm GHC2 together with the fast dissimilarity computation of Lemma 7.4 gives a quite efficient solution for hierarchical average linkage clustering.

7.2.5 Hierarchical Centroidal Clustering

Centroid clustering methods for objects characterized by k-tuples of real numbers are based on the representation of clusters by single points in k-dimensional real space. Clusters (i.e. their centroid points) being merged are replaced by a single centroid point. For the d-cube case we can either allow arbitrary centroid points in $\mathcal{R}^d$ or we restrict the set of centroid points to elements of V^d. Moreover for a cluster $C \subseteq V^d$ we may only allow an element of C as its centroid point, which means that every cluster is represented by its "most typical" object. In this last case we may call the centroid point the *kernel* of C.

To measure the dissimilarity of a cluster C we either take the maximum distance of a point from C to the centroid point (*radius* of C), the sum of these distances or their mean (*mean radius*). Since the number of vertices is constant for a given cluster C, minimizing the mean distance coincides with minimizing the sum of distances from the centroid point. Thus the one-centroid problem can be solved for both optimization criteria similarly. Note however that this holds only for single clusters and is in general not true for the entire cluster tree. Minimizing the mean distance on one hand and the sum of distances on the other may result in different cluster trees.

Thus the centroid point of C has always to be chosen such that it minimizes the maximum (sum, mean) distance to points in C. We use the Hamming distance if the centroid point is in V^d and the L_1-metric for arbitrary centroid points from $\mathcal{R}^d$. We therefore distinguish between three different domains for centroid points and three different methods to determine the centroid point of a cluster and to compute its dissimilarity.

Minimizing the Radius of a Cluster

We first consider hierarchical centroid clustering minimizing the radius of the cluster. To this end let for a cluster $C \subseteq V^d$ the dissimilarity $d(C)$ be the maximum distance from points in C

to its centroid point. For an arbitrary point p we denote with $d(C,p)$ the maximum distance from points in C to p. Note that for any set P of points containing the centroid of C we have $d(C) \leq d(C,p)$ for all $p \in P$. In the following lemma we now show that this δ-function is indeed monotonous, i.e. fulfills δ-function property 2).

Lemma 7.7 *For $C \subseteq V^d$ let the dissimilarity $d(C)$ be the maximum distance between vertices of C and the centroid p of C, where p minimizes this maximum distance among all possible points in either $\mathcal{R}^d$, V^d or C. Then for the hierarchical centroid clustering the dissimilarity $d(C)$ is monotonous.*

Proof. If centroid points are from either V^d or R^d the lemma is obviously correct. Adding vertices to a cluster C can never decrease the radius of C since we choose the centroid point of C always such that it minimizes the maximum distance to points in C.

When we restrict the centroid point of a cluster C_i to be a vertex of the cluster itself, another cluster C_j may contain a vertex with a smaller maximum distance to vertices in C_i and thus the above argumentation does not hold. We therefore utilize the cluster hierarchy given in the proof of Lemma 7.6 and show Lemma 7.7 by induction over the merge steps.

Clearly merging single points can not violate the monotonicity. So let the merge step $C = C_i \cup C_j$ be the first step where an inversion occurs, i.e. $d(C) < d(C_i)$ where we assume w.l.o.g that $d(C_i) \geq d(C_j)$. Then by induction the last (monotonous) merge operation in this subtree was the agglomeration of $C_i = C_i' \cup C_i''$.

The kernel p of C can not lie in C_i because then $d(C,p)$ has to be at least $d(C_i,p) \geq d(C_i)$, a contradiction to the inversion. But then $p \in C_j$ and since $C_i' \cup C_j \subset C$ we have $d(C_i' \cup C_j) \leq d(C_i' \cup C_j, p) \leq d(C,p) = d(C) < d(C_i)$, i.e. merging C_j and C_i' results in a smaller dissimilarity than merging C_i' and C_i'', a contradiction to the current existence of C_i in the cluster tree. Thus the maximum distance from the centroid point is monotonous for all mentioned domains. □

If centroid points are allowed to be from $\mathcal{R}^d$ we can reduce the optimization problem of finding the centroid point minimizing the maximum distance to a point in C to linear programming by doing a binary search for the minimum distance r over the possible range $\{0,\ldots,\frac{d}{2}\}$. (Note that the point $(\frac{1}{2},\ldots,\frac{1}{2})$ always achieves the upper bound.) We then ask whether for a given distance r there exists a centroid point $p \in \mathcal{R}^d$ for C such that the L_1-distance to every point $c_i \in C$ does not exceed r. We can formulate this as a linear inequality $x_i \cdot p \leq r - n_1(c_i)$, for every $c_i \in C$. Thereby $n_1(c_i)$ is the number of ones in c_i and the j^{th} component x_i^j of x_i is set to 1 if the corresponding bit c_i^j of c_i is zero, and to -1 else. (The dot-product $x_i \cdot p$ is given by $\sum_{j=1}^{d} x_i^j p^j$.) Finding a feasible solution for this linear system (i.e. a centroid point with the required radius r) can be solved in polynomial time (e.g. by Karmarkar's interior point methods [96, 99]).

For the case that centroid points are restricted to V^d we have to do the binary search only for integer distances from $\{1,\ldots,d\}$. However, in the context of linear programming the problem if for a given distance r there exists a centroid point in V^d, is now an instance of integer programming (see e.g. [96, 99]). In general integer programming asks whether a given system of linear inequalities, in d variables and with integer coefficients, has an integer solution. This problem is well known to be NP-complete [43, 96], especially the variant in which the solution variables are required to belong to $\{0,1\}$ (zero-one integer programming, shown to be NP-complete by a simple transformation from 3SAT). For an overview and a discussion on the complexity of integer programming see e.g. Chapter 9.5 in [83] and references given therein.

Note however, that this does clearly **not** imply that minimizing the radius to a centroid point in V^d is also NP-complete. To our knowledge the complexity class of this problem is still unknown. Moreover note that we consider the running time as a function of n and d, i.e. we require that it is polynomial in both parameters. Obviously a straight forward approach runs in time $O(dn2^d)$ and thus we get a linear time solution for constant d. If n is a small constant, the problem can also be solved in polynomial time, e.g. in $O(d^3)$ for $n = 3$, see next subsection.

For complexity reasons we therefore restrict our investigations to the simpler case where for a cluster C its centroid point has to be contained in C.

Theorem 7.5 *Given a set $P \subseteq V^d$ with n elements. We require that the centroid point of a cluster has to be contained in the cluster and minimizes the maximum distance of points from the cluster to it. Then the hierarchical centroid clustering tree of P can be computed in $O(d^2n^2)$ time and $O(dn)$ space.*

Proof. Properties 1) and 4) for δ-functions are obviously fulfilled and Lemma 7.7 shows that also property 2) holds. To apply Algorithm GHC1 it thus remains to show that for an agglomeration step in the cluster tree we can compute the kernel of two clusters C_i and C_j in $O(d|C_i| \cdot |C_j|)$ time. To this end for every point p_i of a cluster we store the maximum distance to another point within the same cluster. This requires only $O(n)$ additional space for the whole cluster tree and the centroid point of a cluster C can thus be obtained in $O(|C|)$ time by taking the point with the minimum value. The kernel of the union of two clusters C_i and C_j (i.e. an agglomeration step) now can be computed in $O(d|C_i| \cdot |C_j|)$ time. For any point of C_i (and symmetrically for C_j) we have to compute the maximum distance to a point in C_j (C_i, respectively) and store the maximum of this distance and the previous maximum of the point. From this information we can compute the kernel for the union of two clusters in the promised time bound and thus the whole cluster tree with Algorithm GHC1 in $O(d^2n^2)$ time and $O(dn)$ space. □

As mentioned in Section 7.2.2 we have to use the version of GHC1 where after an agglomeration step we reconsider all clusters C_j, $j > i$. Thus we have to include the line "j=i+1" in step 2biiA.

A Special Case for Minimizing the Centroid Radius

In this subsection we consider the radius problem for $n = 3$, which is a special case of finding a centroid point which minimizes the radius to a given set of points. More formally we have
INSTANCE: Given are 3 vertices v_1, v_2 and v_3 of $\mathcal{C}^d$
PROBLEM: Report the vertex $x \in V^d$ which minimizes the radius to the given vertices, i.e. $x = \min_{x \in V^d}\{\max_{v \in \{v_1,v_2,v_3\}} d_H(x, v)\}$

Theorem 7.6 *The radius problem for $n = 3$ can be solved in $O(n^3)$ time and $O(n)$ space.*

Proof. Let the three given vertices be v_1, v_2 and v_3, where $v_i = (v_{i_1}, v_{i_2}, \ldots, v_{i_d})$ with $v_{i_j} \in \{0, 1\}$, $1 \leq i \leq 3$, $1 \leq j \leq d$. Now arrange these three vectors in a $3 \times d$ matrix and consider for every bit position j, $1 \leq j \leq d$, the column vector, each consisting of three elements.

Note that for equal column vectors only the number of corresponding zeros and ones of the centroid point is decisive, but not their exact bit position. We thus can group identical column vectors together and ask for the number of $1s$ of the centroid point at corresponding positions in order to minimize the radius. Moreover if the number of $1s$ of the centroid point is given for every group, the time to compute the maximal distance from the centroid point to the three vertices (i.e. the radius) is clearly linear in the number of groups.

For three vertices there are at most $2^3 = 8$ different such column vectors. For the two vectors $(0, 0, 0)$ and $(1, 1, 1)$ the corresponding bit of the optimal centroid point has obviously to be 0 and 1, respectively. The remaining six types of column vectors build three pairs of complementary vectors. We thus can replace each column vector with, say, $v_{1_j} = 1$ by its complementary vector and search for the optimal centroid point of the new set. From the solution for this reduced set we can easily obtain the solution of the original problem: We just have to reverse the bits of the optimal centroid point for any position where we have taken the complementary column vector. This gives us the correct solution, since it does not change the distance between the centroid point and the vertices, i.e does not affect the radius.

The reduced problem has at most three different kinds of column vectors, namely $(0, 0, 1)$, $(0, 1, 0)$, and $(0, 1, 1)$, respectively. We now ask for the number of vectors of each type for which we assign a 1 bit to the optimal centroid point.

Since any of the three vectors exists at most d times the number of possible $1s$ for the centroid point is in the range $\{0, 1, \ldots, d\}$, and thus there are at most $(d + 1)^3$ different possibilities. In fact $\left(\frac{d}{3}\right)^3$ is a tighter upper bound, since the sum of the cardinality of the groups can not exceed d, but this still remains $\Theta(d^3)$. If we fix the number of $1s$ for every group we can compute the radius in constant time, as mentioned above. Thus, by exhaustive checking all possibilities, we can find the centroid point minimizing the radius in $O(d^3)$ time. □

Minimizing the Mean Radius and Sum of Distances

Next we consider hierarchical centroid clustering minimizing the mean radius of the cluster, which is equivalent to minimize the sum of distances to the centroid point, i.e. both criteria generate the same centroid point. For a cluster $C \subseteq V^d$ let now the dissimilarity $d(C)$ be the mean distance (sum of distances) from points of C to its centroid point and similarly $d(C, p)$ the mean distance (sum of distances) from points of C to p. We first consider whether the δ-functions are monotonous, i.e. fulfill δ-function property 2).

Lemma 7.8 *For $C \subseteq V^d$ let the dissimilarity $d(C)$ be the mean distance (sum of distances) between vertices of C and the centroid p of C, where p minimizes this mean distance (sum of distances) among all possible points in either $\mathcal{R}^d$ or V^d. Then for the hierarchical centroid clustering the dissimilarity $d(C)$ is monotonous.*

Proof. The monotonicity of the sum of distances is obvious since by adding vertices (and thus distances) to a cluster we can only increase the sum. (We cannot decrease the sum of distances by changing the centroid point, since the "old" centroid point already achieved the optimum.)

For the mean radius this is less obvious: To prove the lemma we basically reuse the idea of the proof of Lemma 7.6 (cf. Fig. 7.6), doing it by induction over the merge steps. It is clear that merging single points can not violate the monotonicity. So let the merge step $C = C_i \cup C_j$ be the first step where an inversion occurs, i.e. $d(C) < d(C_i)$ where we assume w.l.o.g that $d(C_i) \geq d(C_j)$. Then by induction the last (monotonous) merge operation in this subtree was the agglomeration of $C_i = C_i' \cup C_i''$.

Let p be the centroid point of C. From the inversion we have $d(C,p) = d(C) < d(C_i) \leq d(C_i, p)$. W.l.o.g. we assume that $d(C_i', p) \leq d(C_i'', p)$ and thus $d(C_i'', p) > d(C, p)$ since $d(C_i, p) = d(C_i' \cup C_i'', p)$. But then $d(C_i' \cup C_j) \leq (C_i' \cup C_j, p) = d(C \setminus C_i'', p) < d(C,p) = d(C) < d(C_i)$ i.e. merging C_j and C_i' results in a smaller dissimilarity than merging C_i' and C_i'', a contradiction to the current existence of C_i in the cluster tree. Thus $d(C)$ is indeed monotonous if we allow centroid points to be from $\mathcal{R}^d$ or V^d. $\square$

Thus if centroid points are allowed to be in V^d or $\mathcal{R}^d$ then both δ-functions are monotonous. It is now interesting to observe what happens if we restrict the domain for centroid points to the set of vertices contained in cluster. Surprisingly in this case the two δ-functions have quite different monotonicity properties.

Lemma 7.9 *For $C \subseteq V^d$ let the dissimilarity $d(C)$ be the sum of distances between vertices of C and the centroid p of C, where p minimizes this sum of distances among all possible points in C. Then for the hierarchical centroid clustering the dissimilarity $d(C)$ is monotonous.*

Proof. The first part to prove the monotonicity is identical with the second paragraph of the proof of Lemma 7.8. Now let p be the kernel of C and observe that p can not be contained in C_i since then it immediately follows that $d(C) > d(C_i)$. Thus $p \in C_j$ and we have $d(C_i) > d(C) = d(C_i, p) + d(C_j, p) > d(C_i', p) + d(C_j, p) = d(C_i' \cup C_j, p) \geq d(C_i' \cup C_j)$. i.e. merging C_j and C_i' results in a smaller dissimilarity than merging C_i' and C_i'', a contradiction to the current existence of C_i in the cluster tree. $\square$

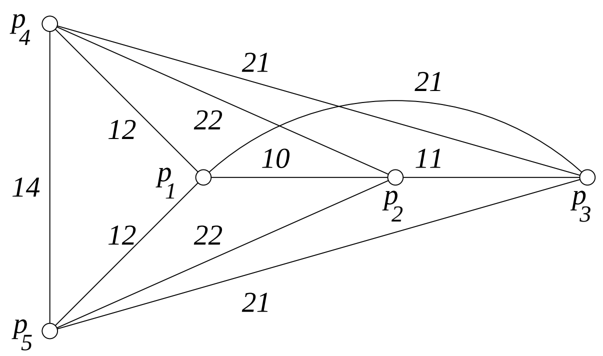


Figure 7.7: A five point example showing that hierarchical centroid clustering is not monotonous when centroid points are restricted to cluster points.

We now consider the case where for the kernel $p \in C$ $d(C)$ is the mean distance from the remaining points of C to p. The problem here is that in the proof of Lemma 7.8 either

$$
\begin{aligned}
p_1 &= (\ \mathtt{0000000000\ 0000000000\ 000000000}\ ) \\
p_2 &= (\ \mathtt{1111111111\ 0000000000\ 000000000}\ ) \\
p_3 &= (\ \mathtt{1111111111\ 1111111111\ 100000000}\ ) \\
p_4 &= (\ \mathtt{0000000000\ 0000011111\ 100111111}\ ) \\
p_5 &= (\ \mathtt{0000000000\ 1111100000\ 111001111}\ )
\end{aligned}
$$

Table 7.1: The five points of Fig. 7.7 embedded into $\mathcal{C}^{29}$.

$d(C_i) \leq d(C_i, p)$ or $d(C_i' \cup C_j) \leq d(C_i' \cup C_j, p)$ need not to be true as was already pointed out in the proof of Lemma 7.7. In fact Figure 7.7 gives a five point example where this is the case and thus an inversion occurs. The given distances not only fulfill the triangle inequality but can easily be embedded into the hypercube of dimension 29 (cf. Table 7.1).

Let us give a brief explanation how the example works. In the first two steps of the hierarchy obviously points p_1, p_2 and p_3 are merged to a cluster C_A with kernel p_2 and mean radius $d(C_A) = 10.5$. For the next agglomeration step the two possibilities are either adding p_4 (or symmetrically p_5) to C_A, resulting in a cluster C_B with kernel p_1 or p_2 and mean radius $d(C_B) = 14\frac{1}{3}$, or merging p_4 and p_5 to C_B with $d(C_B) = 14$. Thus the third step merges p_4 and p_5 to C_B. After the last step all points are in the final cluster C and p_1 minimizes the mean radius, i.e. is the kernel of C. But the mean radius of C is 13.75 which is less than $d(C_B) = 14$, the promised inversion.

One key fact for this example is that $d(C_B) = 14 > d(C_B, p_1) = 12$ and thus the argumentation used to prove Lemma 7.8 can not be applied.

Let us now take the algorithmic point of view if the centroid points are from either $\mathcal{R}^d$ or V^d. As for average linkage (cf. Section 7.2.4) we denote by $n_0(C, b)$ the number of elements from C with a '0' on position b and with $n_1(C, b)$ the number of elements from C with a '1' on position b, $b = 1, \ldots, d$. We first show that the centroid point minimizing the mean distance (sum of distances) over all points in either V^d or $\mathcal{R}^d$ can be computed very efficiently in the d-cube case.

Lemma 7.10 *The centroid point $p \in V^d$ of a cluster $C \subseteq V^d$ minimizing the mean distance (sum of distances) of points of C to p can be computed in $O(d|C|)$ time. Moreover using the hierarchical clustering method for n objects from V^d the centroid point of the union of two clusters $C_i, C_j \subseteq V^d$ can be computed in $O(d)$ time, using $O(dn)$ additional space for the whole cluster tree.*

Proof. Let $C_i = \{p\}$ and $C_j = C$ and consider the special case of Lemma 7.4 for these two clusters. From the prove given there it follows that we can minimize the sum of the distances between C_i and C_j, and thus the mean of the distances between p and C, if we choose p to be the "average element" of C: every bit position b of p is set to '1' if $n_1(C, b) > n_0(C, b)$ and to '0' else (thus ties are broken by favoring the assignment of '0'). Clearly $n_1(C, b)$ and $n_0(C, b)$ can be computed in $O(|C|)$ time and we can thus obtain the centroid point for C in $O(d|C|)$ time.

If we compute the centroid points during the hierarchical clustering we can store $n_0(C_i, b)$ and $n_1(C_i, b)$ for all clusters C_i of the tree, using only $O(dn)$ additional space for the whole cluster tree. Then the centroid point of a cluster C_i or the union of two clusters C_i, C_j can be computed in $O(d)$ time. □

With the above observation we can also handle the dynamic version of one-centroid problems for a given set C. Adding a vertex to C or removing it can be done in $O(d)$ time and space if we store and update $n_0(C, b)$ and $n_1(C, b)$ appropriately.

Lemma 7.11 *Let $p \in \mathcal{R}^d$ be the centroid point of a cluster $C \subseteq V^d$ minimizing the mean distance (sum of distances) of points of C to p, using the L_1-metric. Then either $p \in V^d$ or there exists another centroid point $p' \in V^d$ with the same mean distance (sum of distances) to points in C.*

Proof. For a fixed index $b \in \{1, \ldots, d\}$ consider a single coordinate value p^b of the centroid point p. Clearly p^b must be in the range $[0 \ldots 1]$. The sum of the distances of all elements from C to p for the direction b is $n_o(C, b)p^b + n_1(C, b)(1 - p^b) = p^b(n_o(C, b) - n_1(C, b)) + n_1(C, b)$. We can minimize this sum by setting $p^b = 0$ if $n_o(C, b) \geq n_1(C, b)$ and $p^b = 1$ if $n_o(C, b) < n_1(C, b)$. Since this holds independently for all coordinates b we thus can minimize the sum and mean of distances by taking a centroid point from V^d. If $n_o(C, b) \neq n_1(C, b)$ for all directions $b = 1, \ldots, d$ then p is unique and in V^d. Else we have found one centroid point $p' \in V^d$ from an infinite set of possible centroid points with the same mean distance as p. □

Lemma 7.11 shows that we can handle the cases for V^d and $\mathcal{R}^d$ in the same way. Together with Lemma 7.10 it follows that properties 1) to 3) for δ-functions are fulfilled and therefore we can apply Algorithm GHC2.

Theorem 7.7 *Given a set $P \subseteq V^d$ with n elements. The centroid point of a cluster is from either V^d or $\mathcal{R}^d$ and minimizes the mean distance (sum of distances) of points from the cluster to the centroid point. Then the hierarchical centroid clustering tree of P can be computed in $O(dn^3)$ time and $O(dn)$ space.*

We use the same algorithm to minimize the mean radius and to minimize the sum of the distances. Moreover for a given cluster C the optimal centroid point is equivalent for both cases. Nevertheless the resulting cluster trees will be different in general, since the value of the dissimilarity is different for the two criteria. Therefore the choice which clusters are merged in the next step can be different and thus the methods result in different cluster trees.

As for average linkage the dissimilarity of a cluster can be computed very efficiently in $O(d)$ time and thus much faster than required by δ-function property 3). But again the bottleneck of Algorithm GHC2 is the number of updates of the best merge cluster information in the worst case. For experiments on centroid clustering see Section 7.4, showing that the average number of necessary update steps is small and thus Algorithm GHC2 is a practicable solution.

Finally let us consider the case where a centroid point has to be an element of the cluster itself. It is interesting to note that the one-kernel problem for minimizing the mean distance (sum of distances) can be solved in $O(dn^2)$ time with the same algorithm as for minimizing the maximum distance (cf. Theorem 7.5). Thus while minimizing the radius of a cluster gets easier when restricted to cluster points, the problem of minimizing the mean radius looses the monotonicity property and seems to be harder to compute in this case. For the problem of minimizing the sum of the distances to the kernel, we can again reuse the idea of Theorem 7.5 to perform the agglomeration step of two clusters C_i and C_j in $O(|C_i| \cdot |C_j|)$ time. Together with Lemma 7.9 we can thus apply Algorithm GHC2 and get

Theorem 7.8 *Given a set $P \subseteq V^d$ with n elements. The centroid point of a cluster has to be contained in the cluster and to minimize the sum of distances from points in the cluster to it. Then the hierarchical centroid clustering tree of P can be computed in $O(dn^3)$ time and $O(dn)$ space.*

7.3 Divisive Hierarchical and k-Clustering

Similar to agglomerative hierarchical methods, divisive hierarchical clustering builds a hierarchical cluster tree out of a set $S \subseteq V^d$ of n objects. Whereas agglomerative methods start with the smallest clusters, i.e. the single objects, and build the hierarchy by gradually merging them, divisive algorithms work the other way round and start with the cluster C containing all objects. Step by step they divide the "largest" existing cluster into two smaller subsets, according to a certain optimization criterion which determines the different methods, until after $n-1$ steps all remaining clusters are single objects. As for agglomerative clustering a hierarchical cluster tree is again a family of clusters such that any two clusters are disjoint iff they lie on different paths (seen from the root), or one includes the other iff they are on the same path. Thus the properties of the cluster trees are the same as described in see Section 7.2. It is easy to see that all divisive methods we present are monotonous, thus we do not mention this explicitly in the sequel.

7.3.1 Two-Clustering

The crucial step for divisive clustering is the iterated "optimal" splitting of a cluster C into two disjoint subsets C_1 and C_2. The choice of an optimal two-clustering determines the hierarchical clustering and thus the cluster tree. Therefore in the following two subsections we consider the optimal division of a set $C \subseteq V^d$ of hypercube vertices into two disjoint subsets C_1 and C_2. The first subsection gives two algorithms to minimize the maximum diameter of the resulting subclusters, and to maximize the minimum inter-cluster distance between them, respectively. In the second subsection we consider two-center problems.

Related to two-clustering is the k-clustering problem which is well known to be NP-complete for many cases. Since k-clustering turns out to be a subproblem of divisive hierarchical clustering for several criteria, we will give some brief remarks on k-clustering in Section 7.3.2.

Diameter and MST Two-Clustering

A first optimization criterion for two-clustering might be to separate objects of different clusters as well as possible, i.e. to maximize the minimum distance between them. Obviously this task is related to the minimum spanning tree (MST) problem, cf. Section 7.2.1. Removing the longest edge of the MST results in two connected components with the required property, and thus from Lemma 7.1 we immediately get the following:

Theorem 7.9 *Given a set $C \subseteq V^d$ with n elements. A two-clustering of C maximizing the minimum distance between objects of different clusters can be computed in $O(dn^2)$ time and $O(dn)$ space.*

---

1. Label every element $c_i \in C$ as 'blank' and put it into union i
2. **FOR** $diss = d$ **DOWN TO** 1 **DO**

   FOR $i = 1$ **TO** n **DO**

   FOR $j = i + 1$ **TO** n **DO**

   IF $d_H(c_i, c_j) = diss$ **THEN**

   (a) **IF** find(c_i)=find(c_j) **THEN**
      i. **IF** label(c_i)=label(c_j) **THEN**
         A. Label all elements still labeled 'blank' by either 'A' or 'B' arbitrarily
         B. According to their labeling separate the elements into the two clusters 'A' and 'B', minimizing the maximum diameter $diss$, achieved by c_i and c_j
         C. Output clusters 'A' and 'B' and **STOP**

   (b) **ELSE** {c_i and c_j belong to different sets}
      i. **IF** c_i and c_j are labeled 'blank' **THEN**
         A. Set label(c_i)='A' and label(c_j)='B'
         B. Union c_i and c_j
      ii. **ELSE IF** either c_i or c_j is labeled 'blank' **THEN**
         A. Label the 'blank' element conversely to the labeled element
         B. Union c_i and c_j
      iii. **ELSE** {both elements are labeled}
         A. **IF** label(c_i)=label(c_j) **THEN**
            Exchange all labels of the smaller set
         B. Union the sets containing c_i and c_j

Figure 7.8: Two-clustering algorithm minimizing the maximum diameter for $C \subseteq V^d$ with running time $O(d^2n^2)$.

Another evident optimization criterion for two-clustering is the minimization of the maximum diameter of the resulting clusters. In contrast to the previous criterion this means that we minimize the maximum dissimilarity of objects within the same cluster. It can thus be seen as the divisive counterpart of C-link. An $O(d^2n^2)$ time algorithm is given in Figure 7.8.

The basic idea of the algorithm is to consider the interpoint distances in non-increasing order and then to mark the respective points as belonging to different clusters as long as possible. To this end we use a disjoint-set data structure (also called union-find data structure, [30]) where we put all points into the same set iff their pairwise classification is already fixed. That is, a set contains all vertices where the membership of a vertex to either cluster 'A' or 'B' is fixed with respect to all other vertices of the same set. Thus a single set might be seen as a bipartite graph of objects, where one side is labeled 'A' and the other 'B'. The "edges" between the vertices of the graph relate to the distances already concerned, i.e. all distances above the actual length.

Whenever two points achieve a given distance we have to check whether they already belong to the same set. If this is the case we also consider their labels and if the labels are different nothing remains to do, since we already know that they have to belong to opposite clusters. If the points have the same label then we have found an odd cycle of pairwise relations, where the current "edge" represents the smallest distance within the cycle. In this case we know that at least one subcluster has the actual distance as lower bound for its diameter. We thus can label all still unlabeled vertices arbitrarily and assign all elements labeled 'A' to a cluster 'A' and equally for 'B'.

If the two points are belonging to different sets (e.g. they are still single points) we mark them as 'A' and 'B' accordingly and unify the two sets. If both points were already marked with the same label we exchange the labels of all elements of the smaller set, containing one of the points, before we unify them; see steps 2b i.-iii. of the algorithm given in Figure 7.8 for details. We thus maintain that a set contains only vertices with fixed pairwise relations and that different sets are independent from each other.

We repeat the above steps (in order of non-increasing distance) until an odd cycle of "AB-relations" appears. Since in this way we avoid an odd cycle of long distances as long as possible the algorithm clearly minimizes the maximum diameter of the subclusters.

We now consider the running time of the algorithm. For the union-find operations we use the linked-list representation of disjoint sets and the weighted-union heuristic of [30] where we always append the smaller list onto the longer. Thus a sequence of n union and m find operations takes $O(m + n \log n)$ time. Generating all point distances in non-increasing order is done by the three FOR loops and the first IF statement of step 2 and takes thus $O(d^2n^2)$ time. (Note that we can not use any other method of sorting the $O(n^2)$ distances since we are limited to linear space.) In 2a we have $O(n^2)$ find operations and in 2b $O(n)$ unions. Thus all disjoint set operations can be done in $O(n^2)$ time. The steps in 2ai. can be done in $O(n)$ and all label operations in 2b i.-iii. can be done within the same time as the related union operations. We thus get an overall running time of $O(d^2n^2)$.

Theorem 7.10 *Given a set $C \subseteq V^d$ with n elements. A two-clustering of C minimizing the maximum diameter of the resulting clusters can be computed in $O(d^2n^2)$ time and $O(dn)$ space.*

For practical implementations we can speed up the iteration of *diss* as mentioned for Algorithm GHC1: We decrease *diss* not only by one but reduce it to the next relevant value, i.e. for the next iteration we consider the largest distance which appeares during the current iteration and is strictly less than the actual value of *diss*. Clearly this does not affect the worst case running time, but it may significantely reduce the number of iterations of the outer loop. See Section 7.4 for experimental results.

Centroidal Two-Clustering

For centroidal two-clustering we ask in general for two centroids $c_1, c_2 \in V^d$ and a partition of the given set $S \subseteq V^d$ into two subsets S_1 and S_2 such that the maximum of a given function of these subsets with respect to c_1 and c_2 is minimized. For example if we want to minimize the maximum radius we have to optimize the term $\min_{c_1,c_2 \in V^d} \max_{p_i \in S} \min \{d_H(p_i, c_1), d_H(p_i, c_2)\}$. Note that in this example the division of S into S_1 and S_2 follows implicitly from the assignment of p_i to either c_1 or c_2.

As in Section 7.2.5 we distinguish between to different kinds of centers of a cluster: We either allow arbitrary centroid points from V^d as in the example above, where we will talk of two-centroid problems in the sequel. Or for a cluster $C \subseteq V^d$ we are restricted to an element of C as its centroid point, the so called kernel of C. These problems are then called two-kernel problems.

In the first case we know already that even the one-centroid problem is open if we want to minimize the radius. Clearly if we could solve the corresponding two-centroid problem efficiently, we could also solve the one-centroid problem: Let $S \subseteq V^d$ be the vertex set of the the one-centroid problem. Let S' be S embedded into a $(2d+1)$-dimensional hypercube by adding $d+1$ leading zeros to every vertex, and add the vertex $p = (1, \ldots, 1) \in V^{2d+1}$ to S'. The optimal solution of the two-centroid problem for S' now contains p and the optimal centroid of S as its two centroids. Therefore the complexity status of the two-centroid problem minimizing the maximum radius seems to be open, too.

In Section 7.2.5 we have seen that for the one-centroid problem the mean radius and the sum of distances can be minimized in $O(dn)$ time. Surprisingly the corresponding two-centroid problems seem to be much more difficult and in fact we are not aware of any polynomial time solution. One reason why the approach for the one-centroid problem can not be generalized is the fact that the bits of the centroid points are not longer independent. While for a single centroid point we can optimize every bit position separately, we now have a completely different situation. The bits of the two centroid points and the division of the vertex set S are highly nested. Changing a single bit of a centroid point c_1 may cause a different split of S and thus a different subset for which we want to optimize c_1. Note that once the partition of S is known we can easily compute the two centroid points in $O(dn)$ time.

Let us now briefly consider two-kernel problems. A straight forward solution is to compute all $\binom{n}{2}$ pairs of vertices from S as kernels and for each of these pairs to check the optimal assignment of the points in S. For minimizing the maximum radius we therefore just have to assign each vertex to its nearest kernel which can thus be done in $O(dn^3)$ time. But for e.g. minimizing the maximum mean distance it is not clear to the author whether this can be done as efficiently. Using the $O(dn^3)$ time two-clustering for a divisive hierarchical clustering leads to an $O(dn^4)$ time algorithm. Although some preliminary tests showed that this bound is rarely achieved for uniformly distributed point sets, the algorithm is too time consuming compared to other methods presented in this work. Thus we will not use centroidal two-clustering to develop divisive hierarchical clustering methods.

7.3.2 k-Clustering

A classical result on k-clustering is the following: Given a finite set X, a distance $d(x, y) \in Z_0^+$ for each pair $x, y \in X$, and two positive integers k and b. The question whether there is a partition of X into k disjoint sets $X_1, \ldots, X_k$ such that $d(x, y) \leq b$ for all pairs $x, y \in X_i$, $1 \leq i \leq k$, is NP-complete [43]. It remains even NP-complete for fixed $k = 3$ and all distances in $\{0, 1\}$. Variants in which the sum of $\max_{x,y \in X_i}\{d(x, y)\}$ or $\sum_{x,y \in X_i} d(x, y)$ over all X_i is bounded by b are also NP-complete [43].

A simple example for polynomial time k-clustering is the task to maximize the minimum inter-cluster distance. From the discussion of the last section it follows that for a given point set S we just have to remove the $k - 1$ longest edges of the MST(S) to obtain an optimal k-clustering. Any other division into k clusters would mean to remove another set of at least $k-1$ edges of the MST(S) and thus cause a shorter minimum inter cluster distance. Removing the edges from the MST(S) can be done in $O(dn)$ time and space and thus by Lemma 7.1 it immediately follows that we can compute a k-clustering of S maximizing the minimum distance between objects of different clusters in $O(dn^2)$ time and $O(dn)$ space.

7.3.3 Divisive Hierarchical Clustering

As we have seen for two-clustering the problem of maximizing the minimum distance between objects of different subclusters of a set S is related to the $MST(S)$. Clearly for this optimization criterion the optimal k-clustering of S and the resulting cluster tree after $k - 1$ steps of a divisive hierarchical clustering are identical, since in both cases we just have to remove the $k - 1$ longest edges of the corresponding MST. We thus get

Theorem 7.11 *Given a set $S \subseteq V^d$ with n elements. A divisive hierarchical cluster tree of S maximizing the minimum distance between objects of different clusters can be computed in $O(dn^2)$ time and $O(dn)$ space.*

Let us now consider the more challenging case of divisive hierarchical clustering minimizing the maximum diameter of the clusters. From Section 7.3.1 we know that one division step of this hierarchical clustering can be done in $O(d^2n^2)$ time if we only "locally" minimize the cluster diameter. If we also want to take into account that the resulting clusters are iteratively divided and that assigning unlabeled vertices or independent sets arbitrarily to cluster 'A' or 'B' affects the optimal division of the next level (or even more levels further down) we immediately get at least a three clustering problem. (Note that in steps 2ai A. and B. of the algorithm given in Figure 7.8 the assignment of the vertices is in fact arbitrary in the sense that we might exchange the labels 'A' and 'B' without effecting the minimization of the maximum diameter.)

From the last section we know that three-clustering to minimize the maximum diameter is already NP-complete and thus the best we can hope for is a heuristic, locally trying to avoid all long distances. E.g. all not yet marked vertices p in step 2aiA might be assigned to the cluster with the smaller maximum distance to p and the labels of a set of step 2aiB might be exchanged in order to minimize the maximum distance between vertices of the set and vertices of the corresponding (partially built) clusters. Clearly all distances we additionally have to check for this heuristic were not considered in the algorithm before, since else the concerned vertices would have to belong to the same set already.

In other words we may continue the steps of the two-clustering algorithm of Figure 7.8, even after determining the diameter of the resulting subclusters, until we have considered all inter-point distances. To this end we just have to skip steps 2ai A to C in Figure 7.8; if required we instead have only to store the largest value of $diss$, which occurs in this section, as the resulting diameter. Thus one dividing step can still be done in $O(d^2n^2)$ time, including the heuristic post processing. This yields a divisive hierarchical cluster algorithm minimizing the maximum diameter with $O(d^2n^3)$ time and $O(dn)$ space. Note however that still the order of assigning the sets and thus the given order of the vertices might influence the result of this heuristic.

If we do not add the heuristic to the division step we can improve the running time by a linear factor: Whenever we perform one division step we do not need to consider all previously considered inter point distances in the sequel. Therefore the generation of all point pairs in order of non-increasing distance needs only to be done once for the whole hierarchical clustering and takes $O(d^2n^2)$ time. Moreover all disjoint set operations can be done in $O(n^2 \log n) = O(dn^2)$ time since we have $O(n^2)$ find and not more than $O(n^2)$ union operations. Together with the discussion of Section 7.3.1 we thus get an overall running time of $O(d^2n^2)$.

Theorem 7.12 *Given a set $S \subseteq V^d$ with n elements. A divisive hierarchical cluster tree of S that (locally) minimizes the maximum diameter of the clusters can be computed in $O(d^2n^2)$ time and $O(dn)$ space.*

7.4 Experimental Results

In this section we give numerical results on the number of necessary vertex comparisons, the running time and the number of rounds for the presented algorithms. We do not consider the quality of the resulting clusters, since this mainly depends on the application in mind and can thus not be made without specific requirements. We are therefore only interested in the efficiency of the implementations.

Additional to all mentioned clustering algorithms we have implemented some more variants to compare the behavior of the different basic approaches. Overall we thus have 14 different clustering algorithms. Table 7.2 gives an overview of the algorithms, their name, the basic method used, the worst case running time, the section they are described and where to find corresponding tables and figures.

Let us first give a brief overview of the examined algorithms. SLINK is the the S-link implementation of the MST-algorithm. CLINK1 is the implementation of C-link using GHC1 and skipping unnecessary steps in the FOR-iteration as mentioned in Section 7.2.2, while CLINK3 uses the simple FOR-loop and goes through all iterations. CLINK2 implements the C-link method using GHC2. MINRADI minimizes the radius of the cluster to its kernel and MINRAD2 is a variant of it based on GHC2. We will use the last four mentioned algorithms to compare the two basic approaches GHC1 and GHC2.

ALINK, ASUMLNK and AMEANLNK are the three average linkage methods A-link, A_{sum}-link and A_{mean}-link of Section 7.2.4. CTRDMEAN minimizes the mean distance of the

algorithm	based on	cf. Section	worst case	see Appendix
SLINK	MST	7.2.1	$O(dn^2)$	Tab. 7.3, 7.17; Fig. 7.11, 7.16
CLINK1	GHC1	7.2.3	$O(d^2n^2)$	Tab. 7.4, 7.18; Fig. 7.12, 7.21
CLINK3	GHC1	7.2.3	$O(d^2n^2)$	Tab. 7.5
MINRADI	GHC1	7.2.5	$O(d^2n^2)$	Tab. 7.6, 7.19
CLINK2	GHC2	7.2.3	$O(dn^3)$	Tab. 7.7, 7.20; Fig. 7.13, 7.17 7.22
ALINK	GHC2	7.2.4	$O(dn^3)$	Tab. 7.8
ASUMLNK	GHC2	7.2.4	$O(dn^3)$	Tab. 7.9
AMEANLNK	GHC2	7.2.4	$O(dn^3)$	Tab. 7.10
MINRAD2	GHC2	7.2.5	$O(dn^3)$	Tab. 7.11, Fig. 7.14
CTRDMEAN	GHC2	7.2.5	$O(dn^3)$	Tab. 7.12, Fig. 7.19
CTRDSUM	GHC2	7.2.5	$O(dn^3)$	Tab. 7.13
KRNLSUM	GHC2	7.2.5	$O(dn^3)$	Tab. 7.14, 7.21; Fig. 7.20, 7.23; 7.25
DIVIDIAM	divisive	7.3.3	$O(d^2n^2)$	Tab. 7.15, 7.22; Fig. 7.15, 7.18, 7.24
DIVIDIA2	divisive	7.3.3	$O(d^2n^3)$	Tab. 7.16

Table 7.2: Implemented and examined clustering algorithms.

vertices to the centroid of a cluster while CTRDSUM minimizes the sum of distance to the centroid and KRNLSUM minimizes the sum of distance to the kernel of a cluster, respectively.

Finally DIVIDIAM is a divisive clustering algorithm minimizing the maximum diameter of the subclusters. While DIVIDIAM stops for one division step after determining the minimum diameter of the subclusters, DIVIDIA2 uses the heuristic mentioned in Section 7.3.3 and has thus to go through all iterations for any divisive step.

In a first phase we have generated 42 test sets with uniformly distributed random vertices. For dimensions $d = 16, 32, 64, 128, 256, 512$ we used set sizes of $n = 100, 200, 500, 1000, 2000, 5000$ and 10000 elements. All 14 implemented algorithms were applied to compute their cluster trees on these sets.

In a second phase we considered special point sets to get additional information. To this end we generated the vertices around a fixed number k of centers, such that the resulting sets are structured sphere-like around each center. Note, however, that the centers need not to belong to the set. We used $k = 10$ centers and required the Hamming distance between two centers to be $\geq \frac{d}{2}$. The points were uniformly distributed around these centers with the limitation that the Hamming distance to the corresponding center must not exceed $\frac{d}{8}$. Thus it is easy to see that no reasonable clustering method (especially non of the implemented algorithms) will combine vertices from different centers, as long as not every center has already "collected" all of its vertices. This forces of course a very special behavior of the algorithms and we will see that in fact the experimental results are sometimes quite different compared to that for random sets.

For these special sets we used the same dimensions and number of vertices as for random sets, except that for $d = 16$ we skipped all sets of size $n \geq 2000$. This was necessary because for $d = 16$ it is not possible to compute a set with more than 1370 vertices fulfilling the above requirements. Moreover we examined only the most interesting algorithms for these sets, since implementations based on the same approach behaved very similarly.

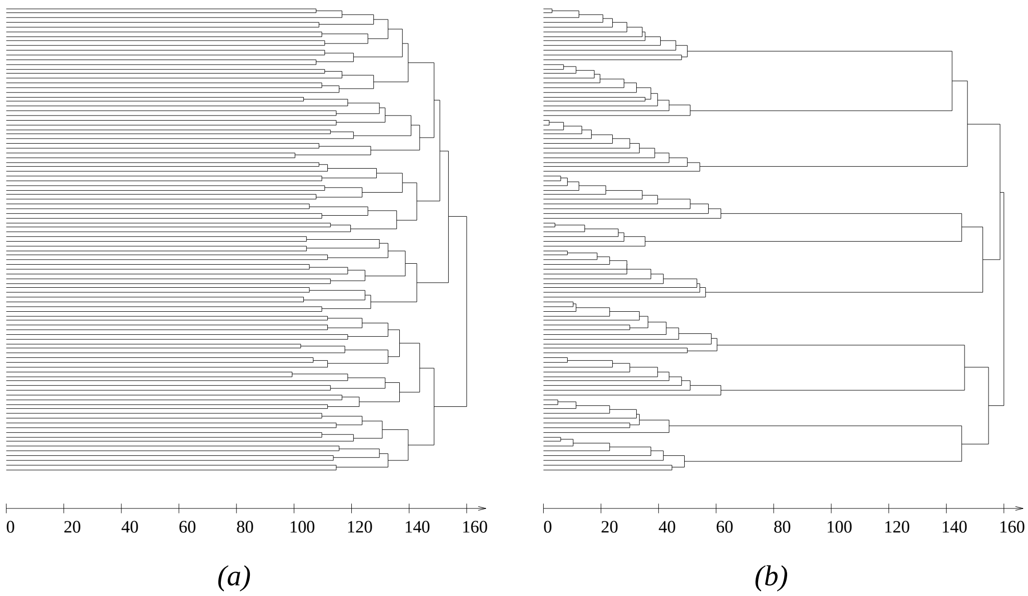


Figure 7.9: Dendrograms for CLINK1 with n=100 objects of dimension d=256; (a) for a random vertex set and (b) for a special vertex set with 10 cluster centers.

Figure 7.9 gives two typical dendrograms (cf. Section 7.2) for CLINK1 with $n = 100$ elements of dimension d=256. On the left side (a) shows a uniformly distributed random set. The difference between the distance of the minpair (approx. 100 bit) and the diameter of the final cluster (approx. 160 bit) is rather small. For the special set given in (b) one directly can see the 10 "cluster balls". Note the significant distinction of the diameter of these subclusters compared to the final cluster.

All experiments were done on customary personal computers (Pentium© processor with 90 MHz) and clearly the running time may differ on other systems. Therefore we also give the number of necessary vertex comparisons, which is the most basic operation of all mentioned methods and thus strongly related to the running time. Note, however, that we distinguish between two different ways of vertex comparisons. The first one is the usual computation of the Hamming distance between two vertices, as e.g. used for S-link or C-link. But for another class of algorithms, which use the idea of storing the number of ones and zeros for every bit position explicitly (e.g. A-link), we have to compare integer vectors of size d, refer to Sections 7.2.4 and 7.2.5 for details.

Although both variants run asymptotically in $\Theta(d)$ time, the first one has a much smaller constant. This reflects the fact that on usual computers the Hamming distance of a 16 or 32 bit number can be computed within one step. Thus the constants of the second class of algorithms is expected to be significantly larger. In fact we can observe the following from the tables. While for algorithms using the binary distance computation we can perform up to about 1.4 10^5 vertex comparisons per second on the mentioned hardware, for ALINK, ASUMLNK, AMEANLNK, CTRDMEAN and CTRDSUM we have a typical number of 1000 to 10000 (with a maximum of 30000 for $d = 16$) vector computations. Clearly the actual number of comparisons per second linearly depends on the size of the vertices (vectors), i.e. the dimension. Thus the running time to compute the entire cluster hierarchy includes an additional factor d, which can be seen from the given curves. See Figures 7.16 to 7.18 and compare them to the number of vertex comparisons, given in Figures 7.11, 7.13 and 7.15, respectively.

Since it is clear from the algorithms which comparisons are involved, we will just speak of number of 'comparisons' in the sequel. But one should have this in mind when comparing the number of comparisons between different algorithms.

All tables and figures of experimental results are given in Appendices 7.6.1 to 7.6.4

The tables containing the running time, the number of necessary comparisons and, for GHC2-based algorithms, the number of iterations, for random vertex sets are given in Appendix 7.6.1.

Appendix 7.6.2 contains a selection of curves showing the number of comparisons, the running time and, for GHC2, the number of iterations for several implementations.

Similarly Appendices 7.6.3 and 7.6.4 contain tables and figures, respectively, for the above mentioned special vertex sets. For these sets we included only the more significant results.

Let us now point out some interesting observations made from the experiments.

- The number of expected iterations for GHC2-based algorithms depends logarithmically on the number of vertices, see e.g. Figures 7.19, 7.20 and 7.25. Thus the running time of the algorithms was observed to be $O(dn^2 \log n)$ rather than $O(dn^3)$. For random sets the dependency of the number of iterations on the dimension d is rather weak for all algorithms, cf. the typical curves in Fig. 7.20, except for AMEANLNK and CMEANLNK (see Fig. 7.19).

- For GHC1-based algorithms it is, as expected, more efficient to skip unnecessary iterations in the FOR-loop (cf. Section 7.2.2), as can be seen from the Tables for CLINK1 (Tab. 7.4) and CLINK3 (Tab. 7.5).

- Considering the dependency of the number of vertex comparisons from the dimension d for random sets the algorithms behave quite differently. For example SLINK is entirely independent from d (c.f. Fig. 7.11), while e.g. for CLINK1 (c.f. Fig. 7.12) and DIVIDIAM (c.f. Fig. 7.15) there is a very distinctive relation. This is typical for all implementations based on GHC1 and for both divisive algorithms. In contrast the dependency for GHC2 based algorithms is rather weak, see e.g. Figures7.13 and Figures7.14.

- For special point sets most implemented algorithms used approximately the same or more time to compute the cluster hierarchy than for the corresponding random sets. Especially algorithms based on GHC2 were running up to 10 times longer (e.g. MINRAD2), while SLINK, algorithms based on GHC1 and divisive algorithms did not show too much alteration. Sometimes they used even less comparisons than for random sets (e.g. DIVIDIAM, see Table 7.22). Compared to random sets GHC1-based algorithms are typically faster for small dimensions and large sets, e.g. CLINK1 or MINRADI for d=32,64 and n=10000 (Tables 7.18 and 7.19). For high dimensions (d=256 and more) special point sets may need significantly more comparisons (Tab. 7.19).

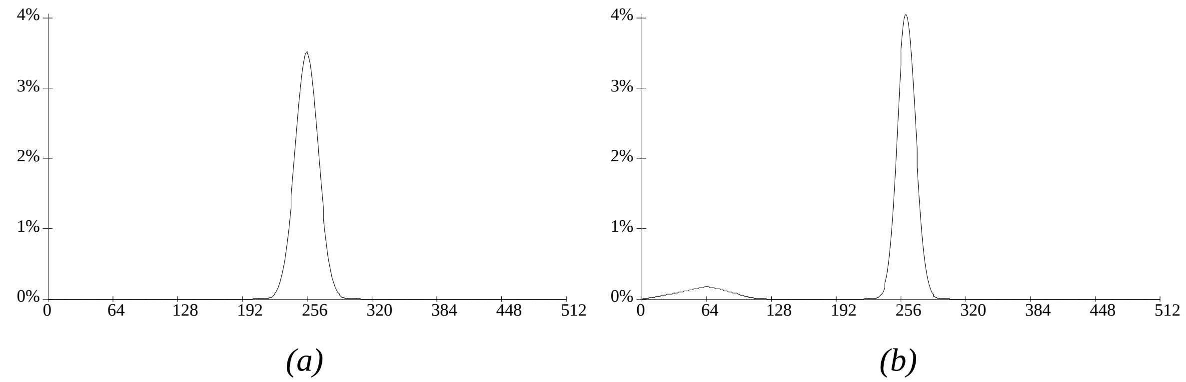


Figure 7.10: Percentage of pairwise Hamming distances for (a) uniformly distributed vertices and (b) a specially structured set (see text) with $n = 10000$ vertices and dimension $d = 512$.

So what caused the break down of the GHC2 method? Considering the Hamming distance between vertices of a random point set it is easy to show that with high expectation all these distances are arranged around $\frac{d}{2}$. A typical range for $d = 512$ and $n = 10000$ is about $\{200, \ldots, 300\}$, see Figure 7.10(a). Since these distances are "not too small" there typically is a large number of BMC-pairs which can be agglomerated in parallel. This may be the main reason why the number of iterations for GHC2-based algorithms is much less than the worst case $n - 1$.

Unfortunately the specially structured sets now force the vertices to build smaller "balls" and thus within these balls a lot of small distances appear. In our examples with 10 cluster centers, for a minimum Hamming distance of $\frac{d}{2}$ between two centers, and for a maximum radius of $\frac{d}{8}$, these intra-ball distances are in the range of $\{0, \ldots, \frac{d}{4}\}$. In addition the inter-ball distances can be in the range $\{\frac{d}{4}, \ldots, d\}$, i.e. are still around $\frac{d}{2}$ (cf. Fig. 7.10(b)). Within one ball the dependency of the BMC-relations is now much higher due to the small and thus often multiple distances. Thus during one iteration of GHC2 only a small number of pairs can be merged. Furthermore, as long as not all pairs within a ball are merged, no BMC outside the ball can be used. Experiments showed that sometimes only one or two agglomeration steps per ball were possible. This increases the number of necessary iterations and thus the running time.

- DIVIDIAM is the only algorithm which runs faster, and has a smaller number of vertex comparisons, for all tested special sets compared to the corresponding random sets. One intuitive explanation is that during the initial iterations DIVIDIAM determines clusters related to the 10 balls around the given centers. Thus for the special sets the advantage of divisive methods, that after the first few iterations only distances within subclusters have to be considered, fully takes effect. Since for our special sets such smaller subclusters are more homogeneous and can be built faster than for random sets, the overall number of necessary vertex comparisons, and thus the running time is smaller.

- For special point sets the dependency for GHC2 based algorithms from the dimension d is stronger. Compare for example the number of comparisons for CLINK2 (Fig. 7.13 vs. Fig. 7.22) or the number of iterations for KRNLSUM (Fig. 7.20 vs. Fig. 7.25). For

DIVIDIAM the situation is contrary, see Fig. 7.15 vs. Fig. 7.24. This again reveals the fact that divisive methods benefit from the special structure of these sets, as was already pointed out in the last paragraph.

- All algorithms based on GHC1 may alternatively be implemented using GHC2. Comparing the tables for CLINK1 and CLINK2 or MINRADI and MINRAD2, respectively, we observe that for random point sets they provide faster running times if the dimension is high enough (e.g. d=256 or 512). In this case they should be considered as a true alternative solution, although their worst case running time is larger.

  But note that for special point sets this is not necessary the case. Here CLINK2 is only faster than CLINK1 if the number of vertices is small enough, for example $d = 512$ and $n = 200$. The same is true when comparing MEANRADI and MEANRAD2. Thus for specially structured sets GHC1-based algorithms may be the better choice, as was already pointed out above.

- What are the fasted algorithms? As to be expected from the worst case running time, S-link is by far the fastest method. The second best implementation was the divisive method minimizing the diameter of the subclusters, DIVIDIAM. But also all other algorithms run efficiently for appropriately sized inputs. GHC2-based algorithms run fast for random vertex files, like KRNLSUM which was among the best. Especially for large vertex sets these algorithms may beat GHC1-based implementations. But for specially structured sets they may break down and should not be used.

7.5 Open Problems

An obvious question concerns the metric to compute the distance of two objects. The measure we used, the Hamming distance, is just the L_1-metric of $\mathcal{R}^d$. For objects from V^d clearly L_∞ makes no sense and for $2 \leq p < \infty$ we get $L_p = (L_1)^{\frac{1}{p}}$, i.e. the dependence between L_1 and L_p is monotone. Since most presented methods (except those using sum or mean of distances) only rely on the relative distance order of pairs of objects, a different choice of p would not change the resulting clusters, except the value of their weight.

Some open questions mentioned in previous sections concern the time complexity of the following problems (as functions of n and d):

- Centroidal clustering, for minimizing the radius to a centroid point in V^d (cf. Section 7.2.5)
  INSTANCE: A vertex set $S \subseteq V^d$, $|S| = n$.
  QUESTION: Compute the vertex $p \in V^d$ which minimizes the maximum distance of a point $q \in S$ to p.

- Two-centroid clustering, for minimizing the maximum sum of distances in subclusters (cf. Section 7.3.1)
  INSTANCE: A vertex set $S \subseteq V^d$, $|S| = n$.
  QUESTION: Compute two centroids $c_1, c_2 \in V^d$ and a division of S into S_1 and S_2, such that $\max \left\{ \sum_{p_i \in S_1} d_H(p_i, c_1), \sum_{p_j \in S_2} d_H(p_j, c_2) \right\}$ is minimized.

Related to this problem is the following question, which is also of interest in its own.

- Two-centroid clustering, for minimizing the maximum sum of distances to two given centroids (see Section 7.3.1)
  INSTANCE: A vertex set $S \subseteq V^d$, $|S| = n$, and two vertices $c_1, c_2 \in V^d$.
  QUESTION: Split S into S_1 and S_2, such that $\max\left\{\sum_{p_i \in S_1} d_H(p_i, c_1), \sum_{p_j \in S_2} d_H(p_j, c_2)\right\}$ is minimized.

A variant of this problem asks to minimize the mean distance to the centroids, i.e. to minimize the term $\max\left\{\frac{1}{|S_1|}\sum_{p_i \in S_1} d_H(p_i, c_1), \frac{1}{|S_2|}\sum_{p_j \in S_2} d_H(p_j, c_2)\right\}$.

For alternative divisive hierarchical clustering methods the following question is of interest.

- Are there more efficient methods for the two-kernel problems mentioned in Section 7.3.1?

7.6 Appendix

7.6.1 Tables for Uniformly Distributed Random Vertex Sets

Section 7.6.1 summarizes the results of our experimental section for uniformly distributed random vertex sets in 14 tables. A discussion of the results is given in Section 7.4.

Tables 7.3 to 7.16 show the running time of the algorithms in seconds, and the number of comparisons, respectively, in dependency of the dimension d and the number of elements n. Tables 7.7 to 7.14 show the results for methods based on Algorithm GHC2 and give in addition the number of iterations of the WHILE loop (cf. Figure 7.4). Finally Tables 7.15 and 7.16 give the results for divisive methods.

$n \setminus d$	16	32	64
100	$5.00\ 10^{-2}$ / $4.95\ 10^{3}$	$5.00\ 10^{-2}$ / $4.95\ 10^{3}$	$5.00\ 10^{-2}$ / $4.95\ 10^{3}$
200	$1.70\ 10^{-1}$ / $1.99\ 10^{4}$	$1.10\ 10^{-1}$ / $1.99\ 10^{4}$	$1.60\ 10^{-1}$ / $1.99\ 10^{4}$
500	$1.05\ 10^{0}$ / $1.24\ 10^{5}$	$1.10\ 10^{0}$ / $1.24\ 10^{5}$	$1.15\ 10^{0}$ / $1.24\ 10^{5}$
1000	$4.62\ 10^{0}$ / $4.99\ 10^{5}$	$4.50\ 10^{0}$ / $4.99\ 10^{5}$	$5.11\ 10^{0}$ / $4.99\ 10^{5}$
2000	$1.81\ 10^{1}$ / $1.99\ 10^{6}$	$1.89\ 10^{1}$ / $1.99\ 10^{6}$	$2.06\ 10^{1}$ / $1.99\ 10^{6}$
5000	$1.23\ 10^{2}$ / $1.25\ 10^{7}$	$1.32\ 10^{2}$ / $1.25\ 10^{7}$	$1.43\ 10^{2}$ / $1.25\ 10^{7}$
10000	$5.34\ 10^{2}$ / $5.00\ 10^{7}$	$5.64\ 10^{2}$ / $5.00\ 10^{7}$	$6.07\ 10^{2}$ / $5.00\ 10^{7}$
$n \setminus d$	128	256	512
100	$5.00\ 10^{-2}$ / $4.95\ 10^{3}$	$6.00\ 10^{-2}$ / $4.95\ 10^{3}$	$1.10\ 10^{-1}$ / $4.95\ 10^{3}$
200	$2.70\ 10^{-1}$ / $1.99\ 10^{4}$	$2.20\ 10^{-1}$ / $1.99\ 10^{4}$	$3.90\ 10^{-1}$ / $1.99\ 10^{4}$
500	$1.59\ 10^{0}$ / $1.24\ 10^{5}$	$1.43\ 10^{0}$ / $1.24\ 10^{5}$	$2.25\ 10^{0}$ / $1.24\ 10^{5}$
1000	$6.21\ 10^{0}$ / $4.99\ 10^{5}$	$5.49\ 10^{0}$ / $4.99\ 10^{5}$	$9.01\ 10^{0}$ / $4.99\ 10^{5}$
2000	$2.41\ 10^{1}$ / $1.99\ 10^{6}$	$2.20\ 10^{1}$ / $1.99\ 10^{6}$	$3.58\ 10^{1}$ / $1.99\ 10^{6}$
5000	$1.64\ 10^{2}$ / $1.25\ 10^{7}$	$1.44\ 10^{2}$ / $1.25\ 10^{7}$	$2.30\ 10^{2}$ / $1.25\ 10^{7}$
10000	$6.93\ 10^{2}$ / $5.00\ 10^{7}$	$6.00\ 10^{2}$ / $5.00\ 10^{7}$	$9.42\ 10^{2}$ / $5.00\ 10^{7}$

Table 7.3: Running time in seconds and No. of comparisons for SLINK.

$n \setminus d$	16	32	64
100	$5.00\ 10^{-1}$ / $6.22\ 10^{4}$	$8.20\ 10^{-1}$ / $9.29\ 10^{4}$	$1.16\ 10^{0}$ / $1.18\ 10^{5}$
200	$2.25\ 10^{0}$ / $2.58\ 10^{5}$	$3.40\ 10^{0}$ / $3.90\ 10^{5}$	$5.50\ 10^{0}$ / $5.62\ 10^{5}$
500	$1.42\ 10^{1}$ / $1.68\ 10^{6}$	$2.66\ 10^{1}$ / $2.93\ 10^{6}$	$3.79\ 10^{1}$ / $3.80\ 10^{6}$
1000	$5.80\ 10^{1}$ / $6.79\ 10^{6}$	$1.11\ 10^{2}$ / $1.20\ 10^{7}$	$1.76\ 10^{2}$ / $1.73\ 10^{7}$
2000	$2.32\ 10^{2}$ / $2.74\ 10^{7}$	$4.48\ 10^{2}$ / $4.87\ 10^{7}$	$7.57\ 10^{2}$ / $7.44\ 10^{7}$
5000	$1.54\ 10^{3}$ / $1.75\ 10^{8}$	$3.18\ 10^{3}$ / $3.30\ 10^{8}$	$5.08\ 10^{3}$ / $4.79\ 10^{8}$
10000	$6.41\ 10^{3}$ / $7.08\ 10^{8}$	$1.34\ 10^{4}$ / $1.33\ 10^{9}$	$2.33\ 10^{4}$ / $2.09\ 10^{9}$
$n \setminus d$	128	256	512
100	$1.10\ 10^{0}$ / $1.58\ 10^{5}$	$2.25\ 10^{0}$ / $2.11\ 10^{5}$	$4.89\ 10^{0}$ / $2.81\ 10^{5}$
200	$5.33\ 10^{0}$ / $7.39\ 10^{5}$	$1.12\ 10^{1}$ / $1.05\ 10^{6}$	$2.39\ 10^{1}$ / $1.37\ 10^{6}$
500	$3.98\ 10^{1}$ / $5.47\ 10^{6}$	$8.03\ 10^{1}$ / $7.48\ 10^{6}$	$1.85\ 10^{2}$ / $1.06\ 10^{7}$
1000	$1.70\ 10^{2}$ / $2.31\ 10^{7}$	$3.74\ 10^{2}$ / $3.45\ 10^{7}$	$8.13\ 10^{2}$ / $4.62\ 10^{7}$
2000	$7.54\ 10^{2}$ / $1.01\ 10^{8}$	$1.58\ 10^{3}$ / $1.45\ 10^{8}$	$3.57\ 10^{3}$ / $2.02\ 10^{8}$
5000	$5.04\ 10^{3}$ / $6.71\ 10^{8}$	$1.06\ 10^{4}$ / $9.69\ 10^{8}$	$2.48\ 10^{4}$ / $1.39\ 10^{9}$
10000	$2.21\ 10^{4}$ / $2.85\ 10^{9}$	$4.49\ 10^{4}$ / $4.07\ 10^{9}$	$1.04\ 10^{5}$ / $5.84\ 10^{9}$

Table 7.4: Running time in seconds and No. of comparisons for CLINK1.

$n \setminus d$	16	32	64
100	$5.00\ 10^{-1}$ / $6.22\ 10^{4}$	$1.05\ 10^{0}$ / $1.12\ 10^{5}$	$2.14\ 10^{0}$ / $2.05\ 10^{5}$
200	$1.98\ 10^{0}$ / $2.58\ 10^{5}$	$4.06\ 10^{0}$ / $4.79\ 10^{5}$	$8.29\ 10^{0}$ / $8.61\ 10^{5}$
500	$1.31\ 10^{1}$ / $1.68\ 10^{6}$	$2.67\ 10^{1}$ / $3.05\ 10^{6}$	$5.56\ 10^{1}$ / $5.54\ 10^{6}$
1000	$5.73\ 10^{1}$ / $6.79\ 10^{6}$	$1.15\ 10^{2}$ / $1.25\ 10^{7}$	$2.39\ 10^{2}$ / $2.30\ 10^{7}$
2000	$2.28\ 10^{2}$ / $2.74\ 10^{7}$	$4.64\ 10^{2}$ / $5.07\ 10^{7}$	$9.78\ 10^{2}$ / $9.44\ 10^{7}$
5000	$1.65\ 10^{3}$ / $1.75\ 10^{8}$	$3.44\ 10^{3}$ / $3.30\ 10^{8}$	$7.02\ 10^{3}$ / $6.04\ 10^{8}$
10000	$6.23\ 10^{3}$ / $7.08\ 10^{8}$	$1.29\ 10^{4}$ / $1.33\ 10^{9}$	$2.69\ 10^{4}$ / $2.44\ 10^{9}$
$n \setminus d$	128	256	512
100	$2.86\ 10^{0}$ / $3.84\ 10^{5}$	$8.08\ 10^{0}$ / $7.44\ 10^{5}$	$2.53\ 10^{1}$ / $1.42\ 10^{6}$
200	$1.85\ 10^{1}$ / $1.58\ 10^{6}$	$3.38\ 10^{1}$ / $3.03\ 10^{6}$	$1.04\ 10^{2}$ / $5.78\ 10^{6}$
500	$1.25\ 10^{2}$ / $1.04\ 10^{7}$	$2.17\ 10^{2}$ / $1.93\ 10^{7}$	$6.72\ 10^{2}$ / $3.70\ 10^{7}$
1000	$3.23\ 10^{2}$ / $4.21\ 10^{7}$	$8.88\ 10^{2}$ / $7.91\ 10^{7}$	$2.71\ 10^{3}$ / $1.49\ 10^{8}$
2000	$2.08\ 10^{3}$ / $1.70\ 10^{8}$	$3.65\ 10^{3}$ / $3.24\ 10^{8}$	$1.09\ 10^{4}$ / $6.03\ 10^{8}$
5000	$1.50\ 10^{4}$ / $1.10\ 10^{9}$	$2.33\ 10^{4}$ / $2.04\ 10^{9}$	$7.03\ 10^{4}$ / $3.83\ 10^{9}$
10000	$3.49\ 10^{4}$ / $4.43\ 10^{9}$	$9.47\ 10^{4}$ / $8.25\ 10^{9}$	$2.88\ 10^{5}$ / $1.55\ 10^{10}$

Table 7.5: Running time in seconds and No. of comparisons for CLINK3.

$n \setminus d$	16	32	64
100	$5.50\ 10^{-1}$ / $5.60\ 10^{4}$	$8.20\ 10^{-1}$ / $8.66\ 10^{4}$	$1.10\ 10^{0}$ / $1.04\ 10^{5}$
200	$2.30\ 10^{0}$ / $2.57\ 10^{5}$	$3.62\ 10^{0}$ / $3.72\ 10^{5}$	$5.27\ 10^{0}$ / $4.90\ 10^{5}$
500	$1.60\ 10^{1}$ / $1.77\ 10^{6}$	$2.85\ 10^{1}$ / $2.84\ 10^{6}$	$3.67\ 10^{1}$ / $3.41\ 10^{6}$
1000	$6.65\ 10^{1}$ / $7.32\ 10^{6}$	$1.21\ 10^{2}$ / $1.21\ 10^{7}$	$1.81\ 10^{2}$ / $1.66\ 10^{7}$
2000	$2.77\ 10^{2}$ / $3.04\ 10^{7}$	$5.04\ 10^{2}$ / $5.05\ 10^{7}$	$7.57\ 10^{2}$ / $6.91\ 10^{7}$
5000	$2.29\ 10^{3}$ / $2.05\ 10^{8}$	$4.19\ 10^{3}$ / $3.45\ 10^{8}$	$6.14\ 10^{3}$ / $4.69\ 10^{8}$
10000	$9.84\ 10^{3}$ / $8.57\ 10^{8}$	$1.78\ 10^{4}$ / $1.42\ 10^{9}$	$2.78\ 10^{4}$ / $2.05\ 10^{9}$
$n \setminus d$	128	256	512
100	$1.76\ 10^{0}$ / $1.42\ 10^{5}$	$2.25\ 10^{0}$ / $1.95\ 10^{5}$	$3.85\ 10^{0}$ / $2.08\ 10^{5}$
200	$8.02\ 10^{0}$ / $6.44\ 10^{5}$	$9.99\ 10^{0}$ / $8.66\ 10^{5}$	$1.94\ 10^{1}$ / $1.06\ 10^{6}$
500	$5.98\ 10^{1}$ / $4.80\ 10^{6}$	$7.42\ 10^{1}$ / $6.47\ 10^{6}$	$1.52\ 10^{2}$ / $8.34\ 10^{6}$
1000	$2.57\ 10^{2}$ / $2.05\ 10^{7}$	$3.46\ 10^{2}$ / $2.98\ 10^{7}$	$6.92\ 10^{2}$ / $3.78\ 10^{7}$
2000	$1.19\ 10^{3}$ / $9.44\ 10^{7}$	$1.41\ 10^{3}$ / $1.22\ 10^{8}$	$3.15\ 10^{3}$ / $1.71\ 10^{8}$
5000	$9.09\ 10^{3}$ / $6.19\ 10^{8}$	$1.00\ 10^{4}$ / $8.60\ 10^{8}$	$2.23\ 10^{4}$ / $1.20\ 10^{9}$
10000	$4.18\ 10^{4}$ / $2.73\ 10^{9}$	$4.36\ 10^{4}$ / $3.66\ 10^{9}$	$9.64\ 10^{4}$ / $5.14\ 10^{9}$

Table 7.6: Running time in seconds and No. of comparisons for MINRADI.

$n \setminus d$	16	32	64
100	$1.04\ 10^{0}$ / $1.15\ 10^{5}$ / 17	$1.10\ 10^{0}$ / $1.19\ 10^{5}$ / 18	$1.21\ 10^{0}$ / $1.11\ 10^{5}$ / 16
200	$5.49\ 10^{0}$ / $5.74\ 10^{5}$ / 22	$5.55\ 10^{0}$ / $5.89\ 10^{5}$ / 23	$6.26\ 10^{0}$ / $6.19\ 10^{5}$ / 24
500	$4.28\ 10^{1}$ / $4.99\ 10^{6}$ / 33	$4.30\ 10^{1}$ / $4.65\ 10^{6}$ / 32	$4.70\ 10^{1}$ / $4.66\ 10^{6}$ / 29
1000	$1.78\ 10^{2}$ / $1.99\ 10^{7}$ / 32	$2.08\ 10^{2}$ / $2.20\ 10^{7}$ / 37	$2.28\ 10^{2}$ / $2.21\ 10^{7}$ / 37
2000	$9.98\ 10^{2}$ / $9.77\ 10^{7}$ / 40	$1.07\ 10^{3}$ / $1.01\ 10^{8}$ / 42	$1.17\ 10^{3}$ / $1.01\ 10^{8}$ / 41
5000	$7.60\ 10^{3}$ / $7.16\ 10^{8}$ / 51	$7.94\ 10^{3}$ / $7.15\ 10^{8}$ / 49	$9.32\ 10^{3}$ / $7.74\ 10^{8}$ / 52
10000	$3.16\ 10^{4}$ / $3.26\ 10^{9}$ / 62	$3.78\ 10^{4}$ / $3.33\ 10^{9}$ / 56	$4.37\ 10^{4}$ / $3.55\ 10^{9}$ / 62
$n \setminus d$	128	256	512
100	$1.54\ 10^{0}$ / $1.26\ 10^{5}$ / 20	$1.26\ 10^{0}$ / $1.12\ 10^{5}$ / 17	$2.20\ 10^{0}$ / $1.21\ 10^{5}$ / 21
200	$4.72\ 10^{0}$ / $6.13\ 10^{5}$ / 23	$6.98\ 10^{0}$ / $6.16\ 10^{5}$ / 24	$1.11\ 10^{1}$ / $6.08\ 10^{5}$ / 24
500	$3.56\ 10^{1}$ / $4.70\ 10^{6}$ / 29	$5.82\ 10^{1}$ / $5.20\ 10^{6}$ / 32	$9.05\ 10^{1}$ / $4.92\ 10^{6}$ / 32
1000	$2.75\ 10^{2}$ / $2.27\ 10^{7}$ / 35	$2.83\ 10^{2}$ / $2.52\ 10^{7}$ / 41	$4.45\ 10^{2}$ / $2.41\ 10^{7}$ / 42
2000	$8.27\ 10^{2}$ / $1.08\ 10^{8}$ / 44	$1.25\ 10^{3}$ / $1.11\ 10^{8}$ / 47	$2.13\ 10^{3}$ / $1.15\ 10^{8}$ / 51
5000	$6.17\ 10^{3}$ / $7.83\ 10^{8}$ / 53	$9.48\ 10^{3}$ / $8.23\ 10^{8}$ / 54	$1.62\ 10^{4}$ / $8.66\ 10^{8}$ / 62
10000	$3.04\ 10^{4}$ / $3.74\ 10^{9}$ / 66	$4.53\ 10^{4}$ / $3.85\ 10^{9}$ / 67	$7.77\ 10^{4}$ / $4.10\ 10^{9}$ / 79

Table 7.7: Running time in seconds, No. of comparisons and No. of iterations for CLINK2.

$n \setminus d$	16	32	64
100	7.70 10^{-1} / 2.17 10^{4} / 17	1.37 10^{0} / 2.13 10^{4} / 18	2.91 10^{0} / 2.26 10^{4} / 16
200	3.07 10^{0} / 8.96 10^{4} / 20	5.71 10^{0} / 8.87 10^{4} / 19	1.10 10^{1} / 8.76 10^{4} / 20
500	1.93 10^{1} / 5.61 10^{5} / 27	3.67 10^{1} / 5.65 10^{5} / 23	7.12 10^{1} / 5.65 10^{5} / 26
1000	7.86 10^{1} / 2.23 10^{6} / 29	1.49 10^{2} / 2.25 10^{6} / 33	2.92 10^{2} / 2.27 10^{6} / 28
2000	3.19 10^{2} / 9.21 10^{6} / 35	6.06 10^{2} / 9.29 10^{6} / 37	1.19 10^{3} / 9.47 10^{6} / 36
5000	2.05 10^{3} / 5.89 10^{7} / 39	3.86 10^{3} / 5.88 10^{7} / 45	7.73 10^{3} / 6.09 10^{7} / 42
10000	8.85 10^{3} / 2.52 10^{8} / 47	1.57 10^{4} / 2.40 10^{8} / 47	3.18 10^{4} / 2.50 10^{8} / 52
$n \setminus d$	128	256	512
100	5.60 10^{0} / 2.22 10^{4} / 16	1.07 10^{1} / 2.15 10^{4} / 16	2.10 10^{1} / 2.12 10^{4} / 17
200	2.20 10^{1} / 8.88 10^{4} / 20	4.51 10^{1} / 9.17 10^{4} / 20	8.51 10^{1} / 8.69 10^{4} / 16
500	1.41 10^{2} / 5.69 10^{5} / 25	2.89 10^{2} / 5.86 10^{5} / 26	5.78 10^{2} / 5.88 10^{5} / 24
1000	5.91 10^{2} / 2.35 10^{6} / 29	1.22 10^{3} / 2.46 10^{6} / 32	2.48 10^{3} / 2.51 10^{6} / 36
2000	2.39 10^{3} / 9.61 10^{6} / 38	4.92 10^{3} / 9.96 10^{6} / 35	9.88 10^{3} / 1.00 10^{7} / 40
5000	1.57 10^{4} / 6.31 10^{7} / 47	3.20 10^{4} / 6.49 10^{7} / 46	6.51 10^{4} / 6.61 10^{7} / 47
10000	6.51 10^{4} / 2.61 10^{8} / 55	1.32 10^{5} / 2.67 10^{8} / 54	2.73 10^{5} / 2.75 10^{8} / 57

Table 7.8: Running time in seconds, No. of comparisons and No. of iterations for ALINK.

$n \setminus d$	16	32	64
100	6.00 10^{-1} / 3.19 10^{4} / 28	1.15 10^{0} / 3.06 10^{4} / 31	2.31 10^{0} / 3.14 10^{4} / 31
200	2.81 10^{0} / 1.39 10^{5} / 37	5.06 10^{0} / 1.33 10^{5} / 39	9.66 10^{0} / 1.31 10^{5} / 37
500	1.77 10^{1} / 8.67 10^{5} / 58	3.47 10^{1} / 9.07 10^{5} / 57	6.52 10^{1} / 8.85 10^{5} / 59
1000	8.07 10^{1} / 3.81 10^{6} / 84	1.48 10^{2} / 3.79 10^{6} / 74	2.77 10^{2} / 3.68 10^{6} / 80
2000	3.31 10^{2} / 1.59 10^{7} / 92	6.13 10^{2} / 1.59 10^{7} / 97	1.18 10^{3} / 1.60 10^{7} / 99
5000	2.25 10^{3} / 1.06 10^{8} / 129	4.34 10^{3} / 1.11 10^{8} / 132	8.22 10^{3} / 1.10 10^{8} / 126
10000	9.20 10^{3} / 4.37 10^{8} / 159	1.81 10^{4} / 4.65 10^{8} / 162	3.35 10^{4} / 4.47 10^{8} / 155
$n \setminus d$	128	256	512
100	4.67 10^{0} / 3.21 10^{4} / 32	8.63 10^{0} / 3.01 10^{4} / 29	1.67 10^{1} / 2.91 10^{4} / 28
200	1.86 10^{1} / 1.30 10^{5} / 39	3.85 10^{1} / 1.35 10^{5} / 40	7.41 10^{1} / 1.30 10^{5} / 37
500	1.32 10^{2} / 9.20 10^{5} / 61	2.54 10^{2} / 8.86 10^{5} / 62	5.29 10^{2} / 9.27 10^{5} / 60
1000	5.74 10^{2} / 3.89 10^{6} / 77	1.13 10^{3} / 3.90 10^{6} / 81	2.21 10^{3} / 3.82 10^{6} / 81
2000	2.30 10^{3} / 1.58 10^{7} / 101	4.60 10^{3} / 1.60 10^{7} / 103	9.30 10^{3} / 1.62 10^{7} / 101
5000	1.57 10^{4} / 1.07 10^{8} / 138	3.13 10^{4} / 1.08 10^{8} / 136	6.08 10^{4} / 1.06 10^{8} / 130
10000	6.66 10^{4} / 4.53 10^{8} / 158	1.32 10^{5} / 4.54 10^{8} / 170	2.58 10^{5} / 4.47 10^{8} / 169

Table 7.9: Running time in seconds, No. of comparisons and No. of iterations for ASUMLNK.

$n \setminus d$	16	32	64
100	$4.90\ 10^{-1}$ / $2.33\ 10^{4}$ / 17	$8.20\ 10^{-1}$ / $2.22\ 10^{4}$ / 16	$1.65\ 10^{0}$ / $2.28\ 10^{4}$ / 20
200	$1.97\ 10^{0}$ / $9.25\ 10^{4}$ / 23	$3.57\ 10^{0}$ / $9.21\ 10^{4}$ / 23	$7.03\ 10^{0}$ / $9.40\ 10^{4}$ / 27
500	$1.29\ 10^{1}$ / $5.93\ 10^{5}$ / 32	$2.33\ 10^{1}$ / $5.91\ 10^{5}$ / 33	$4.39\ 10^{1}$ / $5.85\ 10^{5}$ / 34
1000	$5.30\ 10^{1}$ / $2.42\ 10^{6}$ / 42	$9.32\ 10^{1}$ / $2.34\ 10^{6}$ / 41	$1.78\ 10^{2}$ / $2.36\ 10^{6}$ / 49
2000	$2.21\ 10^{2}$ / $1.00\ 10^{7}$ / 49	$3.87\ 10^{2}$ / $9.71\ 10^{6}$ / 51	$7.34\ 10^{2}$ / $9.73\ 10^{6}$ / 57
5000	$1.41\ 10^{3}$ / $6.34\ 10^{7}$ / 66	$2.47\ 10^{3}$ / $6.16\ 10^{7}$ / 62	$4.75\ 10^{3}$ / $6.26\ 10^{7}$ / 69
10000	$5.78\ 10^{3}$ / $2.58\ 10^{8}$ / 67	$1.00\ 10^{4}$ / $2.49\ 10^{8}$ / 73	$1.94\ 10^{4}$ / $2.55\ 10^{8}$ / 80
$n \setminus d$	128	256	512
100	$3.29\ 10^{0}$ / $2.26\ 10^{4}$ / 21	$6.32\ 10^{0}$ / $2.21\ 10^{4}$ / 21	$1.21\ 10^{1}$ / $2.14\ 10^{4}$ / 22
200	$1.27\ 10^{1}$ / $8.80\ 10^{4}$ / 24	$2.63\ 10^{1}$ / $9.21\ 10^{4}$ / 28	$4.88\ 10^{1}$ / $8.58\ 10^{4}$ / 25
500	$8.37\ 10^{1}$ / $5.73\ 10^{5}$ / 35	$1.72\ 10^{2}$ / $5.99\ 10^{5}$ / 37	$3.51\ 10^{2}$ / $6.12\ 10^{5}$ / 40
1000	$3.47\ 10^{2}$ / $2.37\ 10^{6}$ / 51	$7.21\ 10^{2}$ / $2.49\ 10^{6}$ / 49	$1.44\ 10^{3}$ / $2.51\ 10^{6}$ / 49
2000	$1.44\ 10^{3}$ / $9.86\ 10^{6}$ / 60	$2.93\ 10^{3}$ / $1.01\ 10^{7}$ / 66	$5.92\ 10^{3}$ / $1.03\ 10^{7}$ / 72
5000	$9.38\ 10^{3}$ / $6.37\ 10^{7}$ / 82	$1.92\ 10^{4}$ / $6.62\ 10^{7}$ / 91	$4.09\ 10^{4}$ / $6.77\ 10^{7}$ / 93
10000	$3.89\ 10^{4}$ / $2.62\ 10^{8}$ / 97	$7.98\ 10^{4}$ / $2.73\ 10^{8}$ / 111	$1.63\ 10^{5}$ / $2.82\ 10^{8}$ / 119

Table 7.10: Running time in seconds, No. of comparisons and No. of iterations for AMEANLNK.

$n \setminus d$	16	32	64
100	$1.10\ 10^{0}$ / $1.16\ 10^{5}$ / 17	$6.60\ 10^{-1}$ / $1.26\ 10^{5}$ / 18	$7.70\ 10^{-1}$ / $1.21\ 10^{5}$ / 17
200	$5.65\ 10^{0}$ / $6.39\ 10^{5}$ / 24	$3.08\ 10^{0}$ / $6.02\ 10^{5}$ / 21	$3.52\ 10^{0}$ / $5.83\ 10^{5}$ / 21
500	$4.44\ 10^{1}$ / $4.94\ 10^{6}$ / 32	$2.31\ 10^{1}$ / $4.61\ 10^{6}$ / 27	$2.85\ 10^{1}$ / $4.87\ 10^{6}$ / 29
1000	$1.99\ 10^{2}$ / $2.14\ 10^{7}$ / 32	$1.12\ 10^{2}$ / $2.18\ 10^{7}$ / 35	$1.38\ 10^{2}$ / $2.30\ 10^{7}$ / 35
2000	$1.14\ 10^{3}$ / $1.00\ 10^{8}$ / 40	$5.17\ 10^{2}$ / $1.03\ 10^{8}$ / 43	$6.50\ 10^{2}$ / $1.04\ 10^{8}$ / 37
5000	$9.17\ 10^{3}$ / $7.66\ 10^{8}$ / 64	$9.57\ 10^{3}$ / $7.72\ 10^{8}$ / 51	$4.97\ 10^{3}$ / $7.80\ 10^{8}$ / 50
10000	$2.81\ 10^{4}$ / $3.44\ 10^{9}$ / 67	$2.92\ 10^{4}$ / $3.40\ 10^{9}$ / 57	$2.20\ 10^{4}$ / $3.59\ 10^{9}$ / 54
$n \setminus d$	128	256	512
100	$8.80\ 10^{-1}$ / $1.14\ 10^{5}$ / 16	$1.54\ 10^{0}$ / $1.39\ 10^{5}$ / 19	$2.25\ 10^{0}$ / $1.25\ 10^{5}$ / 17
200	$4.50\ 10^{0}$ / $5.87\ 10^{5}$ / 22	$6.70\ 10^{0}$ / $6.02\ 10^{5}$ / 20	$1.04\ 10^{1}$ / $5.88\ 10^{5}$ / 20
500	$3.65\ 10^{1}$ / $4.83\ 10^{6}$ / 29	$5.35\ 10^{1}$ / $4.87\ 10^{6}$ / 28	$8.92\ 10^{1}$ / $5.02\ 10^{6}$ / 32
1000	$1.74\ 10^{2}$ / $2.26\ 10^{7}$ / 35	$2.56\ 10^{2}$ / $2.30\ 10^{7}$ / 35	$4.11\ 10^{2}$ / $2.29\ 10^{7}$ / 35
2000	$8.77\ 10^{2}$ / $1.09\ 10^{8}$ / 41	$1.20\ 10^{3}$ / $1.03\ 10^{8}$ / 41	$1.97\ 10^{3}$ / $1.05\ 10^{8}$ / 39
5000	$6.45\ 10^{3}$ / $7.99\ 10^{8}$ / 51	$9.46\ 10^{3}$ / $8.11\ 10^{8}$ / 51	$1.55\ 10^{4}$ / $8.27\ 10^{8}$ / 53
10000	$2.77\ 10^{4}$ / $3.50\ 10^{9}$ / 53	$4.25\ 10^{4}$ / $3.59\ 10^{9}$ / 57	$6.84\ 10^{4}$ / $3.64\ 10^{9}$ / 56

Table 7.11: Running time in seconds, No. of comparisons and No. of iterations for MINRAD2.

$n \setminus d$	16	32	64
100	$4.90\ 10^{-1}$ / $2.38\ 10^{4}$ / 19	$8.70\ 10^{-1}$ / $2.38\ 10^{4}$ / 23	$1.87\ 10^{0}$ / $2.57\ 10^{4}$ / 27
200	$1.98\ 10^{0}$ / $9.74\ 10^{4}$ / 25	$3.73\ 10^{0}$ / $9.86\ 10^{4}$ / 28	$7.25\ 10^{0}$ / $1.01\ 10^{5}$ / 29
500	$1.24\ 10^{1}$ / $6.00\ 10^{5}$ / 28	$2.38\ 10^{1}$ / $6.30\ 10^{5}$ / 37	$4.66\ 10^{1}$ / $6.48\ 10^{5}$ / 36
1000	$5.23\ 10^{1}$ / $2.50\ 10^{6}$ / 38	$9.52\ 10^{1}$ / $2.51\ 10^{6}$ / 42	$1.93\ 10^{2}$ / $2.67\ 10^{6}$ / 56
2000	$2.20\ 10^{2}$ / $1.04\ 10^{7}$ / 49	$3.91\ 10^{2}$ / $1.04\ 10^{7}$ / 49	$7.76\ 10^{2}$ / $1.09\ 10^{7}$ / 60
5000	$1.46\ 10^{3}$ / $6.95\ 10^{7}$ / 65	$2.55\ 10^{3}$ / $6.75\ 10^{7}$ / 66	$5.16\ 10^{3}$ / $7.24\ 10^{7}$ / 78
10000	$5.84\ 10^{3}$ / $2.77\ 10^{8}$ / 70	$1.03\ 10^{4}$ / $2.72\ 10^{8}$ / 74	$2.09\ 10^{4}$ / $2.92\ 10^{8}$ / 92
$n \setminus d$	128	256	512
100	$3.62\ 10^{0}$ / $2.60\ 10^{4}$ / 26	$6.81\ 10^{0}$ / $2.46\ 10^{4}$ / 26	$1.36\ 10^{1}$ / $2.49\ 10^{4}$ / 27
200	$1.37\ 10^{1}$ / $9.90\ 10^{4}$ / 32	$2.93\ 10^{1}$ / $1.06\ 10^{5}$ / 42	$5.49\ 10^{1}$ / $1.00\ 10^{5}$ / 32
500	$9.48\ 10^{1}$ / $6.77\ 10^{5}$ / 53	$1.99\ 10^{2}$ / $7.20\ 10^{5}$ / 53	$3.93\ 10^{2}$ / $7.16\ 10^{5}$ / 57
1000	$3.95\ 10^{2}$ / $2.81\ 10^{6}$ / 62	$8.05\ 10^{2}$ / $2.90\ 10^{6}$ / 72	$1.71\ 10^{3}$ / $3.12\ 10^{6}$ / 83
2000	$1.62\ 10^{3}$ / $1.17\ 10^{7}$ / 71	$3.35\ 10^{3}$ / $1.23\ 10^{7}$ / 89	$6.95\ 10^{3}$ / $1.28\ 10^{7}$ / 108
5000	$1.06\ 10^{4}$ / $7.67\ 10^{7}$ / 98	$2.18\ 10^{4}$ / $7.98\ 10^{7}$ / 117	$4.64\ 10^{4}$ / $8.59\ 10^{7}$ / 140
10000	$4.45\ 10^{4}$ / $3.20\ 10^{8}$ / 116	$9.35\ 10^{4}$ / $3.41\ 10^{8}$ / 148	$1.91\ 10^{5}$ / $3.52\ 10^{8}$ / 173

Table 7.12: Running time in seconds, No. of comparisons and No. of iterations for CTRD-MEAN.

$n \setminus d$	16	32	64
100	$5.50\ 10^{-1}$ / $2.84\ 10^{4}$ / 28	$1.15\ 10^{0}$ / $3.06\ 10^{4}$ / 29	$2.30\ 10^{0}$ / $3.17\ 10^{4}$ / 29
200	$2.52\ 10^{0}$ / $1.25\ 10^{5}$ / 36	$4.89\ 10^{0}$ / $1.32\ 10^{5}$ / 39	$9.61\ 10^{0}$ / $1.35\ 10^{5}$ / 37
500	$1.49\ 10^{1}$ / $7.46\ 10^{5}$ / 53	$3.32\ 10^{1}$ / $8.97\ 10^{5}$ / 57	$6.45\ 10^{1}$ / $9.07\ 10^{5}$ / 55
1000	$6.47\ 10^{1}$ / $3.21\ 10^{6}$ / 73	$1.37\ 10^{2}$ / $3.70\ 10^{6}$ / 77	$2.55\ 10^{2}$ / $3.58\ 10^{6}$ / 68
2000	$2.59\ 10^{2}$ / $1.27\ 10^{7}$ / 87	$5.46\ 10^{2}$ / $1.48\ 10^{7}$ / 92	$1.15\ 10^{3}$ / $1.63\ 10^{7}$ / 95
5000	$1.73\ 10^{3}$ / $8.56\ 10^{7}$ / 126	$3.75\ 10^{3}$ / $1.01\ 10^{8}$ / 136	$7.54\ 10^{3}$ / $1.06\ 10^{8}$ / 122
10000	$6.97\ 10^{3}$ / $3.44\ 10^{8}$ / 138	$1.54\ 10^{4}$ / $4.17\ 10^{8}$ / 162	$3.10\ 10^{4}$ / $4.38\ 10^{8}$ / 153
$n \setminus d$	128	256	512
100	$4.39\ 10^{0}$ / $3.18\ 10^{4}$ / 29	$8.12\ 10^{0}$ / $2.94\ 10^{4}$ / 27	$1.62\ 10^{1}$ / $2.96\ 10^{4}$ / 28
200	$1.73\ 10^{1}$ / $1.24\ 10^{5}$ / 32	$3.59\ 10^{1}$ / $1.30\ 10^{5}$ / 35	$7.04\ 10^{1}$ / $1.28\ 10^{5}$ / 36
500	$1.23\ 10^{2}$ / $8.82\ 10^{5}$ / 56	$2.42\ 10^{2}$ / $8.78\ 10^{5}$ / 57	$4.98\ 10^{2}$ / $9.08\ 10^{5}$ / 58
1000	$5.38\ 10^{2}$ / $3.86\ 10^{6}$ / 73	$1.05\ 10^{3}$ / $3.82\ 10^{6}$ / 81	$2.06\ 10^{3}$ / $3.74\ 10^{6}$ / 68
2000	$2.12\ 10^{3}$ / $1.55\ 10^{7}$ / 91	$4.23\ 10^{3}$ / $1.56\ 10^{7}$ / 88	$8.68\ 10^{3}$ / $1.61\ 10^{7}$ / 97
5000	$1.53\ 10^{4}$ / $1.11\ 10^{8}$ / 124	$2.91\ 10^{4}$ / $1.07\ 10^{8}$ / 122	$5.66\ 10^{4}$ / $1.04\ 10^{8}$ / 127
10000	$6.12\ 10^{4}$ / $4.43\ 10^{8}$ / 150	$1.21\ 10^{5}$ / $4.44\ 10^{8}$ / 161	$2.38\ 10^{5}$ / $4.39\ 10^{8}$ / 151

Table 7.13: Running time in seconds, No. of comparisons and No. of iterations for CTRD-SUM.

$n \setminus d$	16	32	64
100	1.98 10^0 / 1.86 10^5 / 28	1.98 10^0 / 1.88 10^5 / 29	2.47 10^0 / 2.12 10^5 / 30
200	1.14 10^1 / 1.05 10^6 / 40	1.05 10^1 / 1.00 10^6 / 39	1.10 10^1 / 9.73 10^5 / 35
500	8.44 10^1 / 8.51 10^6 / 51	9.71 10^1 / 9.34 10^6 / 57	9.71 10^1 / 8.55 10^6 / 52
1000	4.33 10^2 / 4.34 10^7 / 75	4.78 10^2 / 4.55 10^7 / 75	4.80 10^2 / 4.21 10^7 / 70
2000	2.42 10^3 / 2.15 10^8 / 93	2.41 10^3 / 2.04 10^8 / 89	2.47 10^3 / 1.94 10^8 / 81
5000	1.91 10^4 / 1.66 10^9 / 119	2.01 10^4 / 1.66 10^9 / 119	2.08 10^4 / 1.59 10^9 / 109
10000	8.90 10^4 / 7.62 10^9 / 140	9.85 10^4 / 8.05 10^9 / 142	9.33 10^4 / 7.01 10^9 / 126
$n \setminus d$	128	256	512
100	2.64 10^0 / 1.91 10^5 / 28	2.47 10^0 / 2.01 10^5 / 30	3.30 10^0 / 1.73 10^5 / 24
200	1.24 10^1 / 9.48 10^5 / 36	1.13 10^1 / 9.26 10^5 / 36	1.79 10^1 / 9.43 10^5 / 36
500	1.11 10^2 / 8.52 10^6 / 55	1.01 10^2 / 8.45 10^6 / 51	1.64 10^2 / 8.77 10^6 / 54
1000	5.37 10^2 / 4.08 10^7 / 67	4.88 10^2 / 4.05 10^7 / 67	7.67 10^2 / 4.08 10^7 / 66
2000	2.74 10^3 / 1.90 10^8 / 79	2.35 10^3 / 1.95 10^8 / 78	3.60 10^3 / 1.91 10^8 / 80
5000	2.29 10^4 / 1.55 10^9 / 106	1.87 10^4 / 1.51 10^9 / 107	2.90 10^4 / 1.51 10^9 / 101
10000	1.07 10^5 / 7.13 10^9 / 128	8.62 10^4 / 6.85 10^9 / 123	1.31 10^5 / 6.76 10^9 / 121

Table 7.14: Running time in seconds, No. of comparisons and No. of iterations for KRNLSUM.

$n \setminus d$	16	32	64
100	2.20 10^{-1} / 2.60 10^4	3.30 10^{-1} / 3.57 10^4	3.80 10^{-1} / 3.93 10^4
200	7.70 10^{-1} / 9.21 10^4	1.27 10^0 / 1.43 10^5	1.60 10^0 / 1.69 10^5
500	4.17 10^0 / 4.81 10^5	7.47 10^0 / 8.27 10^5	9.45 10^0 / 9.81 10^5
1000	1.50 10^1 / 1.73 10^6	2.93 10^1 / 3.24 10^6	4.93 10^1 / 5.05 10^6
2000	5.07 10^1 / 5.83 10^6	9.68 10^1 / 1.06 10^7	1.79 10^2 / 1.84 10^7
5000	3.69 10^2 / 3.29 10^7	1.00 10^3 / 8.55 10^7	1.56 10^3 / 1.26 10^8
10000	1.35 10^3 / 1.12 10^8	3.79 10^3 / 3.02 10^8	6.08 10^3 / 4.64 10^8
$n \setminus d$	128	256	512
100	3.90 10^{-1} / 4.75 10^4	7.70 10^{-1} / 6.81 10^4	1.87 10^0 / 1.01 10^5
200	1.48 10^0 / 1.88 10^5	3.57 10^0 / 3.21 10^5	6.98 10^0 / 3.90 10^5
500	1.22 10^1 / 1.61 10^6	2.08 10^1 / 1.89 10^6	4.49 10^1 / 2.52 10^6
1000	4.29 10^1 / 5.71 10^6	8.83 10^1 / 8.07 10^6	1.96 10^2 / 1.10 10^7
2000	1.66 10^2 / 2.22 10^7	4.04 10^2 / 3.71 10^7	7.36 10^2 / 4.14 10^7
5000	1.27 10^3 / 1.66 10^8	2.34 10^3 / 2.11 10^8	5.02 10^3 / 2.77 10^8
10000	4.85 10^3 / 6.09 10^8	9.82 10^3 / 8.63 10^8	2.08 10^4 / 1.14 10^9

Table 7.15: Running time in seconds and No. of comparisons for DIVIDIAM.

$n \setminus d$	16	32	64
100	$9.40\ 10^{-1}$ / $1.25\ 10^{5}$	$1.37\ 10^{0}$ / $1.73\ 10^{5}$	$1.93\ 10^{0}$ / $2.27\ 10^{5}$
200	$3.96\ 10^{0}$ / $5.42\ 10^{5}$	$6.10\ 10^{0}$ / $7.60\ 10^{5}$	$9.17\ 10^{0}$ / $1.06\ 10^{6}$
500	$2.87\ 10^{1}$ / $3.49\ 10^{6}$	$4.82\ 10^{1}$ / $5.61\ 10^{6}$	$6.60\ 10^{1}$ / $7.33\ 10^{6}$
1000	$1.24\ 10^{2}$ / $1.45\ 10^{7}$	$2.07\ 10^{2}$ / $2.33\ 10^{7}$	$3.12\ 10^{2}$ / $3.37\ 10^{7}$
2000	$5.12\ 10^{2}$ / $5.89\ 10^{7}$	$8.72\ 10^{2}$ / $9.58\ 10^{7}$	$1.31\ 10^{3}$ / $1.41\ 10^{8}$
5000	$4.17\ 10^{3}$ / $3.77\ 10^{8}$	$7.44\ 10^{3}$ / $6.62\ 10^{8}$	$1.05\ 10^{4}$ / $9.40\ 10^{8}$
10000	$1.67\ 10^{4}$ / $1.52\ 10^{9}$	$3.06\ 10^{4}$ / $2.68\ 10^{9}$	$4.56\ 10^{4}$ / $3.99\ 10^{9}$
$n \setminus d$	128	256	512
100	$2.42\ 10^{0}$ / $3.16\ 10^{5}$	$4.89\ 10^{0}$ / $4.50\ 10^{5}$	$1.09\ 10^{1}$ / $6.17\ 10^{5}$
200	$1.06\ 10^{1}$ / $1.42\ 10^{6}$	$2.27\ 10^{1}$ / $2.07\ 10^{6}$	$4.89\ 10^{1}$ / $2.74\ 10^{6}$
500	$7.92\ 10^{1}$ / $1.05\ 10^{7}$	$1.58\ 10^{2}$ / $1.44\ 10^{7}$	$3.61\ 10^{2}$ / $2.02\ 10^{7}$
1000	$3.35\ 10^{2}$ / $4.44\ 10^{7}$	$7.07\ 10^{2}$ / $6.42\ 10^{7}$	$1.56\ 10^{3}$ / $8.74\ 10^{7}$
2000	$1.47\ 10^{3}$ / $1.94\ 10^{8}$	$3.05\ 10^{3}$ / $2.76\ 10^{8}$	$6.76\ 10^{3}$ / $3.77\ 10^{8}$
5000	$1.01\ 10^{4}$ / $1.31\ 10^{9}$	$2.05\ 10^{4}$ / $1.83\ 10^{9}$	$4.74\ 10^{4}$ / $2.60\ 10^{9}$
10000	$4.47\ 10^{4}$ / $5.54\ 10^{9}$	$8.94\ 10^{4}$ / $7.75\ 10^{9}$	$2.03\ 10^{5}$ / $1.10\ 10^{10}$

Table 7.16: Running time in seconds and No. of comparisons for DIVIDIA2.

7.6.2 Figures for Uniformly Distributed Random Vertex Sets

Section 7.6.2 shows some of the experimental data as curves. The first group (Figures 7.11 to 7.15) shows the number of vertex comparisons over the number n of elements. The second group (Figures 7.16 to 7.18) gives the running time needed to compute the entire cluster hierarchy in seconds. In the last part (Figures 7.19 and 7.20) the number of iterations of the WHILE loop for algorithms based on GHC2 is shown.

For all figures there are six curves drawn for dimension $d = 16, 32, 64, 128, 256, 512$. For a discussion of these curves refer to Section 7.4.

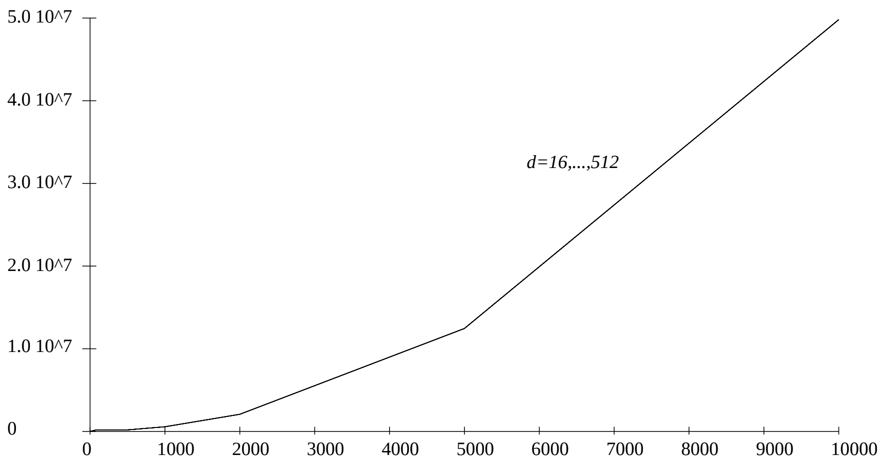


Figure 7.11: Number of vertex comparisons as function of the number of elements for SLINK.

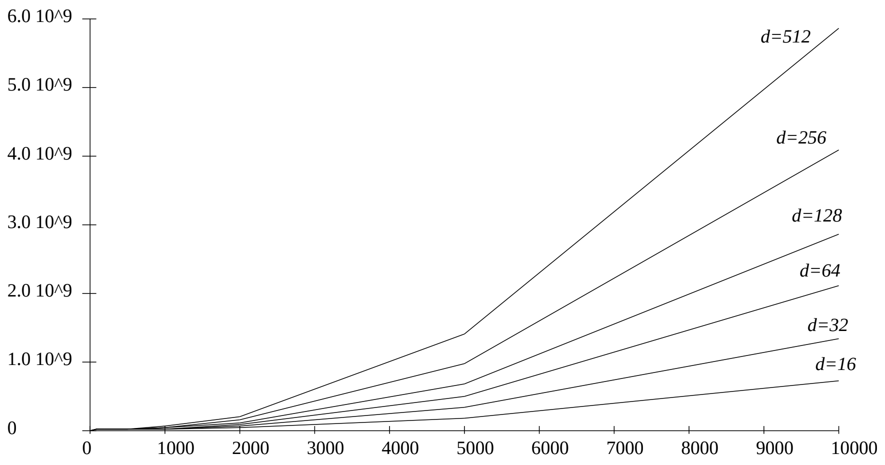


Figure 7.12: Number of vertex comparisons as function of the number of elements for CLINK1.

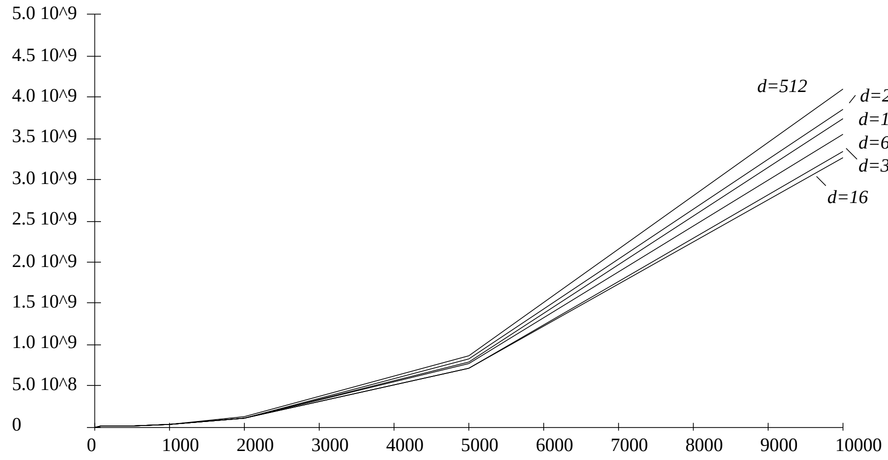


Figure 7.13: Number of vertex comparisons as function of the number of elements for CLINK2.

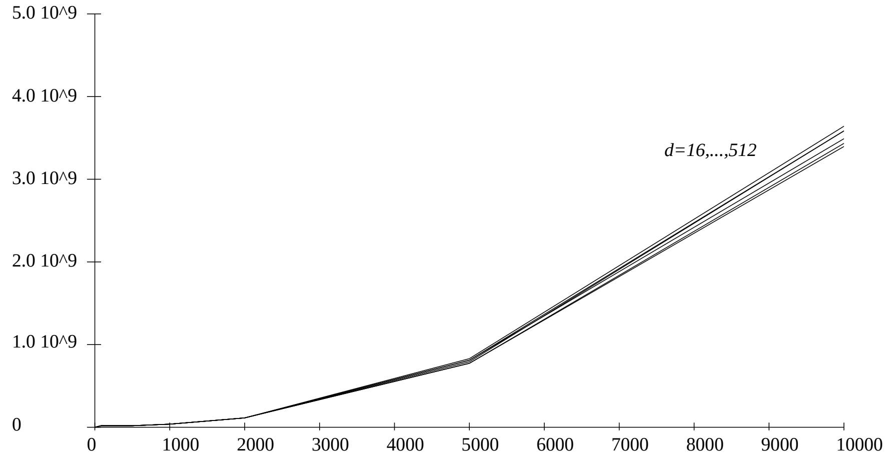


Figure 7.14: Number of vertex comparisons as function of the number of elements for MIN-RAD2.

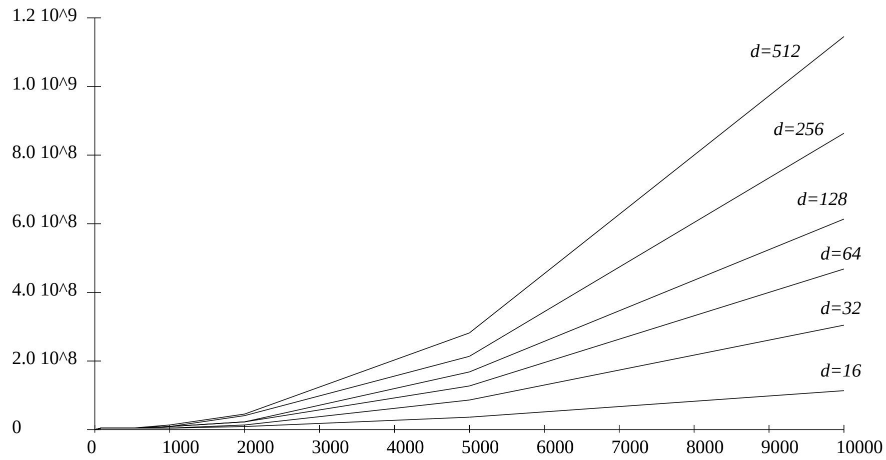


Figure 7.15: Number of vertex comparisons as function of the number of elements for DI-VIDIAM.

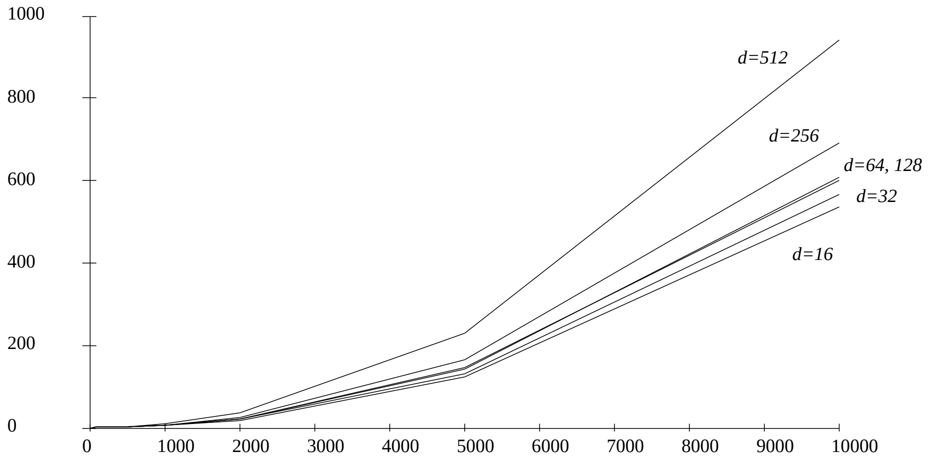


Figure 7.16: Running time in seconds as function of the number of elements for SLINK.

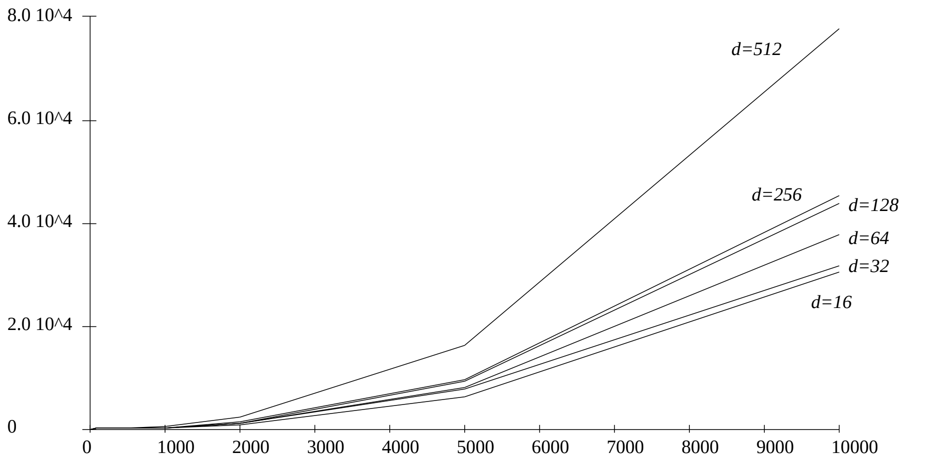


Figure 7.17: Running time in seconds as function of the number of elements for CLINK2.

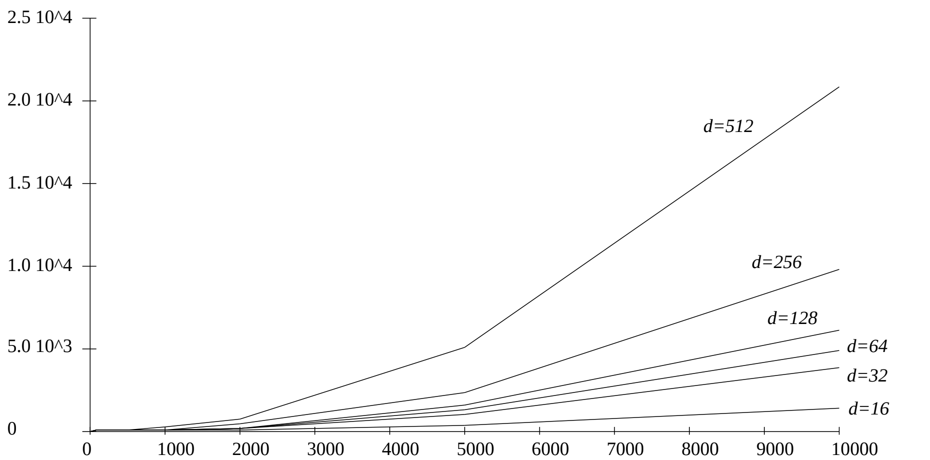


Figure 7.18: Running time in seconds as function of the number of elements for DIVIDIAM.

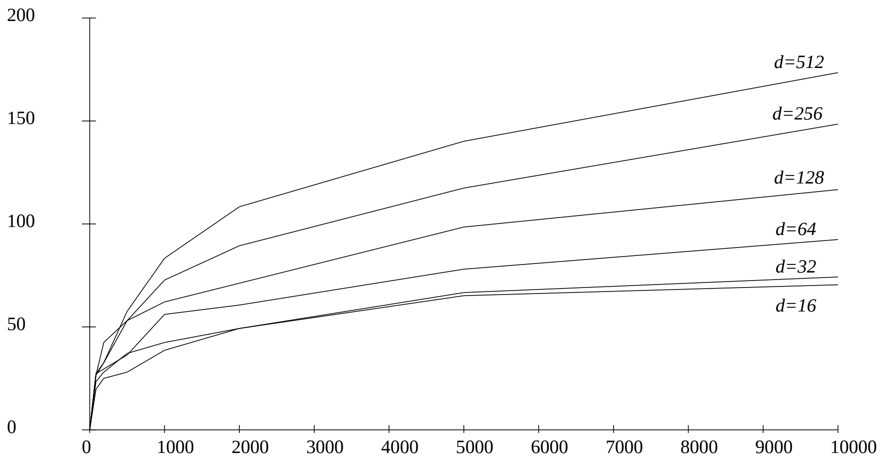


Figure 7.19: Number of iterations of the WHILE loop as function of the number of elements for CTRDMEAN.

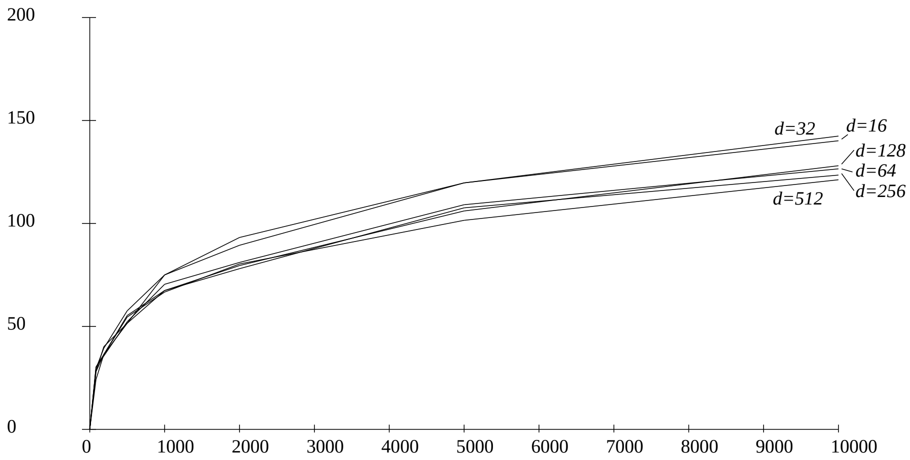


Figure 7.20: Number of iterations of the WHILE loop as function of the number of elements for KRNLSUM.

7.6.3 Tables for Special Vertex Sets with 10 Cluster Centers

Section 7.6.3 summarizes the experimental results for 39 special point sets described in Section 7.4. A discussion of the results can also be found there.

Tables 7.17 to 7.19 show the running time of the algorithms in seconds, and the number of comparisons, respectively, in dependency of the dimension d and the number of elements n. Tables 7.20 and 7.21 show these results for some methods based on Algorithm GHC2 and give in addition the number of iterations of the WHILE loop (cf. Figure 7.4). Finally Table 7.22 gives the results for the divisive method DIVIDIAM.

$n \setminus d$	16	32	64
100	$6.00\ 10^{-2}$ / $4.95\ 10^{3}$	$5.00\ 10^{-2}$ / $4.95\ 10^{3}$	$5.00\ 10^{-2}$ / $4.95\ 10^{3}$
200	$1.60\ 10^{-1}$ / $1.99\ 10^{4}$	$1.80\ 10^{-1}$ / $1.99\ 10^{4}$	$1.20\ 10^{-1}$ / $1.99\ 10^{4}$
500	$1.04\ 10^{0}$ / $1.24\ 10^{5}$	$9.20\ 10^{-1}$ / $1.24\ 10^{5}$	$6.60\ 10^{-1}$ / $1.24\ 10^{5}$
1000	$4.45\ 10^{0}$ / $4.99\ 10^{5}$	$3.60\ 10^{0}$ / $4.99\ 10^{5}$	$2.80\ 10^{0}$ / $4.99\ 10^{5}$
2000	-	$1.42\ 10^{1}$ / $1.99\ 10^{6}$	$1.12\ 10^{1}$ / $1.99\ 10^{6}$
5000	-	$1.04\ 10^{2}$ / $1.25\ 10^{7}$	$7.44\ 10^{1}$ / $1.25\ 10^{7}$
10000	-	$4.45\ 10^{2}$ / $5.00\ 10^{7}$	$3.17\ 10^{2}$ / $5.00\ 10^{7}$
$n \setminus d$	128	256	512
100	$5.00\ 10^{-2}$ / $4.95\ 10^{3}$	$5.00\ 10^{-2}$ / $4.95\ 10^{3}$	$1.20\ 10^{-1}$ / $4.95\ 10^{3}$
200	$1.80\ 10^{-1}$ / $1.99\ 10^{4}$	$2.40\ 10^{-1}$ / $1.99\ 10^{4}$	$3.60\ 10^{-1}$ / $1.99\ 10^{4}$
500	$9.20\ 10^{-1}$ / $1.24\ 10^{5}$	$1.34\ 10^{0}$ / $1.24\ 10^{5}$	$2.20\ 10^{0}$ / $1.24\ 10^{5}$
1000	$3.66\ 10^{0}$ / $4.99\ 10^{5}$	$5.36\ 10^{0}$ / $4.99\ 10^{5}$	$8.91\ 10^{0}$ / $4.99\ 10^{5}$
2000	$1.47\ 10^{1}$ / $1.99\ 10^{6}$	$2.16\ 10^{1}$ / $1.99\ 10^{6}$	$3.54\ 10^{1}$ / $1.99\ 10^{6}$
5000	$9.58\ 10^{1}$ / $1.25\ 10^{7}$	$1.39\ 10^{2}$ / $1.25\ 10^{7}$	$2.25\ 10^{2}$ / $1.25\ 10^{7}$
10000	$4.02\ 10^{2}$ / $5.00\ 10^{7}$	$5.77\ 10^{2}$ / $5.00\ 10^{7}$	$9.17\ 10^{2}$ / $5.00\ 10^{7}$

Table 7.17: Running time in seconds and No. of comparisons for SLINK.

$n \setminus d$	16	32	64
100	$2.80\ 10^{-1}$ / $3.90\ 10^{4}$	$3.60\ 10^{-1}$ / $6.18\ 10^{4}$	$7.80\ 10^{-1}$ / $1.14\ 10^{5}$
200	$8.80\ 10^{-1}$ / $1.24\ 10^{5}$	$1.46\ 10^{0}$ / $2.33\ 10^{5}$	$2.98\ 10^{0}$ / $4.14\ 10^{5}$
500	$6.42\ 10^{0}$ / $8.36\ 10^{5}$	$1.01\ 10^{1}$ / $1.57\ 10^{6}$	$1.90\ 10^{1}$ / $2.52\ 10^{6}$
1000	$2.70\ 10^{1}$ / $3.35\ 10^{6}$	$4.00\ 10^{1}$ / $6.05\ 10^{6}$	$8.18\ 10^{1}$ / $1.06\ 10^{7}$
2000	-	$1.48\ 10^{2}$ / $2.20\ 10^{7}$	$3.25\ 10^{2}$ / $4.14\ 10^{7}$
5000	-	$1.10\ 10^{3}$ / $1.46\ 10^{8}$	$2.24\ 10^{3}$ / $2.61\ 10^{8}$
10000	-	$4.18\ 10^{3}$ / $5.36\ 10^{8}$	$8.84\ 10^{3}$ / $9.90\ 10^{8}$
$n \setminus d$	128	256	512
100	$1.65\ 10^{0}$ / $1.83\ 10^{5}$	$3.11\ 10^{0}$ / $2.95\ 10^{5}$	$6.52\ 10^{0}$ / $3.74\ 10^{5}$
200	$6.90\ 10^{0}$ / $7.55\ 10^{5}$	$1.36\ 10^{1}$ / $1.28\ 10^{6}$	$3.65\ 10^{1}$ / $2.08\ 10^{6}$
500	$4.38\ 10^{1}$ / $4.66\ 10^{6}$	$9.05\ 10^{1}$ / $8.53\ 10^{6}$	$2.59\ 10^{2}$ / $1.48\ 10^{7}$
1000	$1.73\ 10^{2}$ / $1.80\ 10^{7}$	$3.66\ 10^{2}$ / $3.43\ 10^{7}$	$1.05\ 10^{3}$ / $6.02\ 10^{7}$
2000	$7.14\ 10^{2}$ / $7.35\ 10^{7}$	$1.48\ 10^{3}$ / $1.38\ 10^{8}$	$4.41\ 10^{3}$ / $2.50\ 10^{8}$
5000	$4.72\ 10^{3}$ / $4.55\ 10^{8}$	$9.40\ 10^{3}$ / $8.62\ 10^{8}$	$2.84\ 10^{4}$ / $1.60\ 10^{9}$
10000	$2.00\ 10^{4}$ / $1.85\ 10^{9}$	$3.76\ 10^{4}$ / $3.41\ 10^{9}$	$1.17\ 10^{5}$ / $6.54\ 10^{9}$

Table 7.18: Running time in seconds and No. of comparisons for CLINK1.

$n \setminus d$	16	32	64
100	$3.80\ 10^{-1}$ / $4.58\ 10^{4}$	$4.40\ 10^{-1}$ / $6.59\ 10^{4}$	$5.40\ 10^{-1}$ / $9.24\ 10^{4}$
200	$1.27\ 10^{0}$ / $1.63\ 10^{5}$	$2.20\ 10^{0}$ / $2.64\ 10^{5}$	$2.36\ 10^{0}$ / $3.88\ 10^{5}$
500	$8.35\ 10^{0}$ / $1.05\ 10^{6}$	$1.67\ 10^{1}$ / $1.87\ 10^{6}$	$2.03\ 10^{1}$ / $3.40\ 10^{6}$
1000	$3.92\ 10^{1}$ / $4.72\ 10^{6}$	$6.69\ 10^{1}$ / $7.51\ 10^{6}$	$1.20\ 10^{2}$ / $2.01\ 10^{7}$
2000	-	$2.42\ 10^{2}$ / $2.75\ 10^{7}$	$5.73\ 10^{2}$ / $9.46\ 10^{7}$
5000	-	$2.11\ 10^{3}$ / $2.03\ 10^{8}$	$3.12\ 10^{3}$ / $5.16\ 10^{8}$
10000	-	$9.92\ 10^{3}$ / $9.05\ 10^{8}$	$1.06\ 10^{4}$ / $1.72\ 10^{9}$
$n \setminus d$	128	256	512
100	$1.21\ 10^{0}$ / $1.48\ 10^{5}$	$2.52\ 10^{0}$ / $2.13\ 10^{5}$	$5.21\ 10^{0}$ / $2.82\ 10^{5}$
200	$4.83\ 10^{0}$ / $5.93\ 10^{5}$	$1.06\ 10^{1}$ / $9.15\ 10^{5}$	$2.85\ 10^{1}$ / $1.54\ 10^{6}$
500	$3.70\ 10^{1}$ / $4.70\ 10^{6}$	$7.81\ 10^{1}$ / $6.82\ 10^{6}$	$2.00\ 10^{2}$ / $1.08\ 10^{7}$
1000	$2.07\ 10^{2}$ / $2.65\ 10^{7}$	$3.82\ 10^{2}$ / $3.33\ 10^{7}$	$9.46\ 10^{2}$ / $5.14\ 10^{7}$
2000	$1.46\ 10^{3}$ / $1.86\ 10^{8}$	$2.29\ 10^{3}$ / $1.99\ 10^{8}$	$4.88\ 10^{3}$ / $2.65\ 10^{8}$
5000	$1.78\ 10^{4}$ / $2.24\ 10^{9}$	$2.94\ 10^{4}$ / $2.56\ 10^{9}$	$5.40\ 10^{4}$ / $2.93\ 10^{9}$
10000	$1.04\ 10^{5}$ / $1.30\ 10^{10}$	$2.16\ 10^{5}$ / $1.88\ 10^{10}$	$3.84\ 10^{5}$ / $2.06\ 10^{10}$

Table 7.19: Running time in seconds and No. of comparisons for MINRADI.

$n \setminus d$	16	32	64
100	$9.90\ 10^{-1}$ / $1.11\ 10^{5}$ / 14	$8.00\ 10^{-1}$ / $1.15\ 10^{5}$ / 16	$9.60\ 10^{-1}$ / $1.27\ 10^{5}$ / 17
200	$4.61\ 10^{0}$ / $5.29\ 10^{5}$ / 17	$4.33\ 10^{0}$ / $6.09\ 10^{5}$ / 20	$5.80\ 10^{0}$ / $7.28\ 10^{5}$ / 26
500	$4.53\ 10^{1}$ / $5.08\ 10^{6}$ / 28	$3.65\ 10^{1}$ / $5.15\ 10^{6}$ / 29	$4.94\ 10^{1}$ / $6.14\ 10^{6}$ / 35
1000	$2.44\ 10^{2}$ / $2.70\ 10^{7}$ / 37	$1.67\ 10^{2}$ / $2.34\ 10^{7}$ / 31	$2.30\ 10^{2}$ / $2.84\ 10^{7}$ / 45
2000	-	$9.00\ 10^{2}$ / $1.13\ 10^{8}$ / 37	$1.23\ 10^{3}$ / $1.38\ 10^{8}$ / 57
5000	-	$7.96\ 10^{3}$ / $9.77\ 10^{8}$ / 55	$9.83\ 10^{3}$ / $1.08\ 10^{9}$ / 65
10000	-	$3.78\ 10^{4}$ / $4.59\ 10^{9}$ / 65	$4.66\ 10^{4}$ / $5.07\ 10^{9}$ / 76
$n \setminus d$	128	256	512
100	$9.70\ 10^{-1}$ / $1.31\ 10^{5}$ / 18	$1.46\ 10^{0}$ / $1.35\ 10^{5}$ / 17	$2.62\ 10^{0}$ / $1.47\ 10^{5}$ / 22
200	$5.74\ 10^{0}$ / $7.77\ 10^{5}$ / 26	$8.66\ 10^{0}$ / $7.93\ 10^{5}$ / 26	$1.59\ 10^{1}$ / $8.99\ 10^{5}$ / 31
500	$4.87\ 10^{1}$ / $6.53\ 10^{6}$ / 37	$8.56\ 10^{1}$ / $7.82\ 10^{6}$ / 42	$1.66\ 10^{2}$ / $9.36\ 10^{6}$ / 53
1000	$2.49\ 10^{2}$ / $3.33\ 10^{7}$ / 53	$4.56\ 10^{2}$ / $4.15\ 10^{7}$ / 65	$9.11\ 10^{2}$ / $5.11\ 10^{7}$ / 80
2000	$1.27\ 10^{3}$ / $1.69\ 10^{8}$ / 76	$2.25\ 10^{3}$ / $2.04\ 10^{8}$ / 87	$4.34\ 10^{3}$ / $2.41\ 10^{8}$ / 100
5000	$1.02\ 10^{4}$ / $1.32\ 10^{9}$ / 107	$1.85\ 10^{4}$ / $1.64\ 10^{9}$ / 137	$3.59\ 10^{4}$ / $1.97\ 10^{9}$ / 159
10000	$4.86\ 10^{4}$ / $6.31\ 10^{9}$ / 128	$8.68\ 10^{4}$ / $7.68\ 10^{9}$ / 183	$1.71\ 10^{5}$ / $9.42\ 10^{9}$ / 218

Table 7.20: Running time in seconds, No. of comparisons and No. of iterations for CLINK2.

$n \setminus d$	16	32	64
100	$1.43\ 10^{0}$ / $1.47\ 10^{5}$ / 17	$1.76\ 10^{0}$ / $1.64\ 10^{5}$ / 20	$2.03\ 10^{0}$ / $1.73\ 10^{5}$ / 22
200	$7.36\ 10^{0}$ / $7.37\ 10^{5}$ / 24	$8.46\ 10^{0}$ / $8.09\ 10^{5}$ / 24	$1.08\ 10^{1}$ / $9.43\ 10^{5}$ / 29
500	$5.99\ 10^{1}$ / $5.91\ 10^{6}$ / 31	$7.00\ 10^{1}$ / $6.66\ 10^{6}$ / 34	$9.17\ 10^{1}$ / $8.00\ 10^{6}$ / 44
1000	$3.45\ 10^{2}$ / $3.33\ 10^{7}$ / 50	$3.47\ 10^{2}$ / $3.31\ 10^{7}$ / 43	$4.56\ 10^{2}$ / $3.94\ 10^{7}$ / 55
2000	-	$1.84\ 10^{3}$ / $1.56\ 10^{8}$ / 52	$2.56\ 10^{3}$ / $2.00\ 10^{8}$ / 71
5000	-	$1.43\ 10^{4}$ / $1.20\ 10^{9}$ / 70	$2.05\ 10^{4}$ / $1.60\ 10^{9}$ / 93
10000	-	$6.57\ 10^{4}$ / $5.49\ 10^{9}$ / 80	$9.85\ 10^{4}$ / $7.67\ 10^{9}$ / 117
$n \setminus d$	128	256	512
100	$2.09\ 10^{0}$ / $1.55\ 10^{5}$ / 19	$2.15\ 10^{0}$ / $1.72\ 10^{5}$ / 20	$3.73\ 10^{0}$ / $1.94\ 10^{5}$ / 26
200	$1.25\ 10^{1}$ / $9.50\ 10^{5}$ / 30	$1.25\ 10^{1}$ / $1.00\ 10^{6}$ / 31	$2.03\ 10^{1}$ / $1.05\ 10^{6}$ / 35
500	$1.13\ 10^{2}$ / $8.46\ 10^{6}$ / 44	$1.20\ 10^{2}$ / $9.66\ 10^{6}$ / 51	$2.05\ 10^{2}$ / $1.05\ 10^{7}$ / 56
1000	$5.86\ 10^{2}$ / $4.34\ 10^{7}$ / 62	$6.47\ 10^{2}$ / $5.16\ 10^{7}$ / 73	$1.15\ 10^{3}$ / $5.93\ 10^{7}$ / 88
2000	$3.70\ 10^{3}$ / $2.52\ 10^{8}$ / 94	$3.58\ 10^{3}$ / $2.85\ 10^{8}$ / 106	$6.32\ 10^{3}$ / $3.23\ 10^{8}$ / 122
5000	$3.20\ 10^{4}$ / $2.18\ 10^{9}$ / 136	$3.34\ 10^{4}$ / $2.69\ 10^{9}$ / 173	$6.24\ 10^{4}$ / $3.21\ 10^{9}$ / 209
10000	$1.61\ 10^{5}$ / $1.10\ 10^{10}$ / 177	$1.80\ 10^{5}$ / $1.42\ 10^{10}$ / 237	$3.42\ 10^{5}$ / $1.74\ 10^{10}$ / 300

Table 7.21: Running time in seconds, No. of comparisons and No. of iterations for KRNLSUM.

$n \setminus d$	16	32	64
100	$2.20\ 10^{-1}$ / $2.26\ 10^{4}$	$2.80\ 10^{-1}$ / $2.80\ 10^{4}$	$4.40\ 10^{-1}$ / $4.43\ 10^{4}$
200	$7.20\ 10^{-1}$ / $7.96\ 10^{4}$	$9.90\ 10^{-1}$ / $1.04\ 10^{5}$	$1.59\ 10^{0}$ / $1.64\ 10^{5}$
500	$4.78\ 10^{0}$ / $5.57\ 10^{5}$	$7.19\ 10^{0}$ / $8.04\ 10^{5}$	$1.04\ 10^{1}$ / $1.07\ 10^{6}$
1000	$1.71\ 10^{1}$ / $1.98\ 10^{6}$	$2.55\ 10^{1}$ / $2.80\ 10^{6}$	$4.49\ 10^{1}$ / $4.58\ 10^{6}$
2000	-	$9.39\ 10^{1}$ / $1.02\ 10^{7}$	$1.67\ 10^{2}$ / $1.69\ 10^{7}$
5000	-	$8.76\ 10^{2}$ / $7.97\ 10^{7}$	$1.07\ 10^{3}$ / $9.38\ 10^{7}$
10000	-	$3.26\ 10^{3}$ / $2.83\ 10^{8}$	$4.44\ 10^{3}$ / $3.67\ 10^{8}$
$n \setminus d$	128	256	512
100	$4.40\ 10^{-1}$ / $4.85\ 10^{4}$	$6.60\ 10^{-1}$ / $5.82\ 10^{4}$	$1.43\ 10^{0}$ / $7.80\ 10^{4}$
200	$1.26\ 10^{0}$ / $1.61\ 10^{5}$	$2.69\ 10^{0}$ / $2.38\ 10^{5}$	$5.82\ 10^{0}$ / $3.22\ 10^{5}$
500	$9.01\ 10^{0}$ / $1.17\ 10^{6}$	$1.49\ 10^{1}$ / $1.34\ 10^{6}$	$3.16\ 10^{1}$ / $1.76\ 10^{6}$
1000	$3.64\ 10^{1}$ / $4.80\ 10^{6}$	$6.28\ 10^{1}$ / $5.67\ 10^{6}$	$1.48\ 10^{2}$ / $8.27\ 10^{6}$
2000	$1.52\ 10^{2}$ / $2.02\ 10^{7}$	$2.96\ 10^{2}$ / $2.69\ 10^{7}$	$6.47\ 10^{2}$ / $3.62\ 10^{7}$
5000	$8.81\ 10^{2}$ / $1.14\ 10^{8}$	$1.77\ 10^{3}$ / $1.59\ 10^{8}$	$4.45\ 10^{3}$ / $2.47\ 10^{8}$
10000	$3.68\ 10^{3}$ / $4.62\ 10^{8}$	$6.27\ 10^{3}$ / $5.48\ 10^{8}$	$1.60\ 10^{4}$ / $8.78\ 10^{8}$

Table 7.22: Running time in seconds and No. of comparisons for DIVIDIAM.

7.6.4 Figures for Special Vertex Sets with 10 Cluster Centers

Section 7.6.4 shows some of the results of the tables in Section 7.6.3 as curves. The first group (Figures 7.21 to 7.24) shows the number of vertex comparisons over the number n of elements. The last Figure (No. 7.25) gives the number of iterations of the WHILE loop for the algorithm KRNLSUM.

For all figures we give five curves for dimension $d = 32, 64, 128, 256, 512$. For a discussion of these curves refer to Section 7.4.

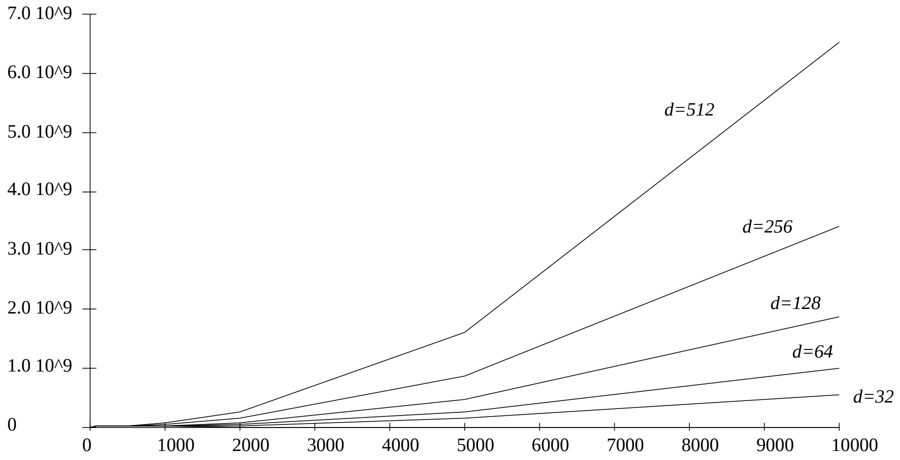


Figure 7.21: Number of vertex comparisons as function of the number of elements for CLINK1.

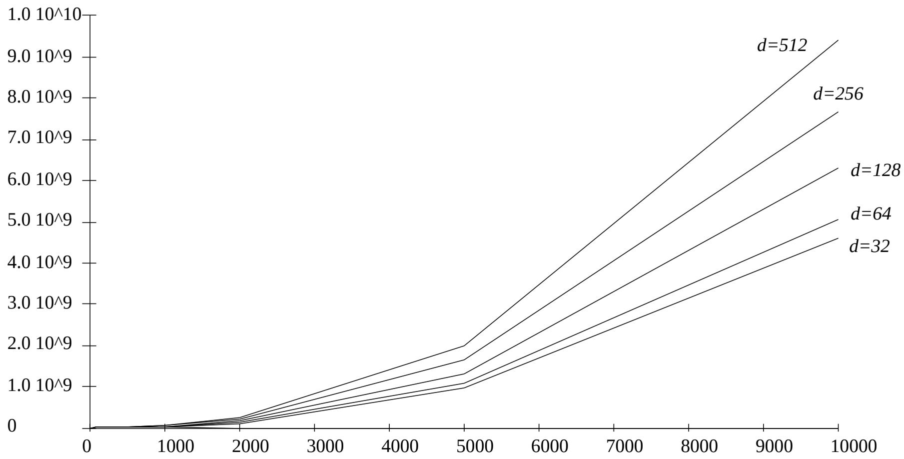


Figure 7.22: Number of vertex comparisons as function of the number of elements for CLINK2.

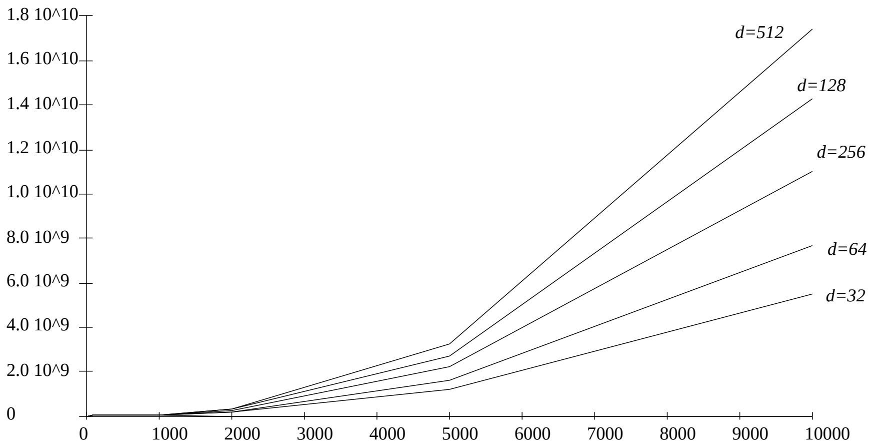


Figure 7.23: Number of vertex comparisons as function of the number of elements for KRNL-SUM.

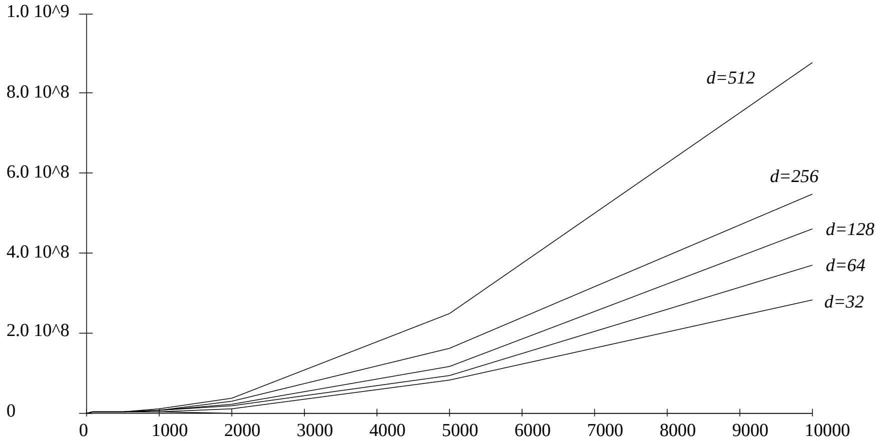


Figure 7.24: Number of vertex comparisons as function of the number of elements for DI-VIDIAM.

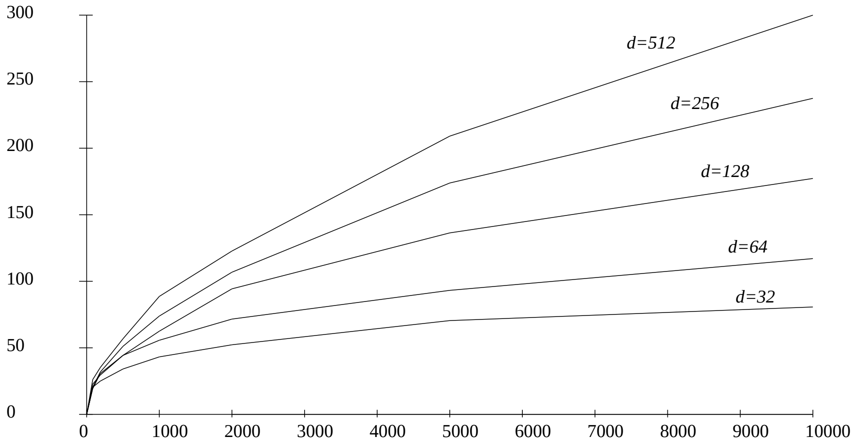


Figure 7.25: Number of iterations of the WHILE loop as function of the number of elements for KRNLSUM.

Chapter 8

Efficient $\{0, 1\}$-String Searching Based on Pre-clustering

8.1 Introduction

8.1.1 Notation and Problem Definition

In this chapter we consider the {0,1}-string search problem. For a given set S of binary strings of fixed length d and a query string q we ask for the most similar string in S. Thereby the dissimilarity of two given strings is the number of disagreeing bits, that is, their Hamming distance. A set of such strings can be seen as a subset of the vertices of a d-dimensional hypercube, and thus the {0,1}-string search problem is a geometric problem in d dimensions.

A formal definition of the string search problem is given below.

{0,1}-String Search Problem:
INSTANCE: Given a set $C \subset V^d$ with $|C| = n$.
QUESTION: For a query string $x \in V^d$ report an element c which attains $\min_{c \in C}\{d_H(x, c)\}$.

In other words, for a given set C of n strings of length d and a query string x we search for the best match for x in C, i.e. an element of C with the minimum Hamming distance to x. Typical applications may have string length of, say, 64 to 512 bits or more, and several thousands to tens of thousands strings. Usually we can expect a huge number of queries for the same set, so that a preprocessing step for the problem is appropriate.

Trivially this problem can be solved in $O(dn)$ time by exhaustive comparison. Note that the constant in this notation is rather small since typical computers can compare 16 or 32 bits within one operations. Including some overhead for storage managing, a typical constant in the big O notation may therefore be $\frac{1}{10}$. Thus if looking for faster practical algorithms we not only have to look for asymptotic running times of $o(dn)$ but also have to ensure small constants.

Clearly the Hamming distance is symmetric and invariant where invariance means that d_H is not affected by permuting coordinates (each bit in a string is equally important), and symmetry means that the probability of being '1' is $1/2$ for all bits in a string randomly chosen from a given set. Suppose this would not be the case, e.g. we can weight the bits according to their position. Clearly any such weighting which fulfills some obvious requirements leads to a fast sorting and searching solution similar to the binary bisection searching. Even if these requirements are only partially fulfilled we might use sorting to find a near optimal solution and then to do some local search according to the sorting. Although this might lead to interesting related problems we will not go into more detail for these special cases.

For a uniformly distributed random vertex set C and a random query point q it is very likely to have a large optimal Hamming distance, typically in the range $\{\frac{d}{3},\ldots,\frac{d}{2}\}$. But for practical applications this distance is much too large since we can not speak of a "very similar" object if about $\frac{1}{3}$ of the bits differ. Therefore we will not only consider random queries but also query vertices which are close to some vertex of the set C, i.e. which guarantee to have a small Hamming distance. We believe that this meets the needs of applications, since for applications there are most likely several groups of very similar "objects". The object, for which the search is done, might belong to one of these groups, that is, it might have a very small Hamming distance to its nearest neighbour. We will therefore study the number of necessary distance computations in comparison to the Hamming distance between the query vertex and its optimal solution. In fact it will turn out that this is a very distinctive relation.

Surprisingly it seems that that up to now there is no simple efficient solution even if we allow to use a wasteful preprocessing. In fact, M. Minsky conjectured in [75] that "...for large data sets with long word length there are no practical alternatives to large searches that inspect large parts of the memory.".

We will give a brief overview of previous approaches in Section 8.1.2. In Section 8.2 we then present an efficient {0,1}-string search algorithm based on hierarchical pre-clustering. To this end we also give several useful observations on the inter- and intra-cluster distances within a cluster tree. Section 8.3 gives the results of an exhaustive number of experiments, indicating that our algorithms work well in practice.

A previous version of this chapter appeared as [5].

8.1.2 Related Work

There are several possible attempts to the $\{0,1\}$-string search problem which can be found in the literature. A possible approach, which works well in two space, is the use of the well known Voronoi-diagram, see [11] for a survey. Unfortunately the storage needs grow exponentially with the dimension (according to the maximal size of a polytope in $d+1$ dimensions, $O(n^{\lceil d/2 \rceil})$), and is therefore not suitable for the d-cube case.

Approximate Elimination Search Algorithm (AESA) and Variants

In his paper Vidal [108] introduced a new technique for finding nearest neighbours in (approximately) constant time, the so called Approximate Elimination Search Algorithm (AESA). This approach is known to work quite well for low dimensional point sets in $\mathcal{R}^d$, where typical values of d are 3 to 8.

There are several improvements reported on the AESA, e.g. a version using only linear storage by Ramasubramanian and Paliwal [88]. This approach is based on a new spherical distance coordinate formulation, the so called (Incremental) Fixed Anchor Point-AESA

((I)FAP-AESA). Moreover in [109] and [110] new formulations and slight extensions of [108] are given, e.g. the Reduced Overhead AESA (ROAESA).

In all these papers the application to the hypercube with the Hamming distance metric is reported, but the examples given there are only for dimension 10 or less. This means that we have at most 1024 possible locations for vertices. Thus a full table for any possible query point with the location of its best matching string in the given set C can easily be maintained. Storing the 10 bit information of the optimal solution for every potential query point needs only 2048 byte (exactly 2KB) and can be initialized with the trivial $O(dn)$ procedure within some milliseconds. Any query now can clearly be answered in constant time with only one index operation and beats thus all AESA based methods by far. Moreover even the overall storage for data and program is in this case much smaller than the storage needed for an implementation of AESA or its variants.

Thus the examples for hypercubes given in this papers are to small to be significant. We therefore considered the mentioned nearest neighbour methods also for higher dimensions. Our experiments showed that AESA and its variants are not suitable for the high dimensional hypercube case, which was also reported by other researchers [10].

One intuitive reason why these nearest neighbour algorithms fail to be efficient is the phenomenon of the interplay of hypersphere and hypercube. Measuring the distance to a given point p is a "unit sphere operation": reporting all points within a distance of at most, say, k to a given point p is the same as reporting all points within a sphere of radius k and centered at p, where the type of sphere depends on the underlying metric. For low dimensions now the d-cube and d-sphere are similar enough, so that algorithms based on spherical distances work well. But for higher dimensions the unit cube and unit sphere behave totally different. While the d-cube still has unit volume, its diameter increases arbitrarily with the dimension. On the other hand the diameter of the unit sphere remains unchanged, whereas its volume vanishes with increasing dimension. This different behavior in "distance" and "volume" makes it impossible to use distances related to the hypersphere to cover vertices and thus portions of the volume of the hypercube.

Note that this is not only true for the Euclidean distance, but for any L_p metric, especially the L_1-metric and thus the Hamming distance. E.g., the distance of an antipodal pair increases even linearly in the dimension for the L_1-metric, whereas it depends only with the square root in the Euclidean case.

To illustrate the situation consider the center point of the cube. If we generalize the Hamming distance to the L_1-metric, then the center point of the d-cube is at distance $\frac{d}{2}$ to the origin (which is quite natural, since it lies half way to the antipodal of the origin). If in 2-space we place the unit circle (with radius 1) at the origin it clearly contains the center point $(\frac{1}{2}, \frac{1}{2})$ of the unit square. This remains true for the unit ball and unit cube in 3D, but already in 4D the center point of the 4-cube lies on the surface of the 4-ball. Starting with dimension 5 the unit ball centered at the origin does not longer cover the center point of the unit cube. (Cf. this with the well known paradoxon of the decreasing volume of the unit ball for dimensions greater than 5.)

Bucketing Algorithms

A very simple algorithm that works well for uniformly distributed points in low dimensions is the bucketing algorithm, also called Elias' algorithm [112]. The space is divided into identical cells, storing all vertices inside a cell in a linear list. For a query point the cells are examined in increasing distance, until the minimum distance to a cell exceeds the distance to the closest point found.

An analysis and some experiments of Elias' algorithm can be found in [90]. In [27] this approach was rediscovered independently and also used for the hypercube case. But the experiments of the mentioned papers give only results for dimension 30 and less, and the approach seems to be inefficient for higher dimensions. Also the restriction to uniformly distributed points seems not to be appropriate for our problem, as will be shown in the next sections.

Local Search

Another attempt to the problem would be to use the well known "local search" technique which is used to solve many hard optimization problems. On a first glance our problem seems to have some nice properties for this approach. First we have no problem finding a starting point for the local search, since any vertex of our set C is a "feasible" solution. And second, since any step of the local search goes towards the minimum, it must decrease the Hamming distance to the query point by at least one. Therefore we know beforehand that the process of local search terminates after at most $O(d)$ iterations.

For the correctness of the local search approach we have to guarantee that, if we have not yet obtained the optimal solution, for any vertex $c \in C$ there is at least one local neighbour which decreases the Hamming distance to the query point. If this condition is fulfilled we clearly can not get stuck in a local optimum.

Obviously the edges of the convex hull of a point set fulfill these requirements. It is not hard to see that any point of C has to lie on the convex hull of C: consider the hypersphere S^d with center $(\frac{1}{2}, \ldots, \frac{1}{2})$ and radius $\frac{\sqrt{d}}{2}$. All vertices of $\mathcal{C}^d$ lie on S^d and thus any subset of V^d entirely consists of extreme vertices. Therefore computing the edge skeleton of the convex hull of C immediately gives us the required local neighbourhood relations for local search.

Unfortunately it turned out that the convex hull of a subset of hypercube vertices has a "complete-graph" like behavior. Figure 8.1 shows the average degree of the convex hull of a subset of vertices of the 10-cube opposed to the size of this set. Note that especially for small subsets, which are likely for higher dimensions, the degree increases linearly with the size. Thus for the interesting cases where $|V^d| = 2^d$ is much larger than the number n of vertices of C it can be seen that the convex hull of C is likely to be a 2-neighbourly polytope (cf. [114]) or a close approximation of it. That is, almost every pair of vertices in C is connected by an edge of the convex hull of C. For example there exists a 2-neighbourly polytope with 1035 edges and 46 vertices of $\mathcal{C}^{12}$.

Thus when starting with some arbitrary vertex, it is not only adjacent already with the optimal solution, but also to any other vertex of C. In this case there is only one iteration of local search, but it will already need $O(dn)$ time.

This observation is strongly related to a result of Füredi [41]. He showed that for a subset

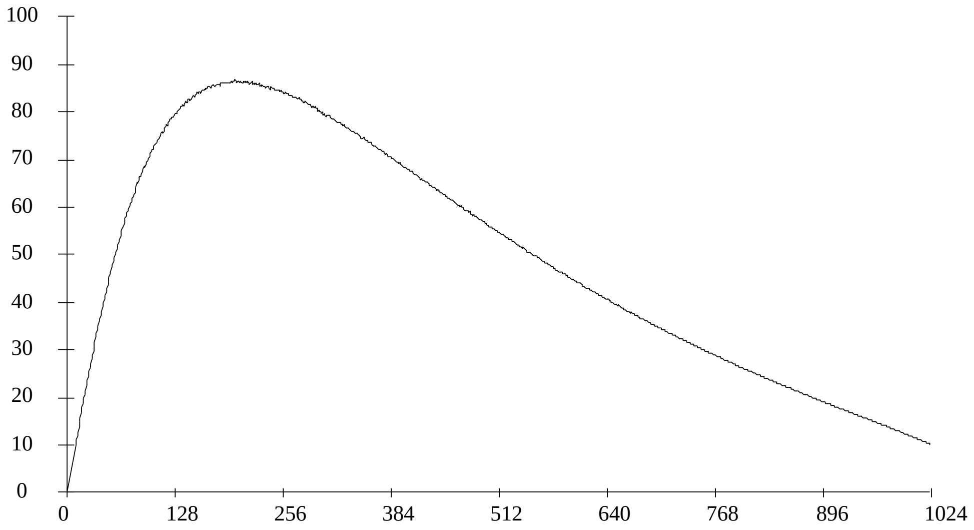


Figure 8.1: Average degree of the convex hull of a subset of n vertices of the 10-cube.

of vertices from $\mathcal{C}^d$ and for an arbitrary $\epsilon > 0$ the convex hull of more than $(2+\epsilon)d$ vertices almost always contains the center of the cube, while the convex hull of less than $(2-\epsilon)d$ points almost always avoids it. From linear programming it is easy to see that this is equivalent to the question whether the diagonal of a pair of vertices with Hamming distance k is part of the convex hull. If the k-dimensional subcube induced by these two vertices contains less than $2k$ vertices from the given set, it is likely that their convex hull does not contain the center of the subcube. The considered diagonal will then be part of the convex hull. Therefore it is reasonable that if the number of vertices is small, the edges of the convex hull will approximate the complete graph.

8.2 $\{0,1\}$-String Searching using Hierarchical Clustering

In this section we use a new approach to investigate the $\{0,1\}$-string search problem for a given set $S \subseteq V^d$, which is based on the structure of hierarchical cluster trees. An agglomerative hierarchical clustering method starts with n single objects, all seen as their own clusters. Step by step two specific clusters (objects at the beginning) are merged to a new cluster until, after $n-1$ steps, all objects form in a single final cluster, the root of the tree. Thus a hierarchical cluster tree is a family of clusters such that any two clusters are disjoint iff they lie on different paths (seen from the root), or one includes the other iff they are on the same path.

In order to utilize the cluster structure for our string searching we compute for any cluster C_i of the cluster tree of S a kernel c_i, which has to be an element of C_i. Depending on the method used to build the cluster tree these kernels may already have been computed during the clustering process. There are several methods to determine the kernel of a given point set and this clearly will effect the efficiency of the algorithms; we will give more details about that later on in this section.

Let us now give a description of our approach. For any query point $q \in V^d$ we start at the

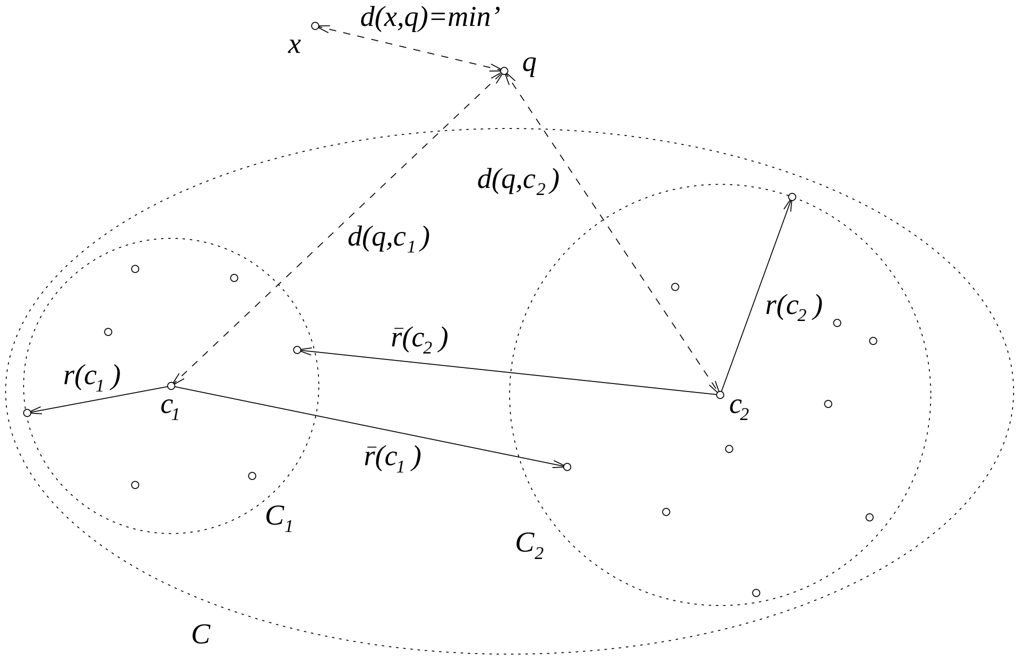


Figure 8.2: Relation of inter point distances for two subclusters C_1 and C_2 of C.

kernel of the whole set and then traverse the cluster tree in a depth-first-search like manner. For any branch of the cluster tree we consider the two subclusters and decide locally whether we have to search into them or not. That is whether or not they may contain vertices which are closer to the query point than the best solution found so far. Thus, by cutting off many of the branches, the algorithm may only consider a (hopefully small) subset of the given vertices of S in order to find the global optimum.

If a vertex $p \in S$ is the kernel of several clusters it may be considered more than once during the search task. Thus in addition to the cluster tree we store in a linear list whether the distance from the query point to a vertex $p \in S$ has already been computed, and, if yes, we store this distance. Note, however, that the overall number of kernels is $|S| - 1$ and that every vertex $p \in S$ has to be a kernel at least once, namely for the cluster only containing p.

In the worst case, if we have to search through all subclusters, we therefore have to consider every kernel, and thus every vertex of S, at most once. Thus the upper bound for the number of d-dimensional comparisons is $|S|$, the same bound as for the trivial approach.

We now give the notation used for our new approach, refer to Figure 8.2. Let $q \in V^d$ be the query point, $x \in S$ the best solution found so far, and let $min' = d_H(x, q)$ denote their Hamming distance. Initially we set x to the kernel of the whole set S. For any branch step during the search in S let C be the cluster under consideration and C_1 and C_2 the two subclusters of C, with kernels c_1 and c_2, respectively. With $r(c_i) = \max_{p \in C_i} d_H(c_i, p)$ we denote the radius of a cluster, and $\overline{r}(c_i) = \min_{p \in C \setminus C_i} d_H(c_i, p)$ is the minimum distance from c_i to a point from C outside the cluster C_i.

The decision whether or not to branch into a subcluster is now based on the following trivial, but useful observation.

Observation 8.1 *Let C be a cluster in the cluster tree of S, and let C_1 and C_2 the two subclusters of C, with kernels c_1 and c_2, respectively. Let $r(c_i)$ and $\overline{r}(c_i)$, $i = \{1, 2\}$, be defined as above. For all vertices $p_1 \in C_1$ we have*

(1a) $d_H(q, p_1) \geq d_H(q, c_1) - r(c_1)$ *and*
(1b) $d_H(q, p_1) \geq \overline{r}(c_2) - d_H(q, c_2)$.

Analogously for all vertices $p_2 \in C_2$ we have

(2a) $d_H(q, p_2) \geq d_H(q, c_2) - r(c_2)$ *and*
(2b) $d_H(q, p_2) \geq \overline{r}(c_1) - d_H(q, c_1)$.

Note that the observation does not require any special property of the hypercube, but only use the triangle inequality. Thus the method given in this section may be generalized for any high dimensional geometric closest point search problem.

Inequality (1a) of Observation 8.1 for subcluster C_1, and analogously (2a) for C_2, states that if the kernel c_1 of the cluster C_1 is too far away from the query point q, then no point in C_1 can achieve a better solution. Clearly this inequality will only provide good bounds if the radius $r(c_1)$ of the cluster is small enough.

The second inequality (1b) shows that if the kernel c_2 of C_2 is a rather good approximation to q and if the points of C_1 are far away from c_2, as we may expect for a good two-clustering of C, then again no point of C_1 can achieve a better solution. This again holds analogously for the subcluster C_2, shown by inequality (2b).

Now let

$$m(C_1) = max\,\{d_H(q, c_1) - r(c_1), \overline{r}(c_2) - d_H(q, c_2), 0\}$$
$$m(C_2) = max\,\{d_H(q, c_2) - r(c_2), \overline{r}(c_1) - d_H(q, c_1), 0\}$$

From Observation 8.1 it follows that $m(C_i)$ is a lower bound for the Hamming distance $d_H(p, q)$ for any point $p \in C_i$ to the query point q. We thus get

Corollary 8.1 *Let C_1, C_2, and min' as defined above. Then C_i, $i = \{1, 2\}$, may contain a new optimal vertex only if $m(C_i) < min'$.*

Corollary 8.1 provides a simple, local criterion whether or not to branch into a subcluster of the cluster tree while searching for an optimum vertex. The use of the relations of distances in our approach is in some sense similar to what AESA [108] and IFAP-AESA [88] do. But because of using the cluster hierarchy we are not limited to a fixed or randomly chosen subset of "anchor points". Instead, we use a set of kernels which reflects the structure of the given data. We thus expect our approach to be adaptive to specially structured vertex sets and thus to be efficient in practice.

8.2.1 Preprocessing

We are now ready to consider algorithmic issues. To generate the cluster tree T of a vertex set $S \subseteq V^d$, $n = |S|$, we utilize the hierarchical clustering methods presented in Chapter 7. The output of the algorithms given there is the full cluster tree T: a linear list gives the indices of the $n-1$ agglomeration steps in a bottom up like manner, from single objects (vertices) to the final cluster, the root of the tree. In addition, for clustering methods based on the kernels of subcluster, these kernels are also given for each cluster (e.g. for KRNLSUM, see Section 7.2.5 in Chapter 7). Thus we might also use this kernel information for precomputation. But since most hierarchical clustering methods are not based on kernels, we give here a general scheme for precomputing all necessary informations from a given cluster tree T.

From Corollary 8.1 we know that we have to precompute c_i, $r(c_i)$, and $\overline{r}(c_i)$ for every cluster C_i which is part of T. We will compute the kernel c_i in a way such that it minimizes the radius of C_i, i.e. $r(c_i) = \min_{c \in C_i} \{\max_{p \in C_i} d_H(c,p)\}$. Considering inequalities (1a) and (2a) of Observation 8.1 this clearly makes sense, since it locally maximizes the lower bound. We thus use a similar approach as was given in Chapter 7 for hierarchical clustering minimizing the radius of a cluster.

We now show that for an agglomeration step $C = C_1 \cup C_2$ in the cluster tree T we can compute the kernel c of C in $O(d|C_1| \cdot |C_2|)$ time. To this end for every vertex p of S we dynamically store the maximum distance to any other point contained in the same cluster as p. This requires only $O(n)$ additional space for the whole cluster tree. Initially, for single vertices, these maximum distances are set to zero. The centroid point of a cluster C can thus be obtained in $O(|C|)$ time by taking the point with the minimum value.

The kernel of the union of C_1 and C_2 now can be computed in $O(d|C_1| \cdot |C_2|)$ time: For any point of C_1 (and symmetrically for C_2) we have to compute the maximum distance to a point in C_2 (C_1, respectively) and store the maximum of this distance and the previous maximum of the point. The vertex c obtaining the minimum of these maxima is the kernel of C and also determines $r(c)$. Moreover for c_1 (c_2) we compute at the same time the minimum distance to a point in C_2 (C_1), which gives $\overline{r}(c_1)$ and $\overline{r}(c_2)$, respectively.

To get all required information we reiterate the above agglomeration step bottom up for every cluster of T. Since thereby any pair of vertices is considered at most once, we have $O(n^2)$ distance computations and get:

Lemma 8.1 *Given the cluster tree T of a vertex set $S \subseteq V^d$, $n = |S|$. The computation of c_i, $r(c_i)$, and $\overline{r}(c_i)$ for every cluster C_i of T can be done in $O(dn^2)$ time and linear space.*

8.2.2 The Algorithms

From the last two subsections we now immediately get the first $\{0,1\}$-string search algorithm SEARCH1 shown in Figure 8.3. Thereby $S \subseteq V^d$ is the vertex set, $q \in V^d$ the query point, and T is the cluster tree of S including the additional information computed in the preprocessing phase.

The correctness of SEARCH1 follows from Corollary 8.1 and the explanations given above. Note that within the procedure DEPTH-FIRST-SEARCH we do not need to consider whether a cluster has subclusters or not. A cluster C without subclusters consists of a single vertex c (its kernel) and has therefore a radius $r(C) = 0$. Thus $m(C) \geq d_H(c,q) \geq min'$ which means that for such a cluster there will not be a recursive call.

Using the linear list mentioned in the above discussion it is straight forward to see that SEARCH 1 runs in time linear in the number k of visited kernels, that is, in time $O(dk)$. Thus, to improve the running time, we have to minimize the number of necessary subcluster branches.

Algorithm SEARCH1(S,T,q)

1. Let C_r be the final cluster of S (root of T) and c_r its kernel
2. Set $min' = d_H(c_r, q)$ and $best = c_r$
3. Call DEPTH-FIRST-SEARCH(C_r)
4. Output $best$

Procedure DEPTH-FIRST-SEARCH(C)

1. Let C_1 and C_2 be the two subclusters of C in T, with kernels c_1 and c_2, respectively
2. **FOR** i = 1 to 2 **DO**
   (a) **IF** $d_H(c_i, q) < min'$ **THEN**
      - $min' = d_H(c_i, q)$
      - $best = c_i$
3. Compute $m(C_1)$ and $m(C_2)$
4. **IF** $m(C_1) < min'$ **THEN** call DEPTH-FIRST-SEARCH(C_1)
5. **IF** $m(C_2) < min'$ **THEN** call DEPTH-FIRST-SEARCH(C_2)
6. Return

Figure 8.3: SEARCH1: A first approach to $\{0,1\}$-string searching using hierarchical clustering.

Observe that after the first recursive call in step 4 of DEPTH-FIRST-SEARCH min' might have changed and thus the call of step 5 might be unnecessary even if $m(C_2) < min'$ was true before step 4. In other words, the sequence of recursive calls might effect the number of vertices we have to consider. Thus a heuristic should try to consider 'promising candidates' first. Instead of doing an inflexible depth first search we can sort the subclusters by a certain criterion and then branch into the "best" one. The hope is to rapidly find a good approximation of the optimum and thus to minimize min', such that we can avoid to consider most of the remaining subclusters.

This observations lead to Algorithm SEARCH2 given in Figure 8.4. For the heuristic we use the Hamming distance from the kernel c_i of a cluster C_i to the query point q. Clearly if the kernel is close to q then we might expect other vertices of C also to be close to q. On the other hand if the kernel is far away, there is little hope to find a"good" vertex within C.

To organize the subclusters we use a simple heap sorted by the distances $d_H(c_i, q)$ of the kernel points c_i to the query point q. Clearly only subclusters C with $m(C) < min'$ are put onto the heap and the algorithm iterates until the heap is empty. Thus all kernels of appropriate clusters are considered and the correctness of this approach follows.

In fact we can also use d linked lists, since $d_H(c_i, q) \in \{0, \ldots, d\}$. Inserting an element into the heap (put) or removing the minimum from it (get) can therefore be done in time $O(d)$. Thus, if k is again the number of kernels to be considered, the whole algorithm runs

---

Algorithm SEARCH2(S,T,q)

1. Let C_r be the final cluster of S (root of T) and c_r its kernel
2. Set $min' = d_H(c_r, q)$ and $best = c_r$
3. PUT(C_r,$d_H(c_r, q)$)
4. **WHILE** heap not empty **DO**
   (a) GET(C,d)
   (b) Let C_1 and C_2 be the two subclusters of C in T, with kernels c_1 and c_2, respectively
   (c) **FOR** i = 1 to 2 **DO**
      i. **IF** $d_H(c_i, q) < min'$ **THEN**
         - $min' = d_H(c_i, q)$
         - $best = c_i$
      ii. **IF** $m(C_i) < min'$ **THEN** PUT($C_i, d_H(c_i, q)$)
5. Output *best*

Figure 8.4: SEARCH2: Using a heap heuristic for subclusters.

in space and time $O(dk)$. Note that, by the same argumentation as given for the procedure DEPTH-FIRST-SEARCH, clusters without subclusters will not be put onto the heap.

Alternatively we might use the lower distance bounds $m(C_i)$ to sort the heap. This again reflects the idea behind the computation of these bounds. It is natural to expect a cluster with a small lower bound to be a promising candidate which should be considered first. We thus give a second heap based version, Algorithm SEARCH3, in Figure 8.5. Again we have $m(C_i) \in \{0, \ldots, d\}$ and thus the above given space and time bounds still hold.

SEARCH3 has the additional advantage that after getting a cluster C from the heap we can check in constant time whether $m(C) < min'$ still holds. If this it not the case then we are done, since all remaining clusters on the heap have an even larger lower bound (step 4b in Figure 8.5). Thus we expect SEARCH3 to be the fastest string search algorithm given in this section, which will be confirmed by the experimental results of Sec. 8.3.

For the running time of all three presented algorithms we have

Theorem 8.1 *Algorithms SEARCH1, SEARCH2, and SEARCH3 run in $O(dk)$ space and time, where k is the number of vertices to be considered.*

8.3 Experimental Results

In this section we give experimental results on the number of necessary distance computations. We compare the three different search algorithms (Subsection 8.3.1) and various hierarchical clustering methods from Chapter 7 (Subsection 8.3.2). Finally we give results for different kinds of special input sets (Subsection 8.3.3).

Algorithm SEARCH3(S,T,q)

1. Let C_r be the final cluster of S (root of T) and c_r its kernel
2. Set $min' = d_H(c_r, q)$ and $best = c_r$
3. PUT(C_r,0)
4. **WHILE** heap not empty **DO**
   (a) GET(C,$m(C)$)
   (b) **IF** $m(C) \geq min'$ **THEN** goto 5
   (c) Let C_1 and C_2 be the two subclusters of C in T, with kernels c_1 and c_2, respectively
   (d) **FOR** i = 1 to 2 **DO**
      i. **IF** $d_H(c_i, q) < min'$ **THEN**
         - $min' = d_H(c_i, q)$
         - $best = c_i$
      ii. **IF** $m(C_i) < min'$ **THEN** PUT($C_i, m(C_i)$)
5. Output *best*

Figure 8.5: SEARCH3: Using a heap controlled by $m(C_i)$.

The diagram of Figure 8.6 shows the relation between the different programs and the data. From the vertex set, which comes from an application or is generated for test purposes, we get the cluster tree T by one of the various hierarchical clustering methods given in Chapter 7. In the preprocessing phase we then compute c, $r(c)$ and $\overline{r}(c)$ for every cluster $C \in T$. Now we can apply one of our search algorithms and treat the query vertices, given either by the application, or specially generated from the input set or taken at random from V^d.

The generation of vertex sets as well as the generation of an appropriate set of query vertices is only needed for test purposes and thus drawn with dashed lines in Fig. 8.6. That we indeed need to generate query vertices instead of performing random queries can be seen from the fact that for higher dimensions (e.g. $d \geq 100$) the expected Hamming distance between a random query vertex and its optimal approximation typically lies in the range $\{\frac{d}{3}, \ldots, \frac{d}{2}\}$ if the number of vertices is small compared to 2^d. Thus the optimal solution would not be significantly better than a random answer, which is expected to have Hamming distance $\frac{d}{2}$. We therefore always generated a query vertex set at random, but with a fixed Hamming distance in the range $\{0, \ldots, \frac{d}{2}\}$ to some (again randomly chosen) vertex of the input set.

For every search we give the number of considered vertices opposed to the Hamming distance between the query vertex and its optimal solution. In fact it turns out that this is a very distinctive relation. Note that for a query vertex chosen at Hamming distance about $\frac{d}{2}$ from "its" input vertex, there might be a "better" vertex in this set, which we will get as the optimal output. Thus the given curves will most likely end before achieving $\frac{d}{2}$.

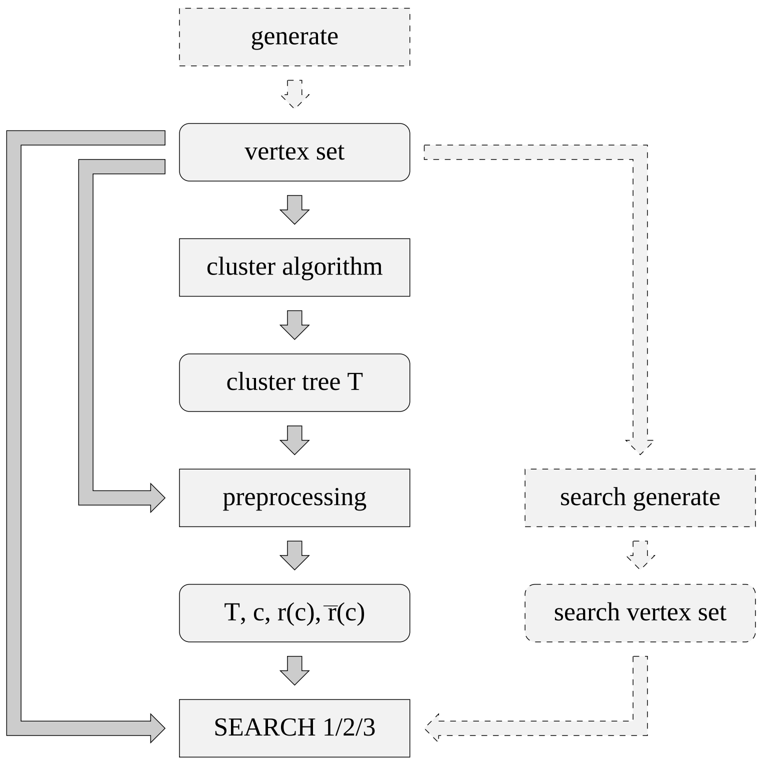


Figure 8.6: Relation between algorithms (rectangular boxes) and data (boxes with rounded corners). Dashed arrows and boxes are only for test purposes.

8.3.1 Comparison of Search Algorithms

In this subsection we compare the three search algorithms of Section 8.2.2 for two different input sets. Both sets consist of $n = 10000$ vertices of dimension $d = 512$. For the first set the vertices are chosen uniformly at random from V^d. Then these sets are clustered by Algorithm MINRADI (cf. Chapter 7) in order to minimize the radius of the clusters. (For a short description of the cluster methods see Sec. 8.3.2.) We thus expect especially inequalities (1a) and (2a) of Observation 8.1 to take effect.

For the second set we generated the vertices around a fixed number k of centers, such that the resulting sets are structured sphere-like around each center, similar to the special sets used in Chapter 7. Note, however, that the centers need not to belong to the set. We used $k = 10$ centers and required the Hamming distance between two centers to be $\geq \frac{d}{2}$. The points were uniformly distributed around these centers with the limitation that the Hamming distance to the corresponding center must not exceed $\frac{d}{8}$. This forces of course a very special structure of the resulting cluster trees and we will see that in fact the experimental results are quite different compared to those for random sets.

We believe that this kind of special vertex sets give a more realistic representation of real word problems than random sets do. For applications there are most likely several groups of very similar "objects", and the object for which the search is done might belong to one of these groups. In fact, not the optimal object, but the group the query object belongs to, might be the required answer.

To cluster these special sets we used a complete linkage clustering algorithm (CLINK1 in Chapter 7).

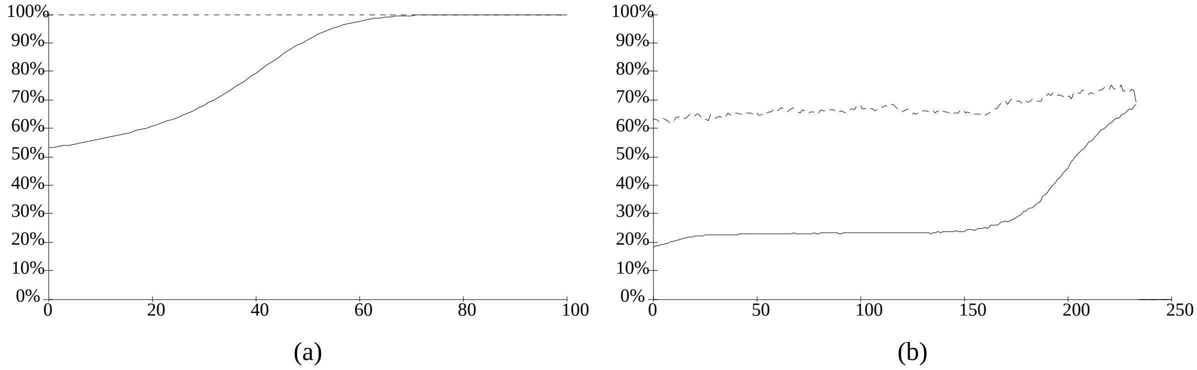


Figure 8.7: SEARCH1: Percentage of considered vertices against the Hamming distance between query vertex and optimal solution. Solid lines give the average case, dashed lines the worst case for over 100000 queries. (a) uniformly distributed random vertices; (b) special set with 10 cluster centers; Both sets with $n = 10000$ vertices of dimension $d = 512$.

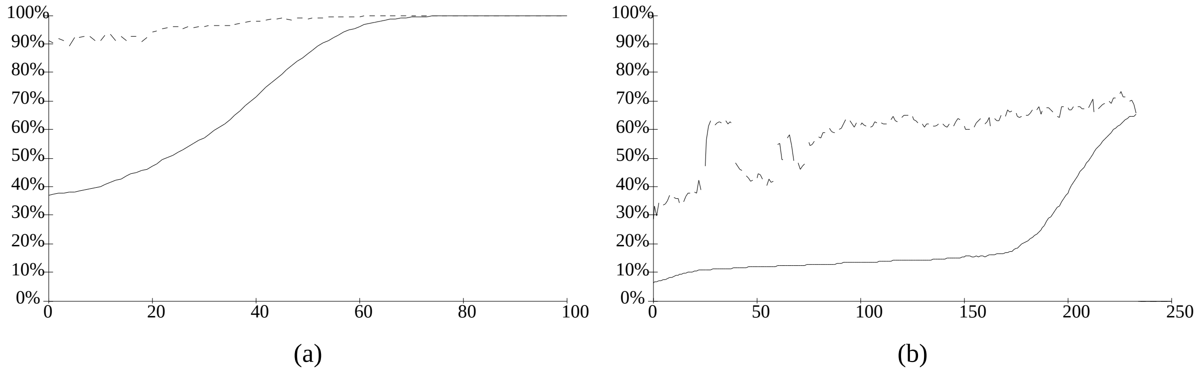


Figure 8.8: Analogous curves as in Fig. 8.7 for Algorithm SEARCH2.

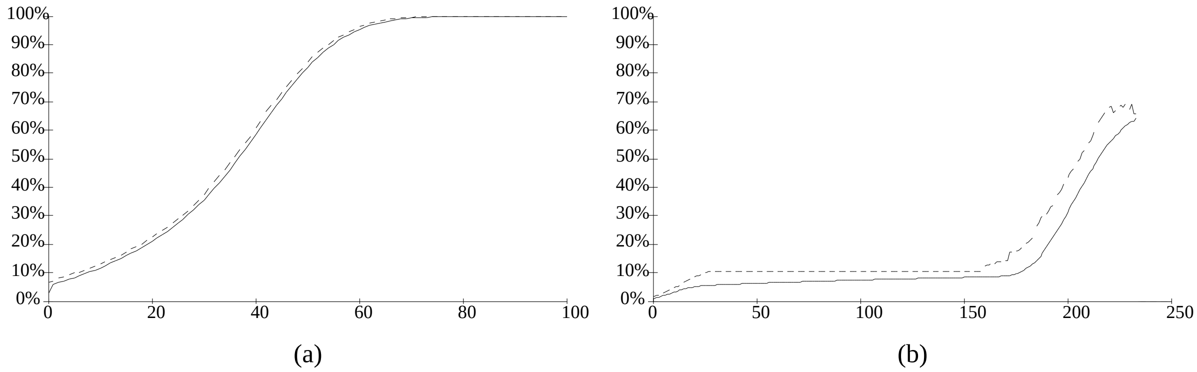


Figure 8.9: Analogous curves as in Fig. 8.7 for Algorithm SEARCH3.

Figures 8.7(a) to 8.9(b) give the results for over 100000 queries each, for Algorithms SEARCH1, SEARCH2, and SEARCH3, respectively.

For the random set the curves show that only for a small Hamming distance from the optimum we can avoid a reasonable number of comparisons. If the distance is too large (typically greater than $\frac{d}{8}$) we have to consider nearly 100% of the vertex set. Despite the fact that for the average case all three algorithms behave quite similar, SEARCH3 is obviously the most efficient one. If we consider the worst case behavior this distinction is very conspicuous. While for SEARCH1 there is always an example which requires at least 99% of the distance computations and also SEARCH2 needs 90% or more, the worst case of SEARCH3 lies only about $2-3\%$ above the average case. This clearly shows that for random vertex sets SEARCH3 is the recommendable algorithm.

For the mentioned special sets the situation is more pleasant for all three algorithms. This is especially true if the Hamming distance to the optimum vertex is small enough to quickly find the "cluster ball" where the query vertex belongs to (in our case below approx. $\frac{d}{3}$).

Again SEARCH3 is clearly the best algorithm, having the average and worst case both below the average case of the other algorithms. For query points which have a very close optimal solution ("near hit") we have to consider only about $2-3\%$ of all vertices. Thus the running time of the search algorithm is output sensitive in this sense. Note that the worst case of SEARCH3 is again very close to the average case, and lies around 10% for Hamming distances between 20 and 150. This reflects the size of the "cluster balls" we have generated for the special sets. If we would use more and thus smaller groups this would improve the worst case. For more details on these effects see Section 8.3.3.

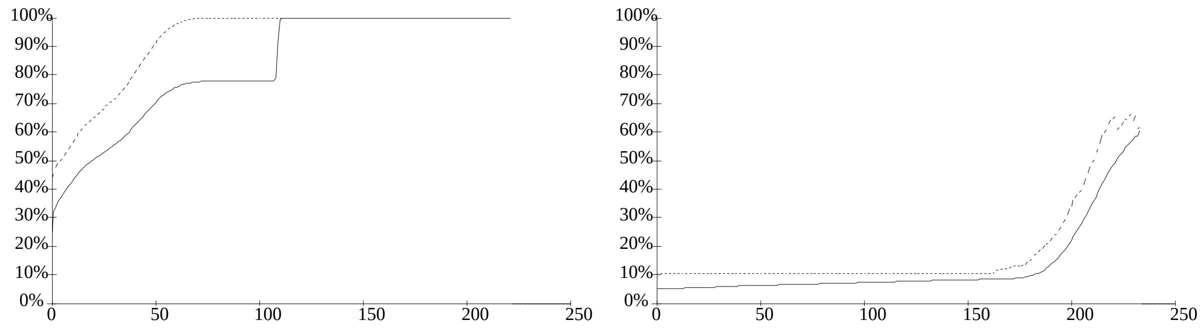
We conclude that SEARCH3 is the most promising search algorithm and in fact we will perform all experiments of the remaining subsections with this algorithm only.

8.3.2 Impact of Different Clustering Methods

In this section we examine the impact of the different hierarchical clustering methods of Chapter 7 for the Algorithm SEARCH3. To this end we consider the same vertex sets, random and special, as in Section 8.3.1. All hierarchical clustering methods mentioned in Chapter 7 were examined and we will use the same notation given there.

Let us first give a brief description of the algorithms we will mention explicitly. SLINK computes the well known single linkage clustering (S-link) which was first suggested by Florek et al. [40] and later on by various other authors. The first efficient general implementation of S-link was given by Sibson [98]. CLINK1 computes the complete linkage clustering (C-link) which minimizes the diameter of the subclusters, and was first mentioned by Sørensen [105]. MINRADI is a centroidal clustering method and minimizes the radius of the cluster. The centroid of the cluster has to be contained in the cluster and is thus called its kernel. All these methods are agglomerative, i.e. build the cluster tree bottom up. In contrast, the two algorithms DIVIDIAM and DIVIDIA2 are divisive hierarchical cluster methods which locally minimize the cluster diameters in a top down like manner. While DIVIDIAM stops for one division step after determining the minimum diameter of the subclusters, DIVIDIA2 uses a post-heuristic to improve the quality of the resulting clusters. For our use of the cluster trees all other hierarchical methods of Chapter 7 turned out be very similar to some of the above methods and thus we will not mention them explicitly.

For random point sets all clustering methods except SLINK gave very similar results. Figure 8.9(a) gives the typical curves for MINRADI which was the best of all methods. The

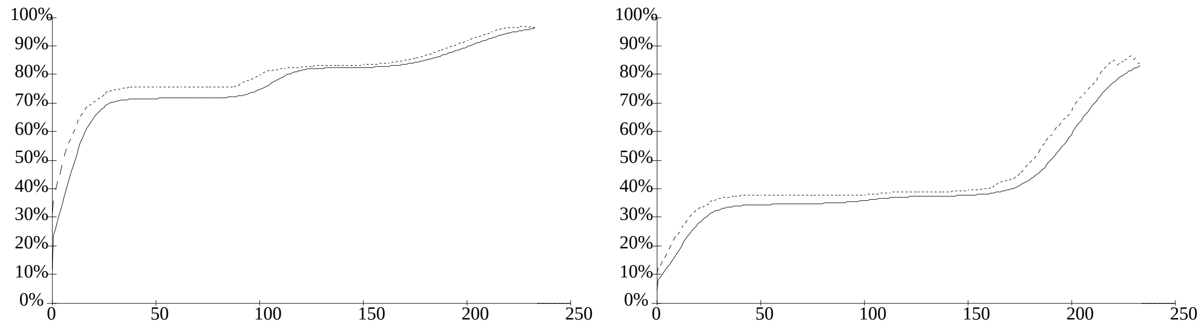


(a) (b)

Figure 8.10: Percentage of considered vertices (analogous Fig. 8.7) for Algorithm SEARCH3 and hierarchical cluster method S-link.

curves of the other methods, except SLINK, all have the same structure, but are slightly varying in absolute values. For example the two divisive methods DIVIDIAM and DIVIDIA2 are the worst.

Figure 8.10(a) shows the curve for S-link which is quite different. Despite the fact that for small Hamming distances there are much more vertices to consider, it has the best average behavior for a Hamming distance between 50 and 100. An intuitive explanation of this characteristic and the sharp bend for a Hamming distance of around 110 bit may be the fact that S-link is a subgraph of the minimum spanning tree. Thus for small clusters the chaining effect of S-link prevents $r(c_i)$ to be sufficiently small. For larger clusters S-link maximizes the inter cluster distance and thus probably also $\overline{r}(c_i)$. Therefore inequalities (1b) and (2b) of Observation 8.1 may take effect. But when the Hamming distance exceeds half the length of the longest edges of the minimum spanning tree, the query vertex lies somewhere in between two kernels and thus no subclusters can be skipped. Clearly this explanations only hold for the average case and thus there is a big gap between average and worst case, which has not been observed for any other cluster method.



(a) (b)

Figure 8.11: Percentage of considered vertices using SEARCH3 for special vertex sets. The cluster tree was build by the two divisive algorithms (a) DIVIDIAM, and (b) DIVIDIA2.

For the mentioned special sets there is a clear distinction between agglomerative and divisive hierarchical cluster methods. While agglomerative methods all showed the very pleasant behavior discussed in the last section, the two divisive methods DIVIDIAM (cf. Fig. 8.11(a)) and DIVIDIA2 (cf. Fig. 8.11(b)) poorly support our search algorithms.

All remaining methods have curves similar to the curve of S-link given in Figure 8.10(b). Only if the Hamming distance is rather small, i.e. less than 5% of the dimension, then C-link showed an interesting variation, see Figure 8.9(b). For query vertices which have a very similar optimal vertex, SEARCH3 needs only to consider $2-3\%$ of the whole vertex set. If the query even gives a “perfect hit” (Hamming distance 0) then only approx. 0.8% of the vertices have to be considered in the average. For the first 5% of the curve this relation seems to increase nearly linear for C-link, see also the next section.

We conclude that there are three especially suitable hierarchical cluster methods. The first is MINRADI which is the best for random vertex sets and behaves well for special sets, too. The second is C-link which is the best for special sets and “near-hit” queries, but may also be used for random sets. S-link is suitable for special sets and also for random vertices, if the queries are expected to be “not to close” to some vertex of the set. The additional advantage of S-link is its simple and thus efficient computation, cf. Chapter 7.

8.3.3 Further Investigations

In addition to the experiments of the last to subsections we investigated a large number of varying vertex sets. In this subsection we summarize the most interesting observations.

To strengthen the results for our special vertex sets with 10 cluster centers we varied the radius of the “cluster balls” and the minimum distance between the cluster centers. Figure 8.12 shows the results for a set of $n = 5000$ vertices of dimension $d = 256$ clustered by S-link. For part (a) the distance between the cluster centers is at least $\frac{d}{2}$ and the radius of the cluster balls around each center varies from $\frac{d}{8}$ to $\frac{d}{3}$. Clearly the average number of vertex comparisons increases with increasing radius, but even for a radius of $\frac{d}{3}$ the results are still satisfactory. Note that in this case there are already vertices which might belong to different cluster centers, i.e. the intersection of the cluster balls is not longer empty.

For Figure 8.12(b) we fixed the radius of the cluster balls with $\frac{d}{8}$, and the minimum distance between the cluster centers varied from $\frac{d}{5}$ to $\frac{d}{2}$. The resulting four curves show only minor differences. This reflects the fact that by choosing the cluster centers randomly the pairwise distance is rarely much less than $\frac{d}{2}$.

Next we considered a very large set with $n = 100000$ vertices of dimension $d = 512$ and $k = 16$ cluster centers. We used S-link to cluster this set and it can be seen from Figure 8.13(a) that Algorithm SEARCH3 still performs excellently for such large sets. Again the worst case roughly reflects the cardinality of the cluster balls.

This immediately leads to an idea for another special vertex set. We randomly chose 10 cluster centers with minimum pairwise distance $\frac{d}{2}$. Within a radius of $\frac{d}{8}$ around each of these centers we again chose 10 subcluster centers. And finally we put a third layer of sub-subcluster centers within a radius of $\frac{d}{16}$ around each subcluster center. Note, however, that all these centers do not yet belong to our vertex set. To generate this set we took 10000 vertices uniformly distributed within a radius of $\frac{d}{32}$ around the sub-subcluster centers. To cluster this set we used C-link. Clearly the special structure of these points is very well suited for clustering and thus we expect it to be a very useful set for Algorithm SEARCH3.

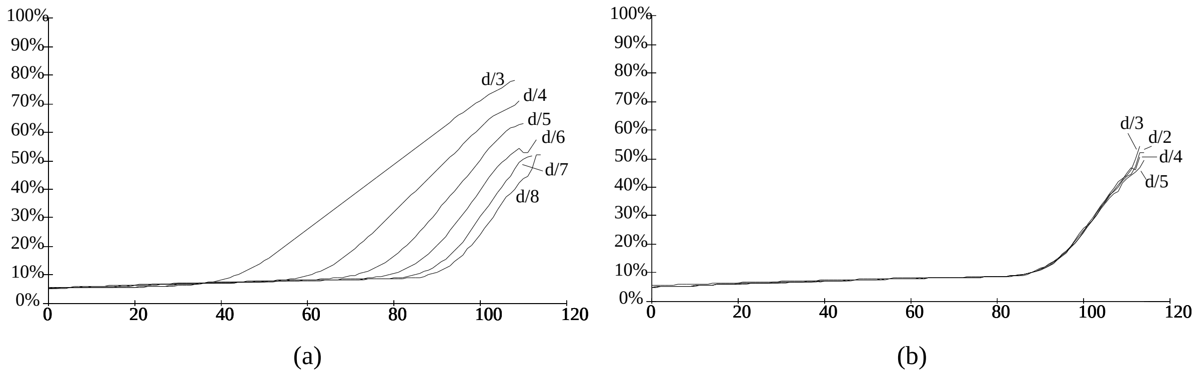


Figure 8.12: Average percentage of considered vertices using SEARCH3 and S-link for special vertex sets with 10 "cluster balls", $d = 256$ and $n = 5000$. (a) Distance between cluster centers $\geq \frac{d}{2}$ and radius of cluster balls varying from $\frac{d}{8}$ to $\frac{d}{3}$. (b) Minimum distance between cluster centers varying from $\frac{d}{5}$ to $\frac{d}{2}$ with fixed radius $\frac{d}{8}$.

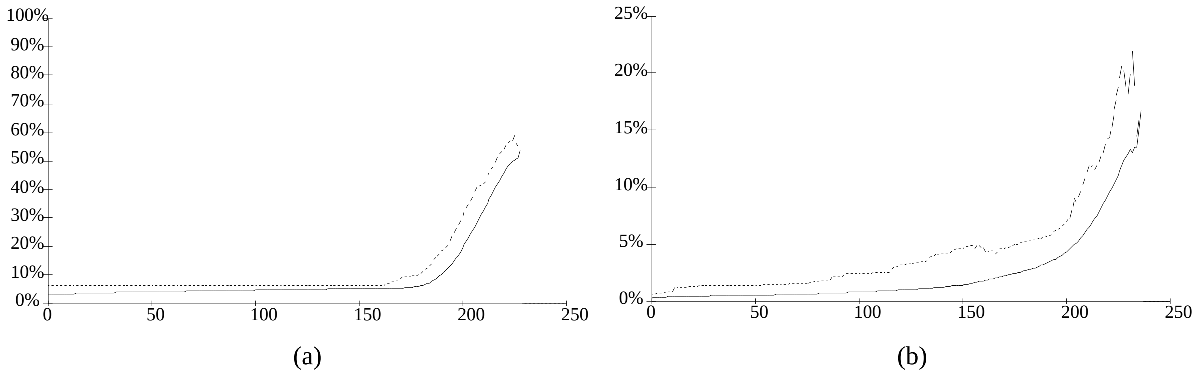


Figure 8.13: Percentage of considered vertices for SEARCH3. (a) Large set with $n = 100000$ vertices and $k = 16$ "cluster balls", $d = 512$, clustered with S-link. (b) Special set with three layers of 10 kernels each, $n = 10000$, $d = 512$, clustered with C-link.

Figure 8.13(b) shows that this is indeed the case. Note the different scaling of the y-axis. In the average case we have to consider less than 0.8% of the vertex set, if the Hamming distance between query vertex and best vertex is less than 100 (which seems to be quite large). If the query is a "near hit" (say at most 4 bit difference) there are about $\frac{1}{3}\%$ ($\frac{2}{3}\%$) of the vertices to be considered in the average (worst) case.

Clearly we do not expect this set to appear in real world problems, but it points out the capability of our approach to adapt to specially structured vertex sets.

For the vertex sets of Section 8.3.1, Figure 8.14 gives the results for "near hit" queries of up to 25 bit difference (about 5% of the dimension). As already mentioned in the last sections this shows the good performance of SEARCH3 for this kind of queries. The random set was clustered by MINRADI and the special set was clustered by CLINK1. In the last section these two methods turned out to be most suitable for the given sets.

For completeness we also considered lower dimensional vertex sets. Figure 8.15 gives examples for two random sets with $d = 16$, $n = 200$, 5000, 30000, and $d = 32$, $n = 200$, 5000, 20000, respectively. From these curves we see that if the number of vertices is rather small

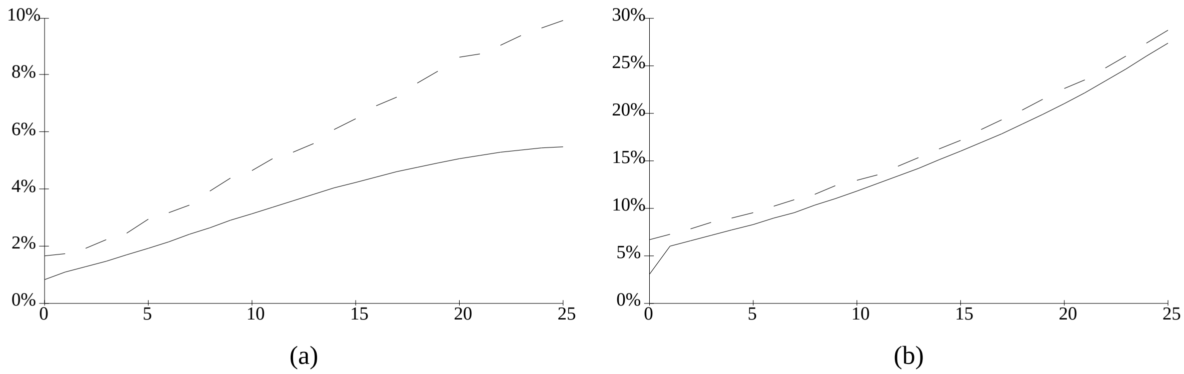


Figure 8.14: Percentage of considered vertices for SEARCH3 for "near hit" queries, $n = 10000$, $d = 512$. (a) Special set with 10 cluster balls, clustered with CLINK1. (b) Uniformly distributed random vertices, clustered with MINRADI.

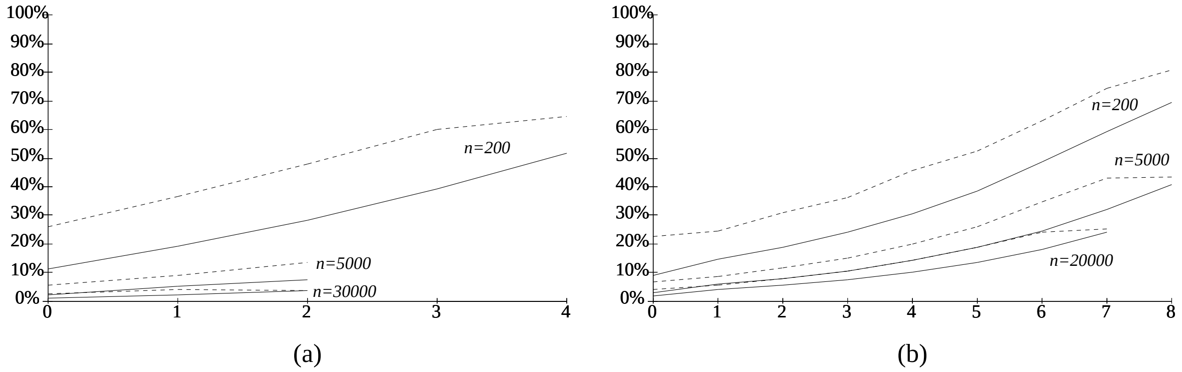


Figure 8.15: Percentage of considered vertices for SEARCH3 for low dimensional random vertex sets. (a) $d = 16$, $n = 200$, 5000, 30000 clustered with CLINK1, and (b) $d = 32$, $n = 200$, 5000, 20000 clustered with MINRADI.

then the percentage of considered vertices is higher, but the absolute number of vertices is still small. Contrarily if the number of vertices is large then SEARCH3 terminates very quickly. For example consider the set with $d = 16$ and $n = 30000$, i.e. nearly half of the vertices of $\mathcal{C}^{16}$ are in the set. Then the optimum solution most likely has a Hamming distance of 0 or 1 (only 3 out of 100000 queries had a Hamming distance of 2). The experiments showed that in the average case less than 1.45% of the vertices of this set had to be considered.

Considering the hierarchical structure of the cluster trees, a question which suggests itself is whether we can utilize this for very huge vertex sets which do not entirely fit into the main memory. The obvious approach is to keep only the "upper" part of the cluster tree in memory. To this end we indicate every kernel by its level in the tree, where, as usual, the level of a vertex is its distance to the root. We then keep all vertices of level less than or equal to some fixed k in memory. If we start to search for a vertex then the first comparisons can obviously be done in memory. The hope is now that the search can skip most subtrees and thus avoids to branch into deeper levels.

Unfortunately, experiments showed that this approach does not really work. Figure 8.16 gives the percentage of internal comparisons (solid line) and of vertices be kept in memory

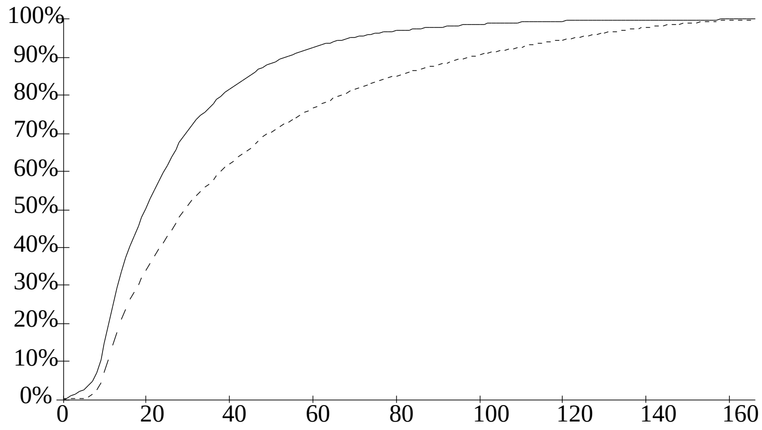


Figure 8.16: Percentage of internal comparisons (solid line) and of internal vertices (dashed line) over the number of stored levels.

(dashed line) against the number k of levels stored in the main memory. We used a set with $n = 10000$ vertices of dimension $d = 512$ and 10 kernels, clustered by C-link. The resulting cluster tree had 166 levels. The queries where chosen to guarantee an optimal Hamming distance in the range $\{0, \ldots, 25\}$. The curves show that the number of internal comparisons for the mentioned approach is at best only about 20% larger than the share of vertices in memory. For arbitrary query vertices the situation would be even worse, and for random point sets the two curves for internal comparisons and the number of vertices be kept in memory are almost identical.

8.4 Conclusion and Open Problems

In this chapter we gave a new approach to the $\{0,1\}$-string search problem, based on hierarchical clustering. We gave simple observations on the relation of intra- and inter-cluster distances and showed how to utilize them for string searching. In Section 8.2.2 we presented three different search algorithms which were thoroughly examined in Section 8.3. The experiments showed that Algorithm SEARCH3 is the most promising method and Section 8.3.2 gives an overview of suitable hierarchical clustering methods. Moreover the curves of Section 8.3 show that SEARCH3 performs quite well if either the distance between the query vertex and the best solution is not too large or the vertex set consists of "well grouped" vertices. In both cases we only need to consider a small percentage of the vertex set.

There are two open questions related to the presented approach:

- Is there a search algorithm which performs similar to SEARCH3 but needs less preprocessing? If there is a large number of queries then the preprocessing for SEARCH3 is acceptable. But if there is only a limited number of queries then especially the hierarchical clustering methods might be too time consuming.
- Does there exists an algorithm for the $\{0,1\}$-string search problem which is guaranteed to run in $o(dn)$ time and $O(dn)$ space for any vertex distribution and any query vertex, similar to what binary bisection search does in the one dimensional case?

Chapter 9

Miscellaneous

In this chapter we give a brief overview on the literature of hypercube related problems which are not mentioned in the previous chapters. Of course this task has to be incomplete and we thus restrict the list to a small number of prominent problems. In Section 9.3 we will also give a short list of hypercube related NP-complete problems, most already cited in earlier chapters. We then conclude the thesis by pointing out that even for the 2-dimensional hypercube (the square) there are simple-looking open problems.

9.1 Triangulating the Hypercube

In this section we consider the following question: What is the smallest number $\Delta(d)$ of d-simplices that is sufficient to triangulate $\mathcal{C}^d$? Note that the simplices are spanned by the vertices of $\mathcal{C}^d$, i.e. we are not allowed to introduce new vertices. The triangulation is thus often called *vertex preserving*. Moreover we only consider facet-to-facet triangulations, i.e. the facets of the simplices are not intersected by edges of other simplices. Triangulations of $\mathcal{C}^d$ with few simplices are for example used in finite element methods for solving differential equations, see [107] for details. Moreover they are of interest for their role in complementary pivoting algorithms used to approximate fixed points of continuous mappings and other optimization problems, see e.g. [70, 100] or Section 8.4.2 in [46] and references therein.

The well known standard triangulation of $\mathcal{C}^d$ consists of $d!$ simplices, all having the origin in common, see e.g. [22]. Thus the "base" of every such simplex lies on one of the d opposite facets of $\mathcal{C}^d$. Since these bases have to build triangulations of the facets it follows by recursion that $\Delta(d) = d\Delta(d-1)$ and thus $\Delta(d) = d!$ for the standard triangulation. That this is also an upper bound can be seen from the fact that the volume of a simplex spanned by vertices of $\mathcal{C}^d$ has to be at least $\frac{1}{d!}$.

On the other hand it is easy to verify that $\mathcal{C}^3$ can be triangulated with less than $3! = 6$ simplices when starting with the "central" equilateral simplex spanned by four vertices of $\mathcal{C}^3$ with pairwise Hamming distance two. This simplex has volume $\frac{1}{3}$ and leads thus to a triangulation of $\mathcal{C}^3$ with 5 simplices. Several other constructions were given for (small) constant dimensions, see e.g. [24, 31, 56, 58, 59, 70].

Sallee [95] gave an method of triangulating $\mathcal{C}^d$ using $O(d^{-2}d!)$ simplices. This upper bound was recently improved by Haiman [49]. He gave a very simple method achieving $O(\rho^d d!)$

$\Delta(1) = 1$		$\Delta(6) = 308$
$\Delta(2) = 2$		$\Delta(7) = 1\ 493$
$\Delta(3) = 5$	$5\ 522 \leq$	$\Delta(8) \leq 11\ 944$
$\Delta(4) = 16$	$26\ 593 \leq$	$\Delta(9) \leq 105\ 341$
$\Delta(5) = 67$	$131\ 269 \leq$	$\Delta(10) \leq 895\ 800$

Table 9.1: $\Delta(d)$ for $d = 1, \ldots, 10$.

simplices, where $\rho < 1$ is a constant. The basic idea is a decomposition of the triangulation to lower dimensional polytopes. He therefore showed that an improvement on the number of simplices for a triangulation of $\mathcal{C}^d$ for some fixed d leads to an asymptotical improvement for higher dimensions, i.e. reduces the value of ρ. The value of ρ given in [49] is ≈ 0.870, attained from Sallee's triangulation [95] of $\mathcal{C}^8$.

Most recently Hughes et. al. [59] determined the exact values of $\Delta(d)$ for dimensions 6 and 7, using an improved linear programming approach of [56]. This leads to the smallest value of $\rho \approx 0.840$ known so far. Together with the main theorem of Haiman [49] this also yields improved upper bounds for $d = 8$ and $d = 10$. (The best known upper bound for $d = 9$ is still given by [22]).

It is well known that every triangulation of $\mathcal{C}^d$ requires at least $\Omega(\sqrt{d!})$ simplices. This lower bound can be derived from volume considerations, see e. g. [49]. The basic observation is that the maximum volume of a simplex contained in the cube is $(d+1)^{(d+1)/2}/2^d d!$, attained by the equilateral simplex (cf. this to the existence of such simplices, discussed in Section 6.3). This lower bound was for example used in [13] to show that perturbation methods are not suitable to compute the convex hull of 0/1-polytopes, cf. also the discussion in Chapter 3.

Improved lower bounds are recently given by Hughes [57] for small dimensions, using linear programming methods. For large d the best known asymptotic lower bounds are due to Smith [104].

Table 9.1 gives the known values of $\Delta(d)$ for $d \leq 10$. It can be seen that the gap between lower an upper bounds increases dramatically even for moderately large d. The results are collected from papers cited above, especially [22, 56, 57, 58, 59]. Lower and upper bounds for specific dimensions can also be found in [24, 93, 94, 104]. Updates of the list given in Table 9.1 can partially also be found on the word wide web, see Problem 5.10 in [115].

A related open question is wheter it is true that the smallest number of $\Delta(d)$ is achieved by a vertex preserving triangulation, i.e. without new vertices; cf. Section 8.4.1 in [46].

9.2 Volumes of 0/1-Polytopes

In this section we give some additional pointers to the literature concerning the volume of the convex hull of random vertex subsets of the hypercube, as well as for the volume of slices and projections of $\mathcal{C}^d$. Some related work can also be found in Chapters 3 and 5.

We do not go into considering the algorithmic issues of computing the volume or convex hull of a subset of vertices from $\mathcal{C}^d$ here. But let us repeat a warning already given in Chapter 3: When using a standard convex hull algorithm one has to be careful, since vertices of $\mathcal{C}^d$ are highly degenerate in the sense that there are many linear dependences between them. This may cause numerical troubles in computing the convex hull and in fact many of the existing software packages fail on this task (cf. [26]). For example perturbation methods may result in superpolynomially sized convex hulls for the d-cube [13].

Of course most of the standard text books on polytopes also give results concerning the hypercube, see e.g. [46, 114]. Especially in Section 5 of [46] valuable references can be found. For example it is shown there that the problem of computing the volume of the intersection of the unit cube with a rational halfspace is $\#P$-hard, see also [37]. This is done by a reduction from a counting version of Integer Knapsack, see problem MP10 in Section 9.3.

Füredi [41] showed that for a subset of vertices from $\mathcal{C}^d$, for $d \to \infty$, and for an arbitrary $\epsilon > 0$ the convex hull of more than $(2+\epsilon)d$ vertices almost always contains the center of the cube, while the convex hull of less than $(2-\epsilon)d$ points almost always avoids it. This is related to the results of [38]. There the authors consider the following questions: What is the expected volume of the convex hull of a subset of uniformly chosen vertices of $\mathcal{C}^d$? And what happens to the expected volume if the vertices are sampled uniformly from within the hypercube? For the first question they show that, as $d \to \infty$, the expected volume vanishes if less than $(2/\sqrt{e}-\epsilon)^d$ vertices are taken, and that volume 1 is approximated for more than $(2/\sqrt{e}+\epsilon)^d$ vertices. Thus the number of necessary vertices to attain significant volume is surprisingly small. For the case that the points are taken from the interiour of the hypercube they show a similar result, where the constant $2/\sqrt{e}$ is replaced by a term which evaluates to approx. 2.136.

The volume of orthogonal projections of $\mathcal{C}^d$ onto orthogonal complementary subspaces is considered in [72]. The author shows that if the subspaces have dimension k and $d-k$, respectively, then the volumes (of dimension k and $d-k$) of the images of the projections are equal. In [74] the volume of intesections of $\mathcal{C}^d$ with hull honest hyperplanes (cf. Chapter 2) is investigated. Moreover they consider the volume between two hyperplanes normal to the main diagonal and at distance $\sqrt{d}/d$ from each other. For both problems explicite formulas are given. Additional results concerning central slices of $\mathcal{C}^d$ are also discussed at the end of Chapter 5.

9.3 NP-complete Problems on the Hypercube

The following list contains some of the NP-complete problems which are strongly related to the hypercube, out of Garey and Johnson's famous book about the theory of NP-completeness [43]. Most of the problems are mentioned in previous chapters of this thesis. Some of the unsolved problems of previous chapters, like e.g. variants of centroidal clustering of Chapter 7, possibly also belong to this list, but we have to leave this as an open problem. The classification numbers and titles are similar to those used in the Appendix of [43].

SR12 **HITTING STRING**

INSTANCE: Finite set A of strings over $\{0,1,*\}$, all having the same length n.

QUESTION: Is there a string $x \in \{0,1\}^*$ with $|x| = n$ such that for each string $a \in A$ there is some i, $1 \leq i \leq n$, for which the i^{th} symbol of a and the i^{th} symbol of x are identical?

MP1 **INTEGER PROGRAMMING**

INSTANCE: Finite set X of pairs (x, b), where x is an m-tuple of integers and b is an integer, an m-tuple c of integers, and an integer B.
QUESTION: Does there exist an m-tuple y of integers such that $x \cdot y \leq b$ for all $(x, b) \in X$ and such that $c \cdot y \geq B$?
Comment: Variant in which all components of y are required to belong to $\{0, 1\}$ (ZERO-ONE INTEGER PROGRAMMING) is also NP-complete, even if each b, all components of each x and all components of c are required to belong to $\{0, 1\}$. Also NP-complete are the questions of whether a y with non-negative integer entries exists such that $x \cdot y = b$ for all $(x, b) \in X$, and the question of whether there exists any y with integer entries such that $x \cdot y \geq 0$ for all $(x, b) \in X$.

MP10 **INTEGER KNAPSACK**

INSTANCE: Finite set U, for each $u \in U$ a size $s(u) \in \mathcal{Z}^+$ and a value $v(u) \in \mathcal{Z}^+$, and positive integers B and K.
QUESTION: Is there an assignment of non-negative integer $c(u)$ to each $u \in U$ such that $\sum_{u \in U} c(u) \cdot s(u) \leq B$ and such that $\sum_{u \in U} c(u) \cdot v(u) \geq K$?
Comment: Remains NP-complete if $s(u) = v(u)$ for all $u \in U$.

MP13 **COMPARATIVE VECTOR INEQUALITIES**

INSTANCE: Sets $X = \{x_1, x_2, \ldots, x_k\}$ and $Y = \{y_1, y_2, \ldots, y_l\}$ of m-tuples of integers.
QUESTION: Is there an m-tuple z of integers such that the number of m-tuples x_i satisfying $x_i \geq z$ is at least as large as the number of m-tuples y_i satisfying $y_i \geq z$, where two m-tuples u and v satisfy $u \geq v$ if and only if no component of u is less than the corresponding component of v?
Comment: Remains NP-complete even if all components of the x_i and y_i are required to belong to $\{0, 1\}$.

MS7 **DECODING OF LINEAR CODES**

INSTANCE: An $n \times m$ matrix $A = (a_{ij})$ of 0's and 1's, a vector $y = (y_1, \ldots, y_m)$ of 0's and 1's and a positive integer K.
QUESTION: Is there a 0-1-vector $x = (x_1, \ldots, x_n)$ with no more than K 1's such that, for $1 \leq j \leq m$, $\sum_{i=1}^{n} x_i \cdot a_{ij} \equiv y_j \pmod 2$?
Comment: If y is the all zero vector, and hence we are asking for a "codeword" of Hamming distance K or less, the problem is open. The variant in which we ask for an x with *exactly* K 1's is NP-complete, even for fixed $y = (0, \ldots, 0)$.

9.4 The Opaque Cube Problem

Let us close this thesis by stating a rather simple looking, but still open question for the 2-dimensional hypercube, i.e. the unit square: *What is the shortest length fence that can block any line of sight across a unit square?*

The best known solution so far is given in Figure 9.1. It consists of three lines coming from three corners and meeting in a point at angles of $2\pi/3$, and a single line connecting the center of the square and the fourth corner. It is not hard to see that this solution indeed

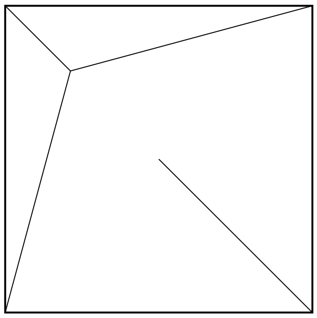


Figure 9.1: The best known opaque square solution.

blocks all lines intersecting the square. The length of these blocking fence is ≈ 2.6390 but it has not been proved that this is in fact best possible. For more details on this problem and some related topics see [23].

Gardner [42] has generalized this problem to 3D and called it the *Opaque Cube Problem*: What is the least area surface that can block all lines of sight through a cube? Here the best known solution has an area of ≈ 4.2324, but it is again an open problem to prove that this is best possible. Clearly this problem can be generalized to any dimension and also to other regions than the hypercube, and it is then called the *Opaque Region Problem*, see e.g. [23].

Bibliography

[1] E. A. Abbott, *Flatland - A Romance of Many Dimensions*, 1884

[2] O. Aichholzer, *Hyperebenen in Hyperkuben - eine Klassifizierung und Quantifizierung*, Diplomarbeit, IGI-TU Graz, 1992

[3] O. Aichholzer, *Clustering the Hypercube*, SFB Report Series 93, TU-Graz, 1996

[4] O. Aichholzer, *Practical Implementations of the Complete Linkage Clustering Method*, Manuscript, 1996

[5] O. Aichholzer, *Efficient {0,1}-String Searching Based on Pre-clustering*, SFB Report Series 94, TU-Graz, December 1996

[6] O. Aichholzer, F. Aurenhammer, *Classifying Hyperplanes in Hypercubes*, SIAM J. Discrete Math, Vol. 9, No.2, pp. 225-232, May 1996

[7] O. Aichholzer, F. Aurenhammer, *Classifying Hyperplanes in Hypercubes*, IIG Tech. Report 384, TU Graz, Feb. 1984

[8] N. Alon, E. E. Bergmann, D. Coppersmith, A. M. Odlyzko, *Balancing Sets Of Vectors*, IEEE Transactions on Information Theory 34, pp 128-130, 1988

[9] N. Alon and Z. Füredi, *Covering the Cube by Affine Hyperplanes*, European Journal of Combinatorics 14, pp 79-83, 1993

[10] H. Alt, it personal communication, 1995

[11] F. Aurenhammer, *Voronoi diagrams - a survey of a fundamental geometric data structure*, ACM Computing Surveys 23), pp 345–405, 1991

[12] F. Aurenhammer, R. Klein, *Voronoi Diagrams*, to appear as Chapter 18 in Textbook on Computational Geometry, J. Sack and G. Urrutia (eds), in preparation

[13] D. Avis, D. Bremner, *How Good are Convex Hull Algorithms?*, Proc. of the 11th Annual Symposium on Computational Geometry, Vancouver, British Columbia, Canada, pp 20-28, June 1995

[14] K. A. Baker, *A Generalization of Sperner's Lemma*, Journal of Combinatorial Theory 6, pp 224-225, 1969

[15] K. Ball, *Cube Slicing in $\mathcal{R}^n$*, Proc. Amer. Math. Soc. 97, pp 465-473, 1986

[16] K. Ball, *Volumes of Sections of Cubes and Related Problems*, Lect. Notes Math. 1376, pp 251-260, 1989

[17] K. Ball, *Some Remarks on the Geometry of Convex Sets*, Lecture Notes in mathematics 1317, pp 224-231, 1988

[18] T. Banchoff, Strauss, *The Hypercube: Projections and Slicing*, video, Intern. Film Buero Inc., Chicage, Illinois

[19] H. Bauersfeld, *Drei Gründe, Geometrisches Denken in der Grundschule zu fördern*, mathematica didactica, Vol. 16, pp 3-36, 1993

[20] E.R. Berlekamp, *Algebraic Coding Theory*, McGraw Hill, New York, 1968

[21] H.H. Bock, *Automatische Klassifikation*, Studia Mathematica, Vandenhoeck & Ruprecht, Göttingen, 1974

[22] J. Böhm, *Some Remarks on Triangulating a d-cube*, Beiträge zur Algebra und Geometrie, Vol. 29, pp 195-218, 1989

[23] K.A. Brakke, *The Obaque Cube Problem*, Amer. Math. Monthly, Vol 99(9), pp 866-871, 1992

[24] M.N. Broadie, R.W. Cottle, *A Note on Triangulating the 5-cube*, Discrete Mathematics, Vol. 52, pp 39-49, 1984

[25] N.G. de Bruijn, *Pólya's theory of counting*, in E. F. Beckenbach (ed.), Applied Combinatorial Mathematics, Wiley, NY 1964

[26] B. Büeler, A. Enge, K. Fukuda, H.-J. Lüthi, *Exact Volume Computation for Polytopes: A Practical Study*, Abstracts of the 12th European Workshop on Computational Geometry, Münster, pp 57-64, 1996

[27] W.A. Burkhard, R.M. Keller, *Some approaches to best-match file searching*, Comm. Ass. Comput. Mach. 16, pp 230-236, 1973

[28] W. Burnside, *Theory of Groups of Finite Order*, Cambridge University Press, 1911

[29] D. Chakerian, D. Logothetti, *Cube Slices, Pictorial Triangles, and Probability*, Mathematics Magazine 64, pp. 219-241, 1991

[30] T.H. Cormen, Ch.E. Leiserson, R.L. Rivest, *Introduction to Algorithms*, MIT Press Cambridge and McGraw-Hill Book Company NY, 1991

[31] R.W. Cottle, *Minimal Triangulation of the 4-cube*, Discrete Mathematics, Vol. 40, pp 25-29, 1982

[32] H.S.M. Coxeter, *Regular Polytopes*, Dover Publications, New York, 1963/73

[33] R. Craigen, *The Range of the Determinant Function on the Set of* $n \times n$ $(0,1)$*-Matrices*, Journal of Comb. Math. and Comb. Comp. 8, pp 161-171, 1990

[34] W.H.E. Day, H. Edelsbrunner, *Efficient Algorithms for Agglomerative Hierarchical Clustering Methods*, Journal of Classification Vol 1, pp 7-24, 1994

[35] D. Defays, *An efficient algorithm for a complete link method*, The Computer Journal, Vol. 20, No 4, pp 364-366, 1977

[36] B.S. Duran, P.L. Odell, *Cluster Analysis - A Survey*, Lecture Notes in Economics and Mathematical Systems 100, Springer Verlag, 1974

[37] M.E. Dyer, A.M. Frieze, *On the complexity of computing the volume of a Polyhedron*, SIAM Journal of Computing, Vol. 17, pp 967-974, 1988

[38] M.E. Dyer, Z. Füredi, C. McDiarmid, *Volumes Spanned by Random Points in the Hypercube*, Random Structures and Algorithms, Vol. 3, pp 91-106, 1992

[39] M.R. Emamy-Khansary, *On the Cuts and Cut Number of the 4-cube*, Journal of Combinatorial Theory, Series A, Vol. 41, pp 221-227, 1986

[40] K. Florek, J. Łukaszewicz, J. Perkal, H. Steinhaus, S. Zubrzycki, *Sur la liaison et la division des points d'un ensemble fini*, Colloquium Math., Vol 2, pp 282-285, 1951

[41] Z. Füredi, *Random Polytopes in the d-Dimensional Cube*, Discrete & Computational Geometry 1, pp 315-319, 1986

[42] M. Gardner, *The Opaque Cube Problem*, Cubism for fun, Vol. 23, p.15, 1990

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

[44] E.N. Gilbert, *Gray Codes and Paths on the n-Cube*, Bell Systems Tech. J. 37, pp 1-12, 1958

[45] R.L. Graham and H.O. Pollak, *On the Addressing Problem for Loop Switching*, Bell Systems Tech. J. 50, pp 2495-2519, 1971

[46] P. Gritzmann, V. Klee, *On the complexity of some basic problems in computational convexity: II Volume and mixed volumes*, in POLYTOPES: Abstract, Convex and Computational, T. Bisztriczky et al. (eds.), pp 373-466, 1994

[47] M. Grötschel and M.W. Padberg, *Polyhedral Aspects of the Travelling Salesman Problem*, in: E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan (eds.), The Travelling Salesman Problem, John Wiley and Sons, London, New York, 1985

[48] B. Grünbaum, *Convex Polytopes*, Interscience, New York, 1967

[49] M. HAIMAN, *A Simple and Relatively Efficient Triangulation of the n-Cube*, Discrete Computaional Geometry, Vol. 6, pp 287-289, 1991

[50] M. HALL JR., *Combinatorial Constructions*, Studies in Combinatorics (G.-C. Rota ed.), MAA Studies in Mathematics 17, pp 218-253, 1978

[51] A.EL-HAMDOUCHI, P. WILLETT, *Comparison of Hierarchical Agglomerative Clustering Methods for document Retrieval*, The Computer Journal, Vol. 32, No.3, 1989

[52] J.A. HARTIGAN, *Clustering Algorithms, Wiley Series in Probability and Mathematical Statistics*, John Wiley & Suns, NY, 1975

[53] J. HÅSTAD, *On the Size of Weights for Threshold Gates*, SIAM Journal of Discrete Mathematics, Vol. 7, pp 484-492, 1994

[54] M. HENK, J. RICHTER-GERBER, G.M. ZIEGLER, *Basic Properties of Convex Polytopes*, manuscript

[55] D. HENSLEY, *Slicing the Cube in $\mathcal{R}^d$ and Probability (Bounds for the Measure of a Central Cube Slice in $\mathcal{R}^d$ by Probability Methods)*, Proc. Amer. Math. Soc., Vol. 73, pp 95-100, 1979

[56] R.B. HUGHES, *Minimum-cardinality triangulations of the d-cube for $d = 5$ and $d = 6$*, Discrete Mathematics, Vol. 118, pp 75-118, 1993

[57] R.B. HUGHES, *Lower Bounds on Cube Simplexity*, Discrete Mathematics, Vol. 133, pp 123-138, 1994

[58] R.B. HUGHES, M.R. ANDERSON, *A triangulation of the 6-cube with 308 simplices*, Discrete Mathematics, Vol. 117, pp 253-256,1993

[59] R.B. HUGHES, M.R. ANDERSON, *Simplexity of the cube*, Discrete Mathematics, Vol. 158, pp 99-150, 1996

[60] R. IMPAGLIAZZO, R. PATURI, M. E. SAKS, *Size-Depth Trade-offs for Threshold Circuits*, Proceedings of the 25th ACM Symposium on Theory of Computing, pp 541-550, 1993

[61] K. JACOBS, *Einführung in die Kombinatorik*, de Gruyter, Berlin New York, 1983

[62] L.S. JOEL, D.R. SHIER AND M.L. STEIN, *Planes, Cubes and Center-Representable Polytopes*, Am. Math. Monthly 84, pp 360-363, 1977

[63] S.C. JOHNSON, *Hierarchical clustering schemes*, Psychometrika 32, pp 241-254, 1967

[64] U.H. KORTENKAMP, *Small 0/1-Polytopes with Many Facets*, `http://www.math.tu-berlin.de/~hund/01-Olympics.html`

[65] U.H. KORTENKAMP, J. RICHTER-GERBERT, A. SARANGARAJAN, G. M. ZIEGLER, *Extremal Properties of 0/1-Polytopes*, Discrete & Computational Geometry, to appear.

[66] D. Krznaric, Ch. Levcopoulos, *The first Subquadratic Algorithm for Complete Linkage Clustering*, Proceedings of ISAAC'95, Ed. J. Staples et.al., LNCS 1004 Springer, pp. 392-401, 1995

[67] J.C. Lagarias and P.W. Shor, *Keller's Cube-Tiling Conjecture is False in High Dimensions*, DIMACS Tech. Rep. 92-13, 1992

[68] J.H. van Lint, R.M. Wilson, *A Course in Combinatorics*, Cambridge University Press, 1992

[69] P. MacNaughton-Smith, *Some statistical and other numerical techniques for classifying individuals*, Home Office Research Unit Report No. 6, H.M.S.O., London, 1965

[70] P.S. Mara, *Triangulations for the Cube*, J. of Combinatorial Theory (A), Vol. 20, pp 170-177, 1976

[71] H. Martini, *Reguläre Polytope und Verallgemeinerungen, in Geometrie und ihre Anwendungen*, O.Giering, J. Hoschek Eds., Carl Hanser Verlag, Munich, 1994

[72] P. McMullen, *Volumes of Projections of Unit Cubes*, Bull. London Math. Soc., Vol. 16, pp 278-280, 1984

[73] L.L. McQuitty, *Hierarchical linkage analysis for the isolation of types*, Educational Psych. Measurement 20, pp 55-67, 1960

[74] W. Meyer, R. Randow, *Ein Würfelschnittproblem und Bernoullische Zahlen*, Mathematische Annalen, Vol. 193, pp 315-321, 1971

[75] M. Minsky, S. Papert, *Perceptrons: an introduction to computational geometry*, The MIT Press, Cambridge, 1969

[76] S. Muroga, *Threshold Logic and its Applications*, Wiley-Interscience, New York, 1971

[77] S. Muroga, I. Toda and S. Takasu, *Theory of majority decision elements*, J. Franklin Inst. 271, pp 376-418, May 1961

[78] S. Muroga, T. Tsuboi, C.R. Baugh, Enumeration of Threshold Functions of Eight Variables, IEEE Transactions on Computers, Vol. C-19, pp 818-825, 1970

[79] F. Murtagh, *A Survey of Recent Advances in Hierarchical Clustering Algorithms*, The Computer Journal, Vol 26, No.4, pp 354-359, 1983

[80] H. Naumann, *Beliebige konvexe Polytope als Schnitte und Projektionen höherdimensionaler Würfel, Simplices und Masspolytope*, Mathematische Zeitschrift 65, pp 91-103, 1956

[81] P.E. O'Neil, *Hyperplane Cuts of an n-Cube*, Discrete Mathematics 1, pp 193-195, 1971/72

[82] P. Orponen, *Computational Complexity of Neural Networks: A Survey*, Nordic Journal of Computing 1, pp 94-110, 1994

[83] C.H. Papadimitriou, *Computational Complexity*, Addison-Wesley, 1994

[84] I. Parberry, *Circuit Complexity and Neural Networks*, The MIT Press, Cambridge, MA, 1994

[85] I. Parberry, *The Computational and Learning Complexity of Neural Networks*, manuscript

[86] G. Pólya, *Kombinatorische Anzahlbestimmungen für Gruppen, Graphen und chemische Verbindungen*, Acta Mathematica Vol. 68, pp 145-254, 1937

[87] F.P. Preparata, M.I. Shamos, *Computational Geometry - An Introduction*, Springer Verlag, 1985

[88] V. Ramasubramanian, K.K. Paliwal, *An efficient approximation-elimination algorithm for fast nearest-neighbour search based on a spherical distance coordinate formulation*, Pattern Recognition Letters 13, pp 471-480, 1992

[89] G. Reinelt, *The Linear Ordering Problem: Algorithms and Applications*, Research and Exposition in Mathematics 8, Heldermann Verlag, Berlin, 1985

[90] R.L. Rivest, *On the optimality of Elias's algorithm for performing best-match searches*, Information Processing, North Holland Publishing Company, pp 678-681, 1974

[91] G. Rote, personal communication, 1996/1997

[92] M.E. Saks, *Slicing the Hypercube*, in: Surveys in Combinatorics, K. Walker (ed.), London Mathematical Society Lecture Note Series 187, Cambridge University Press, pp. 211-256, 1993

[93] F. Sallee, *A Triangulation of the n-cube*, Discrete Mathematics, Vol. 40, pp 81-86, 1982

[94] F. Sallee, *A note on minimal Triangulations of an n-cube*, Discrete Applied Mathematics, Vol. 4, pp 211-215, 1982

[95] F. Sallee, *The Middle-cut triangulations of the n-cube*, SIAM J. Algebraic Discrete Methods, Vol. 5, pp 407-419, 1984

[96] A. Schrijver, *Theory of Linear and Integer Programming*, Wiley-Interscience Series in Discrete Mathematics and Optimization, John Wiley & Sons, Chichester, 1986.

[97] R. Seidel, *Exact Upper Bounds for the Number of Faces in d-Dimensional Voronoi Diagrams*, in Applied Geometry and Discrete Mathematics, Vol 4., P. Gritzmann and B. Sturmfels (ed.), pp. 517–530, 1991

[98] R. Sibson, *SLINK: An optimally efficient algorithm for the single-link cluster method*, The Computer Journal, Vol 16, No 1, pp30-45, 1973

[99] G. Sierksma, *Linear and Integer Programming - Theory and Practice*, Marcel Dekker, Inc, New York, 1996

[100] I. Singer, *Extensions of Functions of 0-1 Variables and Applications to Combinatorial Optimization*, Numer. Funct. Analysis and Optimization, Vol. 7, pp 23-62, 1984/85

[101] K.-Y. Siu, V. Roychowdhury, T. Kailath, *Discrete Neural Computation - A Theoretical Foundation*, Englewood Cliffs, Prentice-Hall, 1995

[102] K.-Y. Siu, V. Roychowdhury, T. Kailath, *Rational Approximation Techniques for Analysis of Neural Networks*, IEEE Transactions on Information Theory, Vol. 40, pp 455-466, 1994

[103] N.J.A. Sloane, *A Short Course on Error Correcting Codes*, CISM Courses and Lectures 188, Springer, Wien, New York, 1975

[104] W.D. Smith, *Lower Bounds for Triangulations of the n-cube*, manuscript, 1994

[105] T. Sørenson, *A method of establishing groups of equal amplitude in plant sociology based on similarity of species content and its application to analyses of the vegetation on Danish commons*, Biol. Skrifter 5, pp 1-34, 1948

[106] E. Sperner, *Ein Satz über Untermengen einer endlichen Menge*, Mathematische Zeitschrift 27, pp 544-548, 1928

[107] M.J. Todd, L. Tuncel, *A new triangulation for simplical algorithms*, SIAM Journal of Discrete Mathematics, Vol. 6, pp 167-180, 1993

[108] E. Vidal, *An algorithm for finding nearest neighbours in (approximately) constant average time*, Pattern Recognition Letters 4, pp 145-157,1986

[109] E. Vidal, *New formulation and improvements of the Nearest-Neighbour Approximating and Eliminating Search Algorithm (AESA)*, Pattern Recognition Letters 15, pp 1-7, 1994

[110] J.M. Vilar, *Reducing the overhead of the AESA metric-space nearest neighbour searching algorithm*, Information Processing Letters 56, pp 265-271, 1995

[111] E.M. Voorhees, *The Effectiveness and Efficiency of Agglomerative Hierarchic Clustering in Document Retrieval*, Ph.D. thesis, Cornell University, 1985

[112] T. Welch, *Bounds on the information retrieval efficiency of static file structures*, Technical Report 88, MIT, 1971

[113] H.G. Wells, *The Time Machine*, p.5, 1895

[114] G.M. Ziegler, *Lectures on Polytopes*, Graduate Texts in Mathematics 152, Springer-Verlag, 1994

[115] G.M. Ziegler, *Lectures on Polytopes: Updates, Corrections, and More*, `http://www.math.tu-berlin.de/~ziegler/www/poly-up.ps.Z`

[116] G. M. Ziegler, *Recent Progress on Polytopes*, in Proc. "Discrete and Computational Geometry: Ten Years Later", Mt. Holyoke, B. Chazelle, J.E. Goodman, R. Pollack, (eds.), Contemporary Mathematics, Amer. Math. Soc., 1996

[117] G. M. Ziegler, *Polytopes and Optimization: Recent Progress and Some Challenges*, manuscript