Constructing Covering Codes

Master Thesis
Luca Teodoro Filippini
Thursday 24th March, 2016

Advisors: Prof. Dr. E. Welzl, C. Annamalai
Department of Theoretical Computer Science, ETH Zürich
Abstract

Given $r, n \in \mathbb{N}$, the problem of constructing a set $C \subseteq \{0, 1\}^n$ such that every element in $\{0, 1\}^n$ has Hamming distance at most $r$ from some element in $C$ is called the covering code construction problem. Constructing a covering code of minimal size is such a hard task that even for $r = 1$, $n = 10$ we don’t know the exact size of the minimal code. Therefore, approximations are often studied and employed. Among the several applications that such a construction has, it plays a key role in one of the fastest 3-SAT algorithms known to date.

The main contribution of this thesis is presenting a Las Vegas algorithm for constructing a covering code with linear radius, derived from the famous Monte Carlo algorithm of random codeword sampling. Our algorithm is faster than the deterministic algorithm presented in [5] by a cubic root factor of the polynomials involved. We furthermore study the problem of determining the covering radius of a code: it was already proven $\mathcal{NP}$-complete for $r = n/2$, and we extend the proof to a wider range of radii. Along the way, we introduce a new Satisfiability problem, and investigate its hardness.
Acknowledgements

Foremost, I would like to thank my advisor Prof. Welzl for having stimulated my interest in the field of combinatorial algorithms during the SAT course, for a great deal of notation borrowed from the course book [19], and for the possibility of writing this thesis under his supervision. I also thank my co-advisor Chidambaram for his suggestions and for proof-reading the thesis.

Furthermore, my gratitude goes to my fellow students in the Theoretical Computer Science track for continuous support throughout my studies, and in particular, to my dear friend Miloš, for his unlimited curiosity in this field. For precious coffee breaks and an everlasting billiards rivalry I have to thank my friends Gabriele, Billy, Kevin and Radek.

I also thank my parents Angela and Rocco and my second mother Lilia, for supporting me in every sense, during my studies. Last, but not least, I am grateful to my beloved fiancée Martina, for her support, and her help during this experience. I love you all, very much.
Notation

Generalities

By $[n]$ we denote the set $\{0, \ldots, n\} \subset \mathbb{N}_0$. By $[x,y] \subset \mathbb{R}$ and $(x,y) \subset \mathbb{R}$ we denote respectively the closed and the open real interval between $x$ and $y$.

We adopt the traditional Landau (so-called "big-O") notation with the following extension: for a function $f(n) \in \omega\left(poly(n)\right)$, we denote by $O^*(f(n))$ the set $O\left(f(n) \cdot poly(n)\right)$, and by $\Theta^*(f(n))$ the set $\Theta\left(f(n) \cdot poly(n)\right)$. We furthermore make use of the term "asymptotically", meaning, if not otherwise specified, "up to a polynomial factor".

By saying that a variable $\epsilon$ is drawn uniformly at random (u.a.r.) from a set $S$, we mean that it is sampled according to the uniform distribution over the elements of $S$, i.e., $\forall s \in S, \Pr(\epsilon = s) = \frac{1}{|S|}$.

We use the acronym w.l.o.g. for “without loss of generality”.

Coding theory

Let $\mathbb{F}_q^n$ denote the n-dimensional space over $\mathbb{F}_q = \{0, \ldots, q - 1\}$, the finite field of $q$ elements, and let its elements be called $q$-ary vectors of length $n$, denoted in bold font. We will mostly consider $\mathbb{F}_2^n$, the $n$-dimensional Hamming space, and denote it as $\mathbb{H}_n$; the definitions of Hamming distance, code, and codeword can be extended from $\mathbb{F}_2$ to any $\mathbb{F}_q$ in the natural way.

For a binary vector $x = (x_1 \ldots x_n)$, we denote by $\overline{y} = (\overline{x}_1 \ldots \overline{x}_n)$ its bitwise negation. For $x,y \in \{0,1\}$, we denote by the symbol $x \oplus y$ the exclusive disjunction (XOR) of $x$ and $y$, i.e., the Boolean function that outputs 0 if and only if both inputs have the same value. For two vectors $c \in \mathbb{H}_i$ and $d \in \mathbb{H}_j$ we denote by $c \cdot d \in \mathbb{H}_{i+j}$ their concatenation, which we extend to sets as $C \cdot D := \{c \cdot d \mid c \in C, \; d \in D\}$.

A covering code (which we will often refer to simply as “code”) is denoted by the symbol $\mathcal{C}$, whereas the symbol $B_n(c,r)$ denotes a Hamming ball with
radius \( r \) centred in \( c \), in the \( n \)-dimensional Hamming space. Moreover, we let \( \text{CR}(\mathcal{C}) \) denote the covering radius of \( \mathcal{C} \).

By \( 0^n \) and \( 1^n \) we denote the all-0 and all-1 vectors of length \( n \). Two binary vectors \( x = (x_1 \ldots x_n) \) and \( y = (y_1 \ldots y_n) \) are called \textit{antipodal} if they are opposite in each coordinate, that is, \( x = \overline{y} \), or equivalently

\[
\sum_{i=1}^{n} x_i \oplus y_i = n.
\]

The Hamming distance between two vectors \( a, b \in \{0, 1\}^n \) is denoted by \( d_H(a, b) \). For \( a \in \{0, 1\}^n \), \( \mathcal{C} \subseteq \{0, 1\}^n \), we will write \( d_H(a, \mathcal{C}) \) as a shorthand for the minimum Hamming distance from \( a \) to an element of \( \mathcal{C} \):

\[
d_H(a, \mathcal{C}) = \min_{c \in \mathcal{C}} d_H(a, c).
\]

### Randomised algorithms

A randomised algorithm is an algorithm having access to a source of randomness, i.e., being able, in one time-step, to “toss a coin” in order to decide the direction of some of its computations. We model this process by allowing the algorithm to access not only the input tape like a deterministic algorithm, but also a tape containing a random sequence of bits.

![Randomised Algorithm Diagram](image)

\[
\text{Input Tape} \quad \cdots \quad \text{Random Bit Sequence} \quad \cdots
\]

\[
\text{Randomised Algorithm} \quad \text{Output Tape} \quad \cdots
\]

**Figure 1:** The diagram of a randomised algorithm.

In this model, there is a probability \( p \in [0, 1] \) for either the running time to be equals to some \( s \), or for the return value to be correct.

A \textit{Monte Carlo} algorithm is a randomised algorithm returning a correct answer with probability \( p \in (0, 1) \) in a deterministically determined running time, whereas a \textit{Las Vegas} algorithm always returns the correct answer, but
the running time is probabilistic. It is always possible to transform a Las Vegas algorithm into a Monte Carlo by limiting the computation of the former up to a certain number of steps, and then returning the result. For the other way round, no pre-determined strategy is known to work in the general case.

Naturally, we are interested in algorithms returning the correct value (or taking at most $s$ loops to return it) with a high probability of success. To be precise, we say that a Las Vegas algorithm, whose input size is $n$, terminates with high probability within $O(s)$ time if it does so with probability at least $1 - 1/n^c$ for any choice of a constant $c \geq 1$. The choice for the constant $c$ should only affect the running time by a factor of $o(s)$. A similar definition applies to Monte Carlo algorithms: one such algorithm returns a correct result $w.h.p.$ if it does so with probability at least $1 - 1/n^c$ for any choice of $c \geq 1$. 

vii
Acknowledgements

Notation

Contents

1 Introduction

1.1 Coding Theory

1.1.1 Covering Codes

1.1.2 The Hardness of Covering Code Construction

1.2 Satisfiability and Covering Codes

1.2.1 Boolean Satisfiability

1.2.2 The Hamming Space and Truth Assignments

1.2.3 Dantsin et al.’s Algorithm

2 The Covering Radius Problem

2.1 Introduction

2.2 Radius and Covering Radius

2.3 \(\mathcal{NP}\)-Completeness

2.3.1 Overview

2.3.2 Preliminaries

2.3.3 Frances and Litman’s Proof

2.3.4 Extending the Proof

2.3.5 Open Problems

2.4 A New Satisfiability Problem

2.4.1 \(r\)-\(n\)-SAT Definition

2.4.2 An Algorithm for \(r\)-\(n\)-SAT

2.4.3 Hardness

2.5 Conditional Lower Bounds

2.5.1 The Strong Exponential Time Hypothesis
## Contents

2.5.2 A Lower Bound on the Running Time for $r$-$n$-SAT . . . 21  
2.6 Conclusion .............................................. 22

3 A New Las Vegas Algorithm 23  
3.1 Introduction ............................................. 23  
3.2 The Monte Carlo Algorithm ............................. 23  
3.3 The Las Vegas Algorithm ................................. 25  
3.4 Estimation of the Result’s Quality ........................ 26

4 Piecewise Linear Codes 29  
4.1 Introduction ............................................. 29  
4.1.1 Piecewise Constant Codes Definition ................. 29  
4.1.2 Piecewise Linear Codes Definition ..................... 31  
4.1.3 Motivation ............................................. 31  
4.1.4 A Conjecture on the Grid Cover ........................ 32 
4.2 Covering the Grid ....................................... 33  
4.2.1 Problem Definition .................................... 33  
4.2.2 2 Dimensions ......................................... 34  
4.2.3 3 Dimensions ......................................... 37  
4.2.4 Larger Dimensions .................................... 40  
4.3 Disproving the Conjecture ................................. 43  
4.3.1 Conclusion ............................................. 47

A Bounds on $K(n,r)$ 49

Bibliography 51
Chapter 1

Introduction

This thesis is organised as follows.

In this chapter, we briefly introduce the reader to the concepts of coding theory that we use throughout the thesis; then, we define the Boolean Satisfiability problem, and present Dantsin et al.’s famous k-SAT algorithm [5] using covering codes.

In Chapter 2, we focus on the problem of computing the covering radius of a code, and present an algorithm for solving it. We furthermore define a new Satisfiability problem, equivalent to the Covering Radius problem, and analyse its hardness.

In Chapter 3, we combine the algorithm for computing the covering radius with a well-known Monte Carlo algorithm for constructing covering codes, obtaining a Las Vegas algorithm with a faster runtime and smaller space consumption than every deterministic algorithm so far known for the construction of codes with large radii.

Chapter 4 is dedicated to the possibility of extending a construction approach based on word subdivision in blocks by Hamming weight from [2] to arbitrarily large codes. Although such approach fails to find an effective strategy in the general case, we manage to prove its shortcomings, and to address possible extensions for further research.

1.1 Coding Theory

1.1.1 Covering Codes

The covering problem is the discrete counterpart of the problem of covering the euclidean space by spheres, a natural problem in geometry. It can be defined over any vector space, but we restrict ourselves to the binary \( n \)-dimensional Hamming space \( \mathbb{H}_n \), i.e. the set \( \{0, 1\}^n \), whose elements are
1. Introduction

called words. Any nonempty subset of the Hamming space is called code, and its elements codewords.

The Hamming distance \( d_H(a, b) : \mathbb{H}_n \times \mathbb{H}_n \mapsto [n] \), for \( a = (a_1a_2 \ldots a_n) \), \( b = (b_1b_2 \ldots b_n) \in \mathbb{H}_n \), is defined as

\[
d_H(a, b) = \sum_{i=1}^{n} (a_i \oplus b_i),
\]

the number of coordinates in which \( a \) and \( b \) differ. The Hamming distance is a metric, since for all \( x, y, z \in \mathbb{H}_n \):

- \( d_H(x, y) \geq 0 \),
- \( d_H(x, y) = 0 \iff x = y \),
- \( d_H(x, y) = d_H(y, x) \), and
- \( d_H(x, y) + d_H(y, z) \geq d_H(x, z) \).

A Hamming ball of radius \( r \) centered in \( s \in \mathbb{H}_n \) is the set

\[
B_n(s, r) = \{ t \in \{0, 1\}^n | d_H(s, t) \leq r \}.
\]

A code \( C \in \mathbb{H}_n \) has covering radius \( r \) if and only if \( r \) is the smallest integer such that every word in the \( n \)-dimensional Hamming space has Hamming distance at most \( r \) from some element in \( C \). Equivalently:

\[
\text{CR}(C) = \arg \min_{r \in [n]} \left( \bigcup_{c \in C} B_n(c, r) = \mathbb{H}_n \right).
\]

A covering code with covering radius \( r \) is called \( r'\)-covering, for every \( r' \geq r \). Every code is \( n \)-covering. A code consisting of two antipodal words is \( \left\lfloor \frac{n}{2} \right\rfloor \)-covering: for instance,

\[
C = \{(000000), (111111)\}
\]

has distance at most three from each word containing at least three 0s due to its first codeword, and distance smaller than three to each word containing less than three 0s due to its second coordinate.

For every binary vector \( s \in \{0, 1\}^n \), there are \( \binom{n}{i} \) vectors at distance \( i \) from it: the binomial coefficient counts the number of ways to choose coordinates of \( s \) that, if flipped, result in a word \( t \) with \( d_H(s, t) = i \). By considering the number of elements at distance \( \{0, 1, \ldots, r\} \) from a vector \( c \in \mathbb{H}_n \), we obtain the formula for the size of a Hamming ball of radius \( r \) around it, also known as volume:

\[
\text{Vol}(n, r) = \sum_{i=0}^{r} \binom{n}{i}.
\]
1.2 Satisfiability and Covering Codes

independent of the centre of the ball. This quantity is often unpleasant
to deal with in calculations, thus estimates for it are employed; the most
accurate, for radii linear in \( n \) (we will later see why this is the range we
most care about), is given by the binary entropy function

\[
H(x) := -x \log_2 x - (1 - x) \log_2 (1 - x),
\]

since for a ball of radius \( \rho n, \rho \in (0, \frac{1}{2}) \), we know from [13, Chapter 10] that

\[
\text{Vol}(n, \rho n) \in \left( 2^{H(\rho)n} \cdot (8n\rho(1 - \rho))^{-1/2}, \ 2^{H(\rho)n} \right).
\]

The sphere covering bound is the number of Hamming balls of radius \( r \) needed
to pack the \( n \)-dimensional Hamming space, i.e., the quantity \( 2^n / \text{Vol}(n, r) \).
Naturally, the sphere covering bound is a lower bound on the size of the
covering code with corresponding covering radius and length.

1.1.2 The Hardness of Covering Code Construction

Given \( n, r \in \mathbb{N} \), the problem of constructing a binary covering code of length
\( n \) and covering radius \( r \) of minimum size is extremely hard to solve. The
extensive search over all subsets of the Hamming space takes doubly expo-
nential time \( O^*(2^{2^n}) \), and no better technique is known, for the general case. 
Therefore, the exact size of the optimal codes is unknown even for small
lengths; see Appendix A for a table of up-to-date bounds on the minimal
size of covering codes.

Furthermore, given a code \( \mathcal{C} \), it is often desirable to be able to compute its
covering radius. Once again, we are underwhelmed: the problem is \( \mathcal{NP} \)-
hard, since the decision problem of lower bounding the covering radius was
proven to be \( \mathcal{NP} \)-complete in [6]. We will show their proof, and consider
the Covering Radius problem in depth in Chapter 2.

1.2 Satisfiability and Covering Codes

1.2.1 Boolean Satisfiability

The Boolean Satisfiability (SAT) problem asks to determine, given a boolean
formula \( F \) in propositional logic, whether there exists an assignment of its
variables such that \( F \) evaluates to \text{true}.

Not only has SAT endless practical applications [14], but it is also and fore-
most of central importance in theoretical computer science, since it was the
first problem to be shown to be \( \mathcal{NP} \)-complete (by Cook in [4], 1971).

We focus on Boolean formulas in conjunctive normal form (CNF). Formally,
by \text{variable} we mean a Boolean variable taking values in \{\text{false, true}\}, which
1. Introduction

can be denoted equivalently by \{0, 1\}; a variable \(x\) can give rise to two literals \(x\) (positive literal) and \(\overline{x}\) (negative literal). A truth assignment, or simply assignment, over the variable set \(x_1, \ldots, x_n\) is a map \(\alpha : \{x_1, \ldots, x_n\} \mapsto \{0, 1\}\). A literal is satisfied if either it is positive and the corresponding variable is true, or it is negative and the corresponding variable is false. A clause \(C\) is a set of literals that doesn’t contain a literal and its negation; we say that a truth assignment over a variable set satisfies a clause \(C\) if it satisfies at least one literal in \(C\), and that it falsifies the clause otherwise.

In the CNF-SAT problem, the input formula is a set of clauses in conjunctive normal form:

\[
F = \bigwedge_{i=1}^{m} \bigvee_{j=1}^{k(i)} u_{i,j},
\]

where \(u_{i,j}\) are the literals of the formula, and \(k(i) \in [n]\) is the number of literals in the \(i\)-th clause.

Given a clause \(C\), we denote by \(C[x_i \mapsto T]\) for \(T \in \{0, 1\}\) the same clause where the literal \(x_i\) takes truth value \(T\):

\[
C[x_i \mapsto T] = \begin{cases} 
C \setminus \{x_i\}, & \text{if } x_i \in C \text{ and } T = 0 \\
C \setminus \{\overline{x}_i\}, & \text{if } \overline{x}_i \in C \text{ and } T = 1 \\
1, & \text{if } x_i \in C \text{ and } T = 1, \text{ or } \overline{x}_i \in C \text{ and } T = 0 \\
C, & \text{otherwise.}
\end{cases}
\]

For a formula \(F\) and \(T \in \{0, 1\}\) we denote by \(F[x_i \mapsto T]\) the original formula where the literal \(x_i\) takes truth value \(T\) in every clause \(C\) of \(F\). We however remove from \(F[x_i \mapsto T]\) satisfied clauses: since a CNF formula is a conjunction of clauses, every clause it contains must be satisfied in order for the whole formula to evaluate to 1. Hence:

\[
F[x_i \mapsto T] = \{C[x_i \mapsto T] | C \in F\} \setminus \{1\}.
\]

For example, assuming \(T = 1\), for every \(C \in F\): if \(C\) contains the literal \(x_i\) we remove the clause from \(F[x_i \mapsto T]\); if \(C\) contains the literal \(\overline{x}_i\) we remove \(\overline{x}_i\) from \(C \in F[x_i \mapsto T]\); and if \(C\) doesn’t contain \(x_i\) nor \(\overline{x}_i\), we keep the clause as it is. Repeating this process until all variables in the variable set are mapped to the truth values \(T_1, \ldots, T_n\) can have two results:

- All clauses are removed from \(F\), which becomes the empty formula (denoted by \(\{\}\)), and is thus satisfied by the assignment \(\alpha = (T_1, \ldots, T_n)\);
- One clause loses all its literals and becomes the empty clause (denoted by \(\Box\)), therefore the formula is falsified by the assignment \(\alpha\).
Mind the strong difference between an empty clause and an empty formula! The formula is satisfied by an assignment $\alpha$ if and only if every clause is; in that case, we call $\alpha$ a 
*satisfying assignment*.

The restriction of the CNF-SAT problem where every clause of the formula has the same size $k$ is called $k$-SAT. For $k \leq 2$, the problem is in $\mathcal{P}$, and it is $\mathcal{NP}$-complete otherwise. Typically, for CNF-SAT and $k$-SAT, the number of variables of the formula is denoted by $n$, and the number of clauses by $m$.

### 1.2.2 The Hamming Space and Truth Assignments

The brute-force solution for CNF-SAT considers every possible assignment on the variable set of the formula, until either finding one satisfying all the clauses of the formula, or exhausting the search. Naturally, this strategy takes time $O(2^n \cdot \text{poly}(m, n))$; in order to improve upon it, a part of the possible assignments needs to be ignored. For instance, if an assignment $\alpha_1$ over a portion of the variable set falsifies a clause, then all assignments containing $\alpha_1$ will not satisfy the formula, and going through all $2^n - |\alpha_1|$ of them makes no sense. Consider, for example, the formula

$$F = x \land (x \lor y \lor z) \land (\neg z \lor \neg x).$$

All assignments with $[x \mapsto 0]$ can be ignored right away!

Observe, now, that the space of all assignments over a variable set of size $n$ coincides with the $n$-dimensional Hamming space. We will now show how an algorithm searching for a satisfying assignment in Hamming balls within the Hamming space performs significantly better than the extensive search. Our task will be to choose the minimal number of Hamming balls covering the assignment space: exactly what a covering code is made for.

### 1.2.3 Dantsin et al.’s Algorithm

In his famous paper [17] from 1999, Schöning presented a Monte Carlo algorithm for $k$-SAT based on random walks, achieving an expected runtime of $(2 - 2/k)^n$. The algorithm works by repeating an exponential number of times the following process:

1. Draw an assignment $\alpha \in \{0, 1\}^n$ uniformly at random;
2. Repeat $3n$ times: if the assignment does not satisfy $F$, flip a random literal in a random unsatisfied clause.

Both the starting point and the local search are thus randomised. The algorithm reaches w.h.p. a satisfying assignment, if there is one; the proof is carried out with a Markov chain modelling the process, and we will not present it here.
1. Introduction

Starting from the local search idea of Schöning’s algorithm, the next year, in [5], Dantsin and Schöning himself, along with Goerdt, Hirsch, Kannan, Kleinberg, Papadimitriou and Raghavan, presented a deterministic local search algorithm solving \( k \)-SAT in \( (2 - 2/(k + 1))^n \), employing a covering code in order to derandomise the choice of a starting assignment for the local search, and Hamming balls of fixed radius for derandomising the search itself.

Consider a \( k \)-SAT formula \( F \). The ball of radius \( r = \rho n \) around an assignment \( \alpha \) has \( \text{Vol}(n, \rho n) \leq 2^{H(\rho)n} \) many elements: in order to explore only part of them, we use the formula itself to prune the search.

**Observation 1.1** Let the input formula \( F \) be false under an assignment \( \alpha \in \{0, 1\}^n \). Then, for an arbitrary clause \( C \in F \) falsified by \( \alpha \), \( F \) has a satisfying assignment at Hamming distance at most \( r \) from \( \alpha \) if and only if there is a literal \( x \in C \) such that \( F[\alpha \mapsto x] \) has a satisfying assignment within the ball of radius \( r - 1 \) around \( \alpha \).

This intuition is captured in the following procedure:

**Algorithm 1:** \( \text{Src-ball}(F, \alpha, r) \)

```plaintext
if \( F[\alpha] = \{\} \) then
    return true;
else if \( r = 0 \) OR \( \emptyset \in F \) then
    return false;
else
    pick a clause \( C \) falsified by \( \alpha \) according to some deterministic rule;
    forall the \( x \in C \) do
        if \( \text{Src-ball}(F[\alpha \mapsto x], \alpha, r - 1) = \text{true} \) then
            return true;
        end
    end
return false;
```

The algorithm features \( r \) recursive calls, each on a clause of size \( k \); thus, it runs in time \( \text{poly}(n) \cdot k^r \). Notice that this can be much smaller than the volume of a ball of radius \( r \): for example, for \( k = 3 \) and \( r = n/2 \), \( \text{Vol}(n, n/2) \in \Theta^*(2^n) \), whereas \( k^r < 1.733^n \). This example already gives us a deterministic 3-SAT algorithm running in \( O^*(1.733^n) \): by Observation 1.1, the two calls \( \text{Src-ball}(F, 0^n, n/2) \) and \( \text{Src-ball}(F, 1^n, n/2) \) find a satisfying assignment, if there is one (recall that \( \{0^n, 1^n\} \) is an \( (n/2) \)-covering code!).
1.2. Satisfiability and Covering Codes

Of course, we can do better by choosing \( r = \rho n \) carefully. Suppose we are already given a covering code \( \mathcal{C} \) of minimal size, i.e.,

\[
|\mathcal{C}| = \frac{2^n}{\text{Vol}(n, \rho n)} \approx 2^{n-H(\rho)n}.
\]

Then, \( \forall \alpha_x \in \mathcal{C} \), we call \( \text{Src-ball}(F, \alpha_x, \rho n) \). The total running time is

\[
2^{(1-H(\rho)n)} \cdot k^{\rho n} = 2^{\left(1-H(\rho)+\rho \log_2(k)\right)n}.
\]

To minimise the exponent, we consider the derivative with respect to \( \rho \):

\[
\frac{\partial(1-H(\rho)+\rho \log_2(k))}{\partial \rho} = \log_2(\rho) + \log_2(1-\rho) + \log_2(k)
\]

\[
= \log_2 \left( k \frac{\rho}{1-\rho} \right).
\]

Setting the derivative to zero, we find that the minimum of the function is reached at \( \rho = 1/(k+1) \). By plugging it into the running time formula, we obtain:

**Theorem 1.2** Given a covering code of length \( n \), radius \( \frac{n}{k+1} \) and minimal size, Dantsin et al.’s algorithm solves \( k\text{-SAT} \) in time \( O^*(\left(2 - \frac{2}{k+1}\right)^n) \).

However, as the reader might have learned by now, we do not know whether the optimal code \( \mathcal{C} \) can at all have minimal (packing) size, let alone how to construct it. Hence, Dantsin et al. consider *approximately* optimal codes, and suggest two possible ways of constructing one.

The first, from [3], is the Monte Carlo approach of drawing random words from \( \{0, 1\}^n \); it produces *w.h.p.* a code with size away from the sphere covering bound by a polynomial factor. We will thoroughly discuss this technique in Section 3.2.

The second approach applies a greedy algorithm designed in [8] for the Set Cover problem to covering code construction. The algorithm works by iteratively choosing a ball containing the maximum number of yet uncovered elements, and outputs a code of size \( O(n \cdot 2^{(1-H(\rho))n}) \); albeit, the running time is upper bounded by an astonishing \( 2^{3n} \text{poly}(n) \). Dantsin et al. overcome this issue with a strategy well known in coding theory, called *blockwise direct sum*. Namely, given two codes \( \mathcal{C}_1 \) of radius \( r_1 \) and \( \mathcal{C}_2 \) of radius \( r_2 \), the code generated by their direct sum

\[
\mathcal{C}_3 = \{(c_1 \oplus c_2) | c_1 \in \mathcal{C}_1, c_2 \in \mathcal{C}_2\}
\]

has radius \( r_1 + r_2 \). Hence, by generating a smaller code of size \( \Theta^*(2^{(1-H(\rho))n/d}) \) in time \( 2^{3n/d} \text{poly}(n) \), and then constructing the blockwise direct sum of the code with itself \( d \) times, we obtain that
Lemma 1.3 A covering code of length $n$, radius $\rho n$, and size $2^{(1-H(\rho))n}\text{poly}_d(n)$ can be built in time $O(2^{2n/d}\text{poly}_d(n))$, where $\text{poly}_d(n)$ is a polynomial of degree $d$.

As a concatenation factor $d$, Dantsin et al. choose $\frac{3}{1-H(\rho)}$, so that the time needed for the construction is as large as the final code’s size.

In Theorem 1.2, we “hide” the polynomials resulting from this concatenation (of possibly high degree) in the Theta-star notation.

This result was improved upon in [16], where Moser and Scheder show a full derandomisation of Schöning’s algorithm, achieving a runtime of $O^*(2^{2/k})^{n+o(n)}$.

In Chapter 3 we will present a Las Vegas algorithm for covering code construction that also employs blockwise direct sum, but achieves a more reasonable code size: the concatenation factor $d$ shrinks by three times, leading to a polynomial of degree $\frac{1}{1-H(\rho)}$. Although our algorithm generates a larger code than the Monte Carlo approach, it ensures that it is actually covering: the gamble is on the running time, which we will make sure stays low enough, in expectation.

The most natural way to guarantee that a given code is $r$-covering is by checking whether its covering radius is at most $r$. Therefore, in the next chapter, we will focus on analysing the Covering Radius problem.
Chapter 2

The Covering Radius Problem

2.1 Introduction

The Covering Radius problem (CRP) asks, given a covering code $C$, to determine its covering radius. The complexity of this problem was firstly studied for a particular class of binary codes called linear codes, in [15] (1984). For these codes, the problem was proven to be extremely hard relatively to the input size: the code is given as a polynomial-sized system of linear inequalities, which can give rise to exponentially many codewords. Hence, for these codes, not only the problem is $\mathcal{NP}$-hard, but $\Pi_2$-complete, which was shown via reduction from a famous problem in the $\Pi_2$ class: Quantified-3-SAT, the 3-SAT problem with "$\forall$" and "$\exists$" quantifiers applied to the formula’s variables. $\Pi_2$ is the second level of the polynomial hierarchy described in [18]; in a nutshell, $\Pi_0 = \mathcal{P}$, $\Pi_1 = \text{co-\mathcal{NP}}$, and $\Pi_{k+1}$ is defined as the class of problems solvable in nondeterministic polynomial time with a machine having access to a subroutine able to compute the solution of a problem in $\Pi_k$ in a single step.

In [7] (2005), it has been proven that also estimating the covering radius of linear covering codes is $\Pi_2$-hard up to a factor $a_0 \in (1,2)$, and is $\mathcal{NP}$-hard for any constant factor. It becomes solvable in polynomial time only for factors larger than $\log_2(n)$.

It took more than ten years from the CRP result on linear codes for a complexity study of the problem for general binary codes to see the light, in Frances and Litman’s paper [6] from 1997, where $\mathcal{NP}$-completeness of the CRP is shown via reduction from 3-SAT.

In this chapter, we present their proof, and introduce a new Satisfiability problem equivalent to the CRP, which we thoroughly analyse in pursuit of uncovering new complexity results for the original problem. We furthermore develop an algorithm for the CRP, since none was ever presented in
the literature, and we will need it in the next chapter.

We will, in this chapter only, consider both the covering radius and the radius of a code, which we are about to define.

2.2 Radius and Covering Radius

We already defined the covering radius $CR(\mathcal{C})$ of a code. The radius $R(\mathcal{C})$, on the other hand, is the smallest integer $r$ such that all the codewords of $\mathcal{C}$ are contained in some $r$-ball in the Hamming space. Formally:

$$R(\mathcal{C}) = \arg \min_{r \in \mathbb{N}} \left\{ \exists c \in \mathbb{H}_n \mid \max_{d \in \mathcal{C}} (d_H(c, d)) = r \right\}.$$

Covering Radius Lower Bound and Radius Upper Bound are the decision problems to determine, given a code $\mathcal{C} \subseteq \{0, 1\}^n$ and $r \in [n]$, whether $\mathcal{C}$ has covering radius at least $r$ (CRLB), or whether $\mathcal{C}$ has radius at most $r$ (RUB).

Karpovsky, in [12], has shown that the two problems are equivalent, since for any code $\mathcal{C}$,

$$R(\mathcal{C}) + CR(\mathcal{C}) = n. \quad (2.1)$$

Proof

$$R(\mathcal{C}) + CR(\mathcal{C}) = \arg \min_{r \in \mathbb{N}} \left\{ \exists c \in \mathbb{H}_n \mid \mathcal{C} \subseteq B_n(c, r) \right\}$$

$$+ \left( \arg \max_{r \in \mathbb{N}} \{ \exists c \in \mathbb{H}_n \mid \mathcal{C} \cap B_n(c, r) = \emptyset \} + 1 \right) \quad (2.2)$$

$$= \arg \min_{r \in \mathbb{N}} \{ \exists c \mid \mathcal{C} \subseteq B_n(c, r) \}$$

$$+ \arg \max_{r \in \mathbb{N}} \{ \exists c \mid \mathcal{C} \subseteq B_n(c, n - r - 1) \} + 1 \quad (2.3)$$

$$= \arg \min_{r \in \mathbb{N}} \{ \exists c \mid \mathcal{C} \subseteq B_n(c, r) \}$$

$$+ \arg \max_{(n-r'-1) \in \mathbb{N}} \{ \exists c \mid \mathcal{C} \subseteq B_n(c, r') \} + 1 \quad (2.4)$$

$$= \arg \min_{r \in \mathbb{N}} \{ \exists c \mid \mathcal{C} \subseteq B_n(c, r) \}$$

$$+ n - \arg \min_{r' \in \mathbb{N}} \{ \exists c \mid \mathcal{C} \subseteq B_n(c, r') \} - 1 + 1$$

$$= n.$$

In (2.2) we express the covering radius as the maximum $r$ such that there exists a word in $\mathbb{H}_n$ at distance larger than $r$ from all the codewords, plus one; (2.3) is a consequence of the fact that $\forall r, c$, the two balls $B_n(c, r)$ and $B_n(c, n - r - 1)$ pack the $n$-dimensional Hamming space (since any $d \in \mathbb{H}_n$ having distance greater than $r$ from $c$, has distance smaller than $n - r$ from $c$); in (2.4), we replace $r$ with $n - r' - 1$. \qed
2.3 \( \mathcal{NP} \)-Completeness

2.3.1 Overview

In [6], Frances and Litman prove that the RUB problem is \( \mathcal{NP} \)-complete, which, via Karpovsky’s equation (2.1), implies \( \mathcal{NP} \)-completeness of CRLB.

We will show their proof, consisting in a polynomial time reduction from a 3-SAT formula \( F \) to a RUB instance \( \mathcal{C} \) such that \( |\mathcal{C}| = |F| + O(n) \); we can immediately deduce that since there are at most \( \binom{n}{3} \cdot 2^3 \in O(n^3) \) many 3-clauses on \( n \) variables, even for codes of polynomial size, any algorithm for RUB needs exponential time, assuming that any algorithm for 3-SAT needs exponential time (we will delve deeper into this assumption, called \textit{Exponential Time Hypothesis}, in Section 2.5.1).

2.3.2 Preliminaries

First of all, we need a few definitions. We will consider the space of binary vectors of even length, i.e., \( \mathbb{H}_{2n} \) for \( n \in \mathbb{N} \).

For \( 1 \leq i \leq j \leq 2n \) and \( \mathbf{c} = (c_1 \ldots c_{2n}) \in \{0,1\}^{2n} \), we denote by \( c_i^j \) the word \( (c_i \ldots c_j) \). A block of a word \( \mathbf{c} \) is a subset of \( \mathbf{c} \) in the form \( c_{2i-1}^{2i-1} \); a block \( c_{2i-1}^{2i-1} \) is a repetition if \( c_{2i-1} = c_{2i} \); a word is doubled if it has even length and all its blocks are repetitions.

We also define \( \mathbf{zo}(i) \), the zero-one vector of length \( 2i \), as \((01 \ldots 01) \in \mathbb{H}_{2i}\). Finally, let

\[
Y_{2n}^1 = \left\{ (01^{n-1} \mathbf{zo}(n-1)) , (10^{n-1} \mathbf{zo}(n-1)) , (01^{n-1} \overline{\mathbf{zo}}(n-1)) , (10^{n-1} \overline{\mathbf{zo}}(n-1)) \right\} .
\]

The role of this set will soon become clear.

\textbf{Lemma 2.1} The first block of the centre of any \( n \)-ball containing \( Y_{2n}^1 \) is a repetition.

\textbf{Proof} For \( \mathbf{c} = (c_1 \ldots c_{2n}) \in \{0,1\}^{2n} \),

\[
Y_{2n}^1 \subseteq B_{2n}(\mathbf{c}, n) \iff \forall \mathbf{y} \in Y_{2n}^1, d_H(\mathbf{c}, \mathbf{y}) \leq n
\iff \forall \mathbf{y} \in Y_{2n}^1, d_H(c_1^2, y_1^2) + d_H(c_3^{2n}, y_3^{2n}) \leq n
\]

Since for an \( \mathbf{x} \in \{\mathbf{zo}(n-1), \overline{\mathbf{zo}}(n-1)\} \), \( d_H(c_3^{2n}, \mathbf{x}) \geq n - 1 \),

\[
Y_{2n}^1 \subseteq B_{2n}(\mathbf{c}, n) \iff \forall \mathbf{y} \in Y_{2n}^1, d_H(c_1^2, y_1^2) \leq 1
\iff c_1 = c_2. \quad \square
\]
The Covering Radius Problem

Now, for \( j \in \{2, \ldots, n\} \), let \( Y_j^{2n} = s_j(Y_1^{2n}) \), where \( s_j \) is the circular right shift of \( 2j - 2 \) bits; that is, for \( c = (c_1 \ldots c_{2n}) \),

\[
s_j(c) = (c_{2n-2j+3} \ldots c_2c_1c_2 \ldots c_2n-2j+2).
\]

Then:

**Lemma 2.2** The centre \( c \in H_{2n} \) of any \( n \)-ball is a doubled vector if and only if it contains the set \( Y_2^{2n} = \bigcup_{j=1}^{n} Y_j^{2n} \).

**Proof** If \( c \) is doubled, consider a (repetition) block \( c_{2j-1}c_{2j} \) of \( c \). Since any block of a word of \( Y_2^{2n} \) is a non-repetition, we have that

\[
\forall i \in \{1, \ldots, n\}, \forall y \in Y_2^{2n}, \quad d_H(c_{2i-1}c_{2i}, y_{2i-1}y_{2i}) = 1,
\]

hence \( d_H(c, y) = n \).

Conversely, by repeating Lemma 2.1’s proof for the \( j \)th block of the centre \( c \) of an \( n \)-ball containing \( Y_j^{2n} \), it holds that \( c_{2j-1} = c_{2j} \). The centre of an \( n \)-ball containing \( Y_2^{2n} \) is therefore doubled. \( \square \)

Furthermore, \( Y_2^{2n} \) has size \( 4n \) and can be constructed in polynomial time by a simple algorithm.

### 2.3.3 Frances and Litman’s Proof

In a nutshell, the reduction from 3-SAT to RUB that we are about to present works as follows:

1. Every clause of the formula on \( n \) variables becomes a codeword of the code of length \( 2n \);
2. The centre of an \((n+1)\)-ball containing the set \( Y_{2(n+1)}^{2n} \) encodes an assignment satisfying the formula if and only if every codeword is contained in the ball.

Given a 3-SAT formula \( F \) on \( n \) variables, we reduce \( F \) to an instance of RUB \( \mathcal{C} \) as follows: for every clause \( D \in F \), we add to \( \mathcal{C} \) the vector \( d = (d_1 \ldots d_{2n}) \) blockwisely defined by

\[
\forall i \in \{1, \ldots, n\}, \quad d_{2i-1}d_{2i} := \begin{cases} 
00, & \text{if } x_i \in D \\
11, & \text{if } x_i \notin D \\
01, & \text{otherwise}
\end{cases}
\]

Notice that exactly three blocks of \( d \) are repetitions, corresponding to the three variables of the clause it reduces from. We now prove that a codeword defined this way is contained in an \((n+1)\)-ball whose centre is doubled if and only if the centre of this ball encodes an assignment satisfying the clause corresponding to the codeword.
2.3. $\mathcal{NP}$-Completeness

**Lemma 2.3** Let $DB(c) : H_n \mapsto H_{2n}$ be the function doubling every bit of its argument, i.e., $DB(c) = (c_1c_1c_2c_2\ldots c_nc_n)$, and let $d \in H_{2n}$ be a vector derived from a 3-clause $D \in F$ as described above. Then, for any $\alpha \in H_n$,

$$d \in B_{2n}(DB(\alpha), n + 1) \iff \alpha \text{ satisfies } D.$$

**Proof** Assume w.l.o.g. that $D = \{x_1, x_2, x_3\}$, thus $d = (1111110101\ldots 01) \in H_{2n}$. Since $DB(\alpha)$ is doubled, it has distance 1 to each of the last $n - 3$ blocks of $d$. Hence, $d \in B_{2n}(DB(\alpha), n + 1)$ if and only if

$$d_H(d|_1^n, DB(\alpha)|_1^n) \leq 4 \iff \exists i \in \{1, 2, 3\} \ s.t. \ d_{2i-1}d_{2i} = DB(\alpha)|_{2i-1}^{2i},$$

which by construction happens if and only if

$$\exists i \in \{1, 2, 3\} \ s.t. \ x_i = \alpha_i,$$

that is, $D$ contains a satisfied literal. □

From Lemma 2.3, we immediately get that

**Corollary 2.4** For a 3-SAT formula $F = \{D_1, \ldots, D_m\}$ and $\alpha \in \{0, 1\}^n$,

$$\{d_1, \ldots, d_m\} \subseteq B_{2n}(DB(\alpha), n + 1) \iff \alpha \text{ satisfies } F.$$

We are finally ready to present the main result of Frances and Litman’s paper.

**Theorem 2.5** The Radius Upper Bound problem is $\mathcal{NP}$-complete.

**Proof** Clearly, RUB $\in \mathcal{NP}$: the centre $c$ of an $r$-ball containing the input code is a linear time verifiable certificate for the radius being at most $r$.

To prove $\mathcal{NP}$-hardness, we reduce 3-SAT to RUB. Given $F = \{D_1, \ldots, D_m\}$ over $n$ variables, we build the code

$$\mathcal{C} = \{d_1 \sim 11, \ldots, d_m \sim 11\} \cup Y_{2(n+1)}$$

in polynomial time from the formula $F$. Now, we want to prove that

$$R(\mathcal{C}) \leq n + 1 \iff F \text{ is satisfiable.}$$

For the first direction, let $c \in H_{2(n+1)}$ be the centre of an $(n + 1)$-ball containing $\mathcal{C}$. Since $Y_{2(n+1)} \subseteq B_{2(n+1)}(c, n + 1)$, by Lemma 2.2 $c$ is doubled. Thus, there exists an $\alpha \in \{0, 1\}^n$ such that $c|_{2n}^{2n} = DB(\alpha)$. We can let $c|_{2n+1}^{2n+2} = 11$, minimising the overall distance: it increases by 1 only with respect to the set $Y_{2(n+1)}$. Therefore,

$$\{d_1, \ldots, d_m\} \subseteq B_{2n}(DB(\alpha), n + 1),$$
and by Corollary 2.4, $\alpha$ satisfies $F$.

For the second direction, let $\alpha \in \{0,1\}^n$ be a satisfying assignment for $F$. Since $DB(\alpha)$ is doubled, by Lemma 2.2,

$$Y_{2(n+1)} \subseteq B_{2(n+1)}(DB(\alpha) \sim 11, n + 1).$$

Furthermore, by Corollary 2.4 and appending the two bits 11 to every codeword not in $Y_{2(n+1)}$,

$$\{d_1 \sim 11, \ldots, d_m \sim 11\} \subseteq B_{2(n+1)}(DB(\alpha) \sim 11, n + 1).$$

Hence, $C \subseteq B_{2(n+1)}(DB(\alpha) \sim 11, n + 1)$, or equivalently, $R(C) \leq n + 1$. \qed

We immediately derive the complexities of the problems related to RUB.

**Corollary 2.6** The RLB and CRUB problems are co-$NP$-complete, and the CRLB problem is $NP$-complete.

### 2.3.4 Extending the Proof

The strategy employed by Frances and Litman to reduce clauses of a 3-CNF formula to words in $H_{2n}$ can be extended to $k$-CNF formulas, for any $k$: when reducing a $k$-clause to a codeword of length $2n$, we get $k$ repetition blocks (corresponding to the literals in the clause) and $n - k$ non-repetition blocks in it. The maximum Hamming distance allowed from any doubled satisfying assignment to the codewords constructed from $F$ is $n - k$ for non-repetition blocks, and $2k - 2$ for satisfying one repetition block out of $k$, hence the total maximum distance is $n + k - 2$. Theorem 2.5’s proof works in the same way, only the code differs slightly: we check whether the following code of length $2(n + k - 2)$ has radius at most $n + k - 2$:

$$C = \{d_1 \sim 1^{2k-4}, \ldots, d_m \sim 1^{2k-4}\} \cup Y_{2(n+k-2)}.$$ 

The padding on the codewords derived from the formula’s clause is larger than in the 3-SAT case, as the set $Y$ needs to have length double as large as the radius we are checking, in order to make sure that the centre of the ball is doubled. The reduction obviously still takes polynomial time and space. Later, as we will try to establish a conditional lower bound on the Radius problem, this extension of the proof will find a purpose.

### 2.3.5 Open Problems

The proof we presented works from 3-SAT to RUB on $O(poly(n))$ codewords and with radius $n/2$. What happens to the complexity of RUB when given a different radius value? Frances and Litman do not answer this question; we will try to, with the help of an equivalent problem, which we will now define.
2.4 A New Satisfiability Problem

2.4.1 \( r\)-n-SAT Definition

Starting from the natural idea of correspondence between assignments on a variable set and binary vectors, we can identify a third problem equivalent to both the CRLB and the RUB problems. Namely, the following Satisfiability problem:

\( r\)-n-SAT:

**Input:** A Boolean formula \( F \) in conjunctive normal form over \( n \) variables, where each clause has size \( n \); \( r \in \llbracket n \rrbracket \).

**Output:** The answer to: does there exist an assignment on the \( n \) variables of \( F \) that satisfies at least \( r \) literals in each clause?

**Theorem 2.7** \( r\)-n-SAT is equivalent to the Covering Radius Lower Bound problem.

**Proof** We will show that \( r\)-n-SAT is equivalent to RUB with radius \((n - r)\), which, via Karpovsky’s equation (2.1), implies that \( r\)-n-SAT is equivalent to the CRLB problem with covering radius \( r \).

First, we take a preliminary step by mapping each \( n \)-clause to a binary vector in the natural way. That is, once we defined an order of the variables of the formula \( F \), we construct a code \( \mathcal{C} \) containing a codeword for each clause of \( F \), as follows: a 0 (1) in position \( i \) of codeword \( j \) means the clause \( C_j \) contains the literal \( \overline{x}_i \)(\( x_i \)). We obtain a bijective mapping from the set of all possible \( n \)-clauses to the \( n \)-dimensional Hamming space.

Now, suppose we are given an assignment \( \alpha \in \{0, 1\}^n \) satisfying at least \( r \) literals in each clause of \( F \). If we consider binary vectors associated to the clauses of \( F \), each vector intersects \( \alpha \) on at least \( r \) positions, thus each clause corresponding to a codeword of \( \mathcal{C} \) contains at least \( r \) literals satisfied by \( \alpha \), or equivalently, \( \mathcal{C} \subseteq B_n(\alpha, n - r) \).

For the opposite direction, assume we know that \( \mathcal{C} \) is contained in an \((n - r)\)-ball centred in \( \alpha \in \{0, 1\}^n \). Then,

\[ \forall \mathbf{c} \in \mathcal{C}, \; d_H(\mathbf{c}, \alpha) \leq n - r. \]

Each codeword intersects with \( \alpha \) on at least \( r \) positions, thus each clause corresponding to a codeword of \( \mathcal{C} \) contains at least \( r \) literals satisfied by \( \alpha \). □

In pursuit of finding an algorithm for solving \( r\)-n-SAT, we developed an algorithm for Covering Radius Upper Bound as well, using \( O(|\mathcal{C}| \cdot \text{Vol}(n, r)) \) time, with the downside of consuming \( O(n2^n) \) bits of space. The algorithm works by reducing the input to a normal satisfiability problem; before presenting it, let us briefly recall a relevant Lemma presented in [19, Chapter 2.1].
Lemma 2.8 If, for a formula $F$ in conjunctive normal form
\[ \sum_{C \in F} 2^{-|C|} < 1, \]
$F$ is satisfiable.

Proof Let $n$ be the number of variables of $F$, $\alpha \in \{0, 1\}^n$ be an assignment drawn uniformly at random, and $X_{C, \alpha}$ the indicator random variable for the event “$C$ is not satisfied by $\alpha$”. For any CNF formula $F$ it holds that
\[
\mathbb{E}(\text{number of clauses dissatisfied by } \alpha) = \mathbb{E}\left( \sum_{C \in F} X_{C, \alpha} \right) \\
= \sum_{C \in F} \mathbb{E}(X_{C, \alpha}) \\
= \sum_{C \in F} 2^{-|C|}.
\]
If the sum evaluates to less than 1, by the probabilistic method there exists an assignment satisfying every clause. \hfill \Box

In the case of an $n$-CNF formula, the Satisfiability problem is a mere matter of counting:

Corollary 2.9 A formula $F$ in conjunctive normal form, in which each clause has size equals to the number of variables $n$, is satisfiable if and only if $|F| < 2^n$.

2.4.2 An Algorithm for $r$-n-SAT

A trivial algorithm for $r$-n-SAT consists in checking, for each assignment, whether it satisfies at least $r$ literals in each clause. This approach takes $O^*(2^n \cdot m)$ time, which can be as big as $O^*(2^{2n})$, but no additional space.

We now present a faster algorithm, running in $O^*(2^n)$ for the instances we care mostly about, and consuming as much space. The idea of the algorithm is to reduce an $r$-n-SAT instance to a CNF-SAT instance where all clauses have size $n$, and which is satisfiable if and only if the decision problem $r$-n-SAT should return “yes” on the original instance. Then, by Corollary 2.9, simply counting the number of clauses will give us the solution.

We thus reduce our $r$-n-CNF input $F$ to an $n$-SAT formula $\hat{F}$ as follows: for each clause $C_i \in F$, we add to $\hat{F}$ the set $\hat{C}_i$ consisting of all clauses with up to $(r - 1)$ flipped literals with respect to $C_i$. Then, if the “best” assignment satisfies only $j \in \{0, \ldots, r - 1\}$ literals in some clause, we are going to add the clause with exactly those $j$ flipped literals to the formula $\hat{F}$, thus adding an unsatisfied clause. If, on the contrary, that clause is satisfied by $r$ or more literals, flipping any subset of size at most $(r - 1)$ can not unsatisfy it.
From a coding theory point of view, considering $C_i$ as a binary vector, it holds that
\[ \hat{C}_i = B_n(C_i, r-1). \]
This operation takes $\Theta^*(|F| \cdot \text{Vol}(n, r))$ time. Afterwards, we just have to check whether $\hat{F}$ consists of all possible $n$-clauses. If it does, by Corollary 2.9 we return “no”; otherwise we return “yes”.

We can easily translate the algorithm in the CRLB terms: we solve the problem by checking whether the union of the $r$-balls centred on the codewords covers the Hamming space. If the algorithm outputs a negative answer, the balls of radius $r-1$ around the codewords cover the Hamming space, hence the covering radius is not lower bounded by $r$. Otherwise, $r$ is a legit lower bound for the covering radius of the code.

**Lemma 2.10** There exists an algorithm solving the Covering Radius Lower Bound problem on a covering code $\mathcal{C}$ of length $n$ and radius $r$ in $\Theta^*(|\mathcal{C}| \cdot \text{Vol}(n, r-1))$ time and space.

But how fast is the algorithm, as a function of $n$?

We know that in order for the question to make sense, by the sphere covering bound, we need $|\mathcal{C}| \geq \frac{2^n}{\text{Vol}(n, r)}$, therefore the running time of the algorithm, for non-trivial instances, is at least $O^*(2^n)$. In the case of the Las Vegas algorithm we will show in the next section, we achieve such minimum (up to a multiplicative factor linear in $n$).

We would like to prove definitely that this running time can not be improved upon. To do so we will discuss the possibility, later in this chapter, of developing a conditional lower bound for $r$-$n$-SAT. Now, we will consider the hardness of the problem as the parameter varies, in pursuit of getting a better understanding of its structure.

### 2.4.3 Hardness

We shall now analyse the complexity of $r$-$n$-SAT for all possible $r$. Let $m := |F|$. Then:

1. For $r = 0$, the problem is trivial: we always return “yes” (we do not admit the possibility of an empty clause being in $F$, thus any assignment satisfies at least 0 literals in each clause).

2. For $r = 1$, by Corollary 2.9 we only need to count whether $m < 2^n$, and return “yes” in such a case. Thus the problem is solvable in time linear in the input size.

3. For $r \in O(1)$, thanks to Lemma 2.10, we can solve the problem in time $O(m \cdot \text{poly}(n))$. 
4. For $r = n/2$, by [6], the problem is $\mathcal{NP}$-complete. Their reduction works from 3-SAT on $n$ variables to the Radius Upper Bound problem on vectors of length $2n + 2$ and radius $n + 1$.

5. For $n/2 < r \leq n - \omega(1)$, the problem is $\mathcal{NP}$-complete, by simply padding all the clauses of a $(n/2)$-n-SAT instance with additional positive literals. For $k$ the number of literals we add to every clause, we get an instance of $(n/2 + k)-(n + k)$-SAT, hence the ratio of the radius w.r.t. the number of variables is $\frac{n + 2k}{2(n + k)}$, which we can get as close to 1 as we wish, albeit the radius is still smaller than the number of variables by a factor of $n/2 \in \omega(1)$. The formula we obtain via this padding contains at least $n/2 + k$ satisfied literals if and only if the original formula contains at least $n/2$, since the assignment mapping every additional variable to 1 satisfies $k$ literals in each clause.

6. For $r = n - z$, $z \in O(1)$, first of all we check in polynomial time whether $m > \text{Vol}(n,z)$, in which case we return “no” right away: as we saw in Theorem 2.7’s proof, the formula, expressed as binary vectors, needs to be contained in a Hamming ball of radius $z$. Once we made sure $m$ is small enough, we try every partial assignment $\alpha_{\text{par}}$ satisfying $r$ literals in the first clause (there are $\binom{n}{r} = n^{O(1)}$ many) and check whether the rest of the clauses can be satisfied by an assignment $\alpha$ containing $\alpha_{\text{par}}$. The total running time is $O(\text{poly}(n))$, hence the problem is in $\mathcal{P}$.

7. For $r = n$, we just check whether the formula has at most one clause (return “yes”) or more (return “no”), in constant time.

The gap for $\omega(1) \leq r < n/2$ in the complexity analysis is yet to be filled.

In order to understand the problem in the best way possible, and to see how it relates to the classical $k$-SAT problem, we also outline the complexity analysis of $r$-$k$-SAT for $k \in \Theta(1)$:

1. For $r = 0$, we always return “yes”.

2. For $r = 1$, the problem corresponds to $k$-SAT (is there an assignment satisfying at least one literal in each clause?), and is thus $\mathcal{NP}$-complete for every $k \geq 3$.

3. For $r = k$, the problem is in $\mathcal{P}$: to solve it, we perform a linear scan of the clause set: upon finding a literal and its negation, we return “no” (because all literals in each clause of the formula need to be satisfied), otherwise we return “yes”.

The rest of the range needs to be looked into specifically. We shall start with $r$-3-SAT formulas, since $r$-2-SAT formulas are “easy” by the observations we just made.
For $r \in \{0, 3\}$, $r$-3-SAT is easy. On the other hand, 1-3-SAT is equivalent to 3-SAT and thus hard. What about 2-3-SAT?

**Lemma 2.11** $2$-$3$-SAT $\in \mathcal{P}$.

**Proof** We show a polynomial time reduction from an instance $F$ of 2-3-SAT on $n$ variables and $m$ clauses to a 2-SAT instance $F^*$ on $n$ variables and up to $3m$ clauses, which is known to be in $\mathcal{P}$. For every clause $C \in F$, we add to $F^*$ the set of clauses

$$C^* = \{ D \mid D \subset C, |D| = 2 \},$$

the 3 subsets of size 2 of $C$.

For an assignment $\alpha$, let

$$A := \"\alpha\text{ satisfies at least 2 literals in every clause of } F\"$$

$$B := \"\alpha\text{ satisfies } F^* \".$$

We will now show that $A \iff B$ and thus prove the validity of the reduction.

For the $A \Rightarrow B$ direction, consider an assignment $\alpha$ satisfying less than two literals in some clause $C = \{x_1, x_2, x_3\}$ of $F$. By construction, $F^*$ contains the three clauses $C^* = \{ \{x_1, x_2\}, \{x_2, x_3\}, \{x_1, x_3\} \}$. If only one literal (say $x_1$) evaluates to true, then the clause of $C^*$ not containing it (in this example, $\{x_2, x_3\}$) evaluates to false. If no literal of $C$ evaluates to true, none of the clauses in $C^*$ do.

For the $A \Rightarrow B$ direction, notice that if at least two literals in any 3-clause $C_i$ are satisfied by $\alpha$, any subset of size 2 of $C_i$ contains a literal satisfied by $\alpha$, which hence satisfies $C_i^*$.

The same approach can be extended to $(k - 1)$-$k$-SAT for any $k$: an assignment $\alpha$ satisfies all but at most one literal in every clause of the formula if and only if the set of all pairs of literals of $C$ is satisfied by $\alpha$. The reduction still takes time polynomial (squared) in the input size $m$.

**Corollary 2.12** $(k - 1)$-$k$-SAT $\in \mathcal{P}$.

Now $r$-3-SAT is analysed completely. The next combination to look into is 2-4-SAT.

**Lemma 2.13** 2-4-SAT is $\mathcal{NP}$-complete.

**Proof** We will again employ a reduction, this time from 3-SAT on $n$ variables and $m$ clauses to 2-4-SAT on $n + 1$ variables and $m$ clauses. The idea is to pad every 3-clause $C$ with a positive literal $x_p$, obtaining a 4-clause $C'$; any assignment on $\{x_1, \ldots, x_n, x_p\}$ having $x_p = 1$ satisfies one literal in every
C′ ∈ F′. Thus, any clause C′ contains at least two satisfied literals if and only if C contains at least one: F′ is a “yes” instance for 2-4-SAT if and only if F is a “yes” instance for 3-SAT. Moreover, a “yes” assignment for F′ (ignoring the variable x_p) is a satisfying assignment for F and vice versa. □

The same approach can be applied, in fact, to 2-k-SAT formulas for any k.

**Corollary 2.14** For every k ≥ 4, 2-k-SAT is \( \mathcal{NP} \)-complete.

**Proof** We can extend the reduction from Lemma 2.13 to work from a \((k-1)\)-SAT formula F on n variables to a 2-k-SAT formula F′ on n + 1 variables consisting of every clause of F padded with the literal x_p. By the same argument as in Lemma 2.13’s proof, F′ is a “yes” instance for 2-k-SAT if and only if F is a “yes” instance for \((k-1)\)-SAT. □

Using what we learned up to now, we are able to shed light on the remaining values of r and k for the problem.

**Theorem 2.15** For every k ≥ 4 and 2 ≤ r ≤ k − 2, r-k-SAT is \( \mathcal{NP} \)-complete.

**Proof** Once again, we proceed by reduction, from a \((k-r+1)\)-SAT formula F on n variables to a r-k-SAT formula F′ on n + r − 1 variables.

We reduce every clause \( C = \{x_1, \ldots, x_{k-r+1}\} \in F \) to a clause \( C′ = \{x_1, \ldots, x_{k-r+1}, x_{p,1}, \ldots, x_{p,r-1}\} \) where the last r − 1 literals, if evaluated to 1, will be satisfied in every clause. As before, C′ contains at least r satisfied literals if and only if C contains at least one. □

We finally closed the “window”: r-k-SAT is always easy if k ≤ 2; for larger ks, it is easy when either r ≥ k − 1 or r = 0.

![Figure 2.1](image-url)
We immediately notice that the complexity of \( r-k \)-SAT in the \( k \in O(1) \) case differs significantly from the \( k = n \) case, which is easy for any \( r = n - O(1) \). The next step in our quest towards understanding of \( r-n \)-SAT is to analyse the \( r-k \)-SAT problem for \( k \in \omega(1) \).

For \( k \in O(n) \) but smaller than \( n \), Theorem 2.15 and Corollary 2.12 still hold, as we can employ the same strategies and obtain a polynomial number of clauses and of steps in both cases. However, one essential constraint makes our strategy inapplicable to the original \( r-n \)-SAT: the requirement that every clause contains every variable.

More precisely, the fundamental problem we need to face, upon designing a reduction from \( (<n) \)-SAT to \( r-n \)-SAT, is how to translate a \( (<n) \)-clause \( C \) to an \( n \)-clause (or a set of \( n \)-clauses) \( C' \) which contains at least \( r \) satisfied literals if and only if \( C \) is satisfied: with which parity do we add the literals corresponding to the variables missing from \( C \)? This question remains unanswered.

### 2.5 Conditional Lower Bounds

#### 2.5.1 The Strong Exponential Time Hypothesis

Impagliazzo and Paturi’s Exponential Time Hypothesis (ETH) [9] asserts that 3-SAT can not be solved in time \( 2^{o(n)} \), where \( n \) is the number of variables in the input formula. Since ETH does not seem to be sufficient for estimating the best possible growth rate for \( \mathcal{NP} \)-complete problems, it is common practice, in modern complexity theory, to consider the Strong Exponential Time Hypothesis (SETH) described in [10]. The SETH asserts the following:

**Assumption 2.16 (SETH)** For some (large) constant \( k = k(\delta) \in \Theta(1) \), the \( k \)-SAT problem can not be solved in \( 2^{\delta n} \) time, for any \( \delta < 1 \).

SETH is widely used for designing conditional lower bounds on the running time of algorithms, following this scheme: given a problem \( \mathcal{P} \), for which we believe any algorithm needs at least \( O(s) \) time, reduce \( k \)-SAT or CNF-SAT to \( \mathcal{P} \), taking care of the time and space needed by the reduction; if solving \( \mathcal{P} \) in \( o(s) \) implies the reduction would solve \( k \)-SAT or CNF-SAT in \( 2^{o(n)} \), then, under SETH, any algorithm for \( \mathcal{P} \) needs indeed at least \( O(s) \) time.

#### 2.5.2 A Lower Bound on the Running Time for \( r-n \)-SAT

The extension of Frances and Litman’s proof we presented in Section 2.3.4 provides us a conditional lower bound under the Strong Exponential Time Hypothesis: since the reduction works from a \( k \)-SAT formula (for any \( k \)) on \( n \) variables and \( m \) clauses to a code of length \( 2n \) and size \( O(m \cdot n) \), we can
prove a conditional lower bound of $O(m \cdot 2^{n/2})$ under SETH. We also get a proof for the fact that, in the same way as $k$-SAT is at least as hard as $(k-1)$-SAT, RUB on radius $r$ is at least as hard as RUB on radius $(r-1)$, at least for $r \geq n/2$. Unfortunately, we were not able to prove that we need $O^*(2^n)$ for solving the problem, which would imply that our algorithm from Lemma 2.10 can achieve the best possible runtime.

2.6 Conclusion

It remains an open problem which running time is required for solving the RUB, CRLB and $r$-n-SAT problems, under SETH. However, we managed to analyse with more depth the behaviour of these problems for varying parameters, and we will use the algorithm we presented for CRLB in the upcoming section, to transform the covering code constructor Monte Carlo algorithm into a Las Vegas algorithm.
Chapter 3

A New Las Vegas Algorithm

3.1 Introduction

In this chapter, we present the main contribution of this thesis: a Las Vegas algorithm that not only is expected to produce an asymptotically optimal covering code in optimal time, but whose expected number of loops to do so is two.

The algorithm combines two fundamental ideas from the covering codes literature:

1) The “classical” Monte Carlo randomised approach for covering code construction from [3] that consists in drawing random elements in \{0, 1\}^n; for an \Omega(n) multiplicative factor away from the minimum size possible for the code (provided by the sphere covering bound), the probabilistic method shows that a covering code of such size exists.

2) Blockwise linear sum of codes in a similar fashion as in Dantsin et al.’s paper [5].

3.2 The Monte Carlo Algorithm

Let’s, first of all, review the Monte Carlo approach for constructing a covering code of nearly optimal size.

**Lemma 3.1** For \( \rho \in (0, \frac{1}{2}) \), there exists a Monte Carlo algorithm outputting a covering code \( \mathcal{C} \) of covering radius at most \( \rho n \) and size \( O(2^{(1-H(\rho))n} \cdot n) \) in time \( \Theta(2^{(1-H(\rho))n} \cdot n) \) with probability larger than \( \frac{1}{2} \).

**Proof** If we draw \( k \) many elements u.a.r. from \{0, 1\}^n with reinsertion for our covering code \( \mathcal{C} \), the probability that a fixed vector \( s \in \mathcal{H}_n \) is at distance
more than \( r \) from \( \mathcal{C} \) is

\[
\Pr(d_H(s, \mathcal{C}) > r) = \left(1 - \frac{\text{Vol}(n, r)}{2^n}\right)^k < e^{-k \cdot \text{Vol}(n, r)/2^n}.
\]

By choosing \( k = \frac{\ln(2)(n+1)2^n}{\text{Vol}(n, r)} \), we get

\[
\Pr(d_H(s, \mathcal{C}) > r) < e^{-\ln(2)(n+1)} = 2^{-n-1}.
\]

By union bounding over all vectors in \( \{0, 1\}^n \), we finally get that

\[
\Pr\left(\max_{s \in \mathbb{F}_n} (d_H(s, \mathcal{C})) > r\right) < 2^n \cdot 2^{-n-1} = \frac{1}{2}.
\]

Thus the complementary probability is

\[
\Pr(\mathcal{C} \text{ is } r\text{-covering}) = \Pr\left(\max_{s \in \mathbb{F}_n} (d_H(s, \mathcal{C})) \leq r\right) > 1 - \frac{1}{2}. \quad \square
\]

To turn the Monte Carlo algorithm into a Las Vegas algorithm, we could put the Monte Carlo in a loop, until we get a satisfactory result. The problem is to check whether the code is good or not: since, as we have shown in Section 2.3, the covering radius check of a code is an \( \mathcal{NP} \)-complete problem, it could take much more time than the construction itself.

The idea is, therefore, to build a smaller optimal code \( \mathcal{C}' \) of length \( \frac{n}{d} \), for a \( d \in \Theta(1) \) called concatenation factor, and then building the blockwise sum of it \( \mathcal{C} = \mathcal{C}'^d \), i.e., the code of length \( n \) made of all \( |\mathcal{C}'|^d \) (not necessarily distinct) concatenations of \( \mathcal{C}' \).

To check the covering radius of the code, we will use the algorithm we presented in Lemma 2.10; the slower but less space-consuming variant of it presented in the beginning of Section 2.4.2 might appear as a valid option, however we will choose the concatenation factor \( d \) such that we do not need to worry about the space consumption of the radius check, as it will correspond to the space taken by the code itself.

Since we have an algorithm for the Covering Radius Lower Bound problem, we will check whether the covering radius is larger than \((\rho n + 1)\) for our code, and, in case of negative answer, conclude that the code has radius at most \( \rho n \). Let us recall the running time of the covering radius check:

**Lemma 3.2** We can check whether code of length \( n \) and size \( O^* (2^{(1-H(\rho))n}) \) has covering radius at most \( \rho n \) in

\[
O^* (2^{(1-H(\rho))n} \cdot \text{Vol}(n, \rho n)) = O^* (2^n)
\]

\text{time and space.}
3.3 The Las Vegas Algorithm

As explained, the algorithm we now present works by generating a code of length \( \frac{n}{d} \) and checking if its covering radius is small enough, in which case it proceeds to generate its \( d \)-blockwise direct sum, to be returned as the final code of length \( n \). We will soon choose the most appropriate value for \( d \).

**Algorithm 2: Las Vegas code construction**

\[
\begin{align*}
\textbf{input} & : n, d \in \mathbb{N}, \rho \in (0, \frac{1}{2}). \\
\textbf{output} & : \text{An asymptotically optimal binary covering code of length } n \text{ and radius } \rho n. \\
N & \leftarrow \frac{n}{d} \\
\text{while } \text{CR}(\mathcal{C}) > \rho N \text{ do} \\
\quad \mathcal{C} & \leftarrow \frac{\ln(2)(N+1)2^{n/d}}{\text{Vol}(N, \rho n)} \text{ elements drawn u.a.r. from } \{0, 1\}^N; \\
\quad \text{check if } \text{CR}(\mathcal{C}) \leq \rho N; \\
\text{end} \\
\text{return } \mathcal{C}' = \mathcal{C}^d;
\end{align*}
\]

We assume for simplicity that \( \frac{n}{d}, \rho \frac{n}{d} \in \mathbb{N} \). By Lemma 3.2 the covering radius can be checked in time and space \( O^*(2^{n/d}) \) by computing the union of the balls around the codewords. Therefore, inside the loop, the two operations of constructing the code and checking its radius take total time

\[
O^*(2^{(1-H(\rho))n/d} + 2^{n/d}) = O^*(2^{n/d})
\]

for every iteration, which by Lemma 3.1 is independently successful with probability at least \( \frac{1}{2} \). In expectation, therefore, the loop is traversed at most twice. Algorithm 2 returns a covering code of length \( n \), radius \( \rho n \) and size

\[
|\mathcal{C}| = \left( \frac{\ln(2)(n/d + 1)2^{n/d}}{\text{Vol}(n/d, \rho n/d)} \right)^d = O\left( \left(2^{(1-H(\rho))n/d}\right)^d \cdot \left(\ln(2)(n/d + 1)\right)^d \right)
\]

\[
= O\left(2^{(1-H(\rho))n} \cdot n^d\right).
\]

If we choose as a concatenation factor \( d = \left\lceil \frac{1}{1-H(\rho)} \right\rceil \), the total running time of the algorithm corresponds to \( O^*(2^{(1-H(\rho))n} \cdot n^d) \), the size of the final code up to polynomial factors.
3. A New Las Vegas Algorithm

**Theorem 3.3** Let \( n \in \mathbb{N}, \rho \in (0, \frac{1}{2}) \). Algorithm 2 constructs a binary covering code of length \( n \), covering radius at most \( \rho n \) and size \( O(2^{(1-H(\rho))n} \cdot n^{\frac{1}{1-H(\rho)}}) \) in expected time and space \( O(2^{(1-H(\rho))n \cdot \text{poly}_\rho(n)}) \), where \( \text{poly}_\rho(n) \) is a polynomial of degree \( O\left(\frac{1}{1-H(\rho)}\right) \).

Notice that we allow the running time to be polynomially larger than the size of the code, since upon computing the running time of the radius check we leave polynomials aside; we do guarantee, however, that the size of the returned code is \( O(2^{(1-H(\rho))n} \cdot n^{\frac{1}{1-H(\rho)}}) \).

### 3.4 Estimation of the Result’s Quality

We can now look at the probability that the loop has to be traversed many more times than expected. By Lemma 3.1,

\[
\Pr(CR(\mathcal{C}) > \rho n) < \frac{1}{2},
\]

For every loop we go through. Let \( L \) be the number of loops we traverse without producing a good result. The probability that we don’t obtain a good code within \( \delta \) loops is

\[
\Pr(L > \delta) < \frac{1}{2^\delta}.
\]

By plugging in \( \delta = c \log_2 n \), for \( c \in \Theta(1) \), we obtain

\[
\Pr(L > c \log_2 n) < \frac{1}{2^{\log_2 n^c}} = \frac{1}{n^c},
\]

thus, \( w.h.p. \), we will get a good code within \( o(\log n) \) loops. Notice that we can get the probability of success arbitrarily high without affecting the size of the built code, whereas the only way to enhance the success probability in the Monte Carlo approach is to draw more elements for the code. Albeit, by increasing the number of drawings by only \( \log_2 (n^c) \), i.e., by drawing \( \frac{2^{d \ln(2)} \cdot (n + \log_2 (n^c))}{\text{Vol}(n, r)} \) many elements \( u.a.r. \) from \( \mathbb{H}_n \) for our covering code \( \mathcal{C} \), we get that

\[
\Pr\left(\max_{s \in \mathbb{H}_n} d_H(s, \mathcal{C}) > r\right) < 2^n \cdot e^{-\ln(2)(n+\log_2 (n^c))} = \frac{1}{n^c},
\]

hence the Monte Carlo approach achieves high probability of success with a much smaller number of words than Algorithm 2.

Let’s now take a look at the concatenation factor \( d \), which is where we perform (in expectation) better than the algorithm from Dantsin et al.: recall
from Lemma 1.3 that their method employs the following number of concatenations:

\[d_{Dant} = \left\lceil \frac{3}{1 - H(\rho)} \right\rceil > \left\lceil \frac{1}{1 - H(\rho)} \right\rceil =: d,\]

which is what we have. Notice that \(d\) impacts only on the polynomial multiplicative factors: we can not get better on the exponential factor, since we already achieve the minimum possible \(2^{(1 - H(\rho))n}\) given by the sphere covering bound.

The following table compares the degrees we get for the polynomials with the ones from Dantsin et al.’s greedy covering code construction from Lemma 1.3.

<table>
<thead>
<tr>
<th>(\rho)</th>
<th>(d_{Dant})</th>
<th>(d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\to \frac{1}{2})</td>
<td>(\to \infty)</td>
<td>(\to \infty)</td>
</tr>
<tr>
<td>(\frac{1}{3})</td>
<td>37</td>
<td>13</td>
</tr>
<tr>
<td>(\frac{1}{4})</td>
<td>16</td>
<td>6</td>
</tr>
<tr>
<td>(\frac{1}{5})</td>
<td>11</td>
<td>4</td>
</tr>
<tr>
<td>(\to 0)</td>
<td>(\to 3)</td>
<td>(\to 1)</td>
</tr>
</tbody>
</table>
4.1 Introduction

4.1.1 Piecewise Constant Codes Definition

In this chapter, we try to generalise a method from [2] for constructing small covering codes with constant covering radius to arbitrary length; their method is called piecewise constant construction and focuses on the Hamming weight \( w(c) \) of the codewords:

\[
\forall c \in \{0, 1\}^n, \quad w(c) = d_H(c, 0^n) = \text{“number of ones in } c\text{”}.
\]

**Definition 4.1** Given a partition of the length of the code as \( n = \sum_{i=1}^{t} n_i \), and of each codeword \( c \) in \( t \) blocks \( c_1 = (c_1 \ldots c_{n_1}), \ldots, c_t = (c_{n-n_t} \ldots c_n) \), where the length of \( c_i \) is \( n_i \), we call the code \( C \subseteq \mathbb{H}_n \) “piecewise constant” if and only if:

if \( C \) contains any vector with \( w(c_1) = w_1, \ w(c_2) = w_2, \ldots, w(c_t) = w_t \), then it contains all such vectors.

We can express piecewise constant codes as \( t \)-uples of integers denoting the weight of the blocks of the codewords.

\[
\begin{array}{cc}
00 & 000 \\
10 & 111 \\
01 & 111 \\
11 & 001 \\
11 & 010 \\
11 & 100 \\
\end{array}
\]

**Figure 4.1:** A piecewise constant code of length 5 = 2+3 described by vectors of weights: \((0,0), (1,3), (2,1)\).
4. Piecewise Linear Codes

Notice that, in this chapter, we refer to “blocks” of a codeword meaning a subset of the word in the form $c_i^j = (c_i \ldots c_j)$ extending the definition from Section 2.3.2, where a block of a codeword was $c_{2i-1}^{2i} = (c_{2i-1} c_{2i})$.

We can furthermore describe the binary Hamming space in terms of weights of its codewords by constructing a $t$-dimensional grid, where $t$ is the number of blocks, and the entry $(w_1, w_2, \ldots, w_t)$ contains the number of words (split in $t$ blocks $c_1 \ldots c_t$) with $w(c_1) = w_1, \ldots, w(c_t) = w_t$. Since the number of words of weight $w$ in the Hamming space is $\binom{n}{w}$, the entry $(w_1, \ldots, w_t)$ corresponds to $\prod_{i=1}^{t} \binom{n}{w_i}$ many binary words. In the grid, we mark the entries we pick for our covering code.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_1$</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>$w_2$</td>
<td>1</td>
<td>2</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

![Figure 4.2: The two-dimensional grid representation of the above code.](image)

The entries of the grid get larger when, from the corners (containing blocks with 0 or maximum weight, thus only one possible codeword), we approach the centre (containing the blocks whose weight is half of their length), corresponding to the highest number of elements because the function $\binom{n}{i}$ is maximum for $i = \frac{n}{2}$ and decreases monotonically and symmetrically, as we move from $i = \frac{n}{2}$ towards $i = 0$ or $i = n$.

For every word of Hamming weight $w$, there exists another word of Hamming weight $w + k$ or $w - k$ at Hamming distance $k$ from it. Therefore, in the piecewise constant setting, the Hamming distance between the sets of codewords corresponding to two $t$-uples $c = (w_1, \ldots, w_t)$ and $c' = (w'_1, \ldots, w'_t)$, is equals to the sum of the weight difference of their blocks:

$$d_M((w_1, \ldots, w_t), (w'_1, \ldots, w'_t)) = \sum_{i=1}^{t} |w_i - w'_i|.$$  

This metric is called Manhattan distance or $\ell_1$-norm. The covering radius of the code hence corresponds to the maximum distance, in the grid, from any entry to the code. In the example in Figure 4.1.1, the covering radius is 2 because the entry $(0, 2)$ has minimum Manhattan distance 2 from the entries representing the code.
4.1 Introduction

4.1.2 Piecewise Linear Codes Definition

Our investigations focus on trying to extend the piecewise constant approach to arbitrary length and linear covering radius, by splitting the length of the code in a constant number $t$ of blocks of size $\frac{n}{t}$. We will refer to covering codes built with this strategy as “piecewise linear”.

Definition 4.2 We call a code $\mathcal{C}$ “piecewise linear” if and only if it is a piecewise constant code where the length $n$ is partitioned in $t \in O(1)$ blocks of equal size.

We will investigate on how to build the optimal piecewise linear code of covering radius $\frac{n}{3}$, since we are interested in linear radii, and $c = 3$ is the smallest integer for which the sphere covering bound tells us that a code of covering radius $\frac{n}{c}$ has exponential size.

We get a grid shaped like a $t$-dimensional cube, each dimension having size $\frac{n}{t} + 1$, corresponding to the possible values of the Hamming weight of a word of length $\frac{n}{t}$. We will refer to Manhattan spheres in such grids simply as “balls”; they are not spheres, though, but cross-polytopes: in 1 dimension, we get a line, in 2 dimensions a square, in 3 dimensions a regular octahedron, etc.

We will affirm that a set of vectors $\tilde{V} = \{v_1, \ldots, v_n\}$ covers the grid meaning that the Manhattan balls of radius $\frac{n}{3}$ centred in the points of $\tilde{V}$ are a cover of the weight grid, and therefore that a piecewise linear code $\mathcal{C}$ of covering radius $\frac{n}{3}$ can be derived from $\tilde{V}$. We will sometimes directly refer to $\tilde{V}$ as “code”, since it is a univocal description of the words of $\mathcal{C}$.

It is of fundamental importance to avoid confusion between piecewise linear codes and traditional covering codes: the first ones are subsets of $\binom{\frac{n}{t}}{t}$, the second ones subsets of $H_n$. We will later outline the distinction more thoroughly.

4.1.3 Motivation

The piecewise linear approach presents many advantages:

- **Polynomial storage space for the code.** Since we deal with $t$-tuples of polynomial numbers, and we have at most $(\frac{n}{t} + 1)^t$ many for $t \in O(1)$, a piecewise linear code requires storage space at most $O(\text{poly}(n))$, even if we pic the whole grid as a code.

- **Polynomial time check of the covering radius.** For any $t$-uple and any piecewise linear code $\tilde{V}$, computing the distance from the $t$-uple to $\tilde{V}$ requires $|\tilde{V}| \cdot t = O(n^t)$ time, since computing the Manhattan distance between two $t$-tuples requires $t$ sums. Therefore, to check the covering radius of a piecewise linear code we need at most $O(n^{2t})$ time.
4. PIECEWISE LINEAR CODES

- A construction strategy independent of \( n \). If we are able to find a piecewise linear code \( \tilde{V} \subseteq \left[ \frac{n}{4} \right]^t \) such that the corresponding covering code is close to optimal size, we can construct such a code for any \( n \) by just performing a multiplication on all weights.

4.1.4 A Conjecture on the Grid Cover

Our pursuit is inspired by the following conjecture:

**Conjecture 4.3** As we enlarge \( t \) and compute the optimal cover for the \( t \)-dimensional grid \( \left[ \frac{n}{4} \right]^t \), the corresponding covering code gets smaller.

The conjecture is motivated by the two extreme cases:

- \( t = n \), i.e., every block has size 1. Then, the grid corresponds to the Hamming space: we have \( \left( \frac{n}{4} + 1 \right)^t = 2^n \) many entries, and the Manhattan distance corresponds to the Hamming distance. Hence, the optimal covering code is the best possible by definition. Thanks to the binary entropy bounds, we know that the size of the cover with covering radius \( \frac{n}{2} \) is in \( O^*(1.0582^n) \).

- \( t = 1 \). In this case, the grid has one dimension, and we can describe it with a line with values \( 0, \ldots, n \). We know that the densest element is \( n/2 \); then, in order to cover it, we place our balls as far away as possible from it, on both sides: the centres of the least costly configurations will have to be placed in the points \( \frac{n}{3} \pm \frac{n}{6} = \left\{ \frac{n}{6}, \frac{5n}{6} \right\} \).

![Figure 4.3](image-url)  
**Figure 4.3:** The most economical strategy to cover the central element.

The corresponding code is \( \frac{n}{3} \)-covering and contains all words with weight \( \frac{n}{6} \) or \( \frac{5n}{6} \). Its size is:

\[
|\mathcal{C}| = 2^\left( \begin{pmatrix} n \\ n/6 \end{pmatrix} \right) \sim 1.5691^n \cdot n^{-1/2}.
\]

The large disparity between these two bounds motivates the research for different \( t \)s.

The final result of this chapter, however, will be a negative one: namely, we show that Conjecture 4.3 does not hold, and after reaching a threshold value we do not obtain a smaller code neither by enlarging the constant \( t \), nor by
modifying the covering radius, and the code’s size tends rapidly to \(2^n\) as the covering radius decreases. This result is expressed in the main theorem of this chapter:

**Theorem 4.4** For \(t, k \in \Theta(1)\), let \(\hat{V}_t \subseteq \mathbb{N}^t\) be the least costly set of centres of Manhattan balls of radius \(\frac{1}{t}\) covering the \(t\)-dimensional grid, and \(K(\hat{V}_t)\) denote the number of elements in the piecewise linear code corresponding to \(\hat{V}_t\).

Then, \(\forall t\) such that \(1 \ll t \in \Theta(1)\), \(K(\hat{V}_t) \approx 2^{(k-2)n/k}\).

The rest of the chapter is dedicated to showing how to cover the grid in the most cost-effective way, and is organised as follows: we start by formally defining the problem we want to solve; then, we show how to solve the problem for covering radius equals to \(\frac{n}{t}\) in 2 and 3 dimensions; finally, we extend the investigation to arbitrary constant dimension and linear covering radius.

### 4.2 Covering the Grid

#### 4.2.1 Problem Definition

Before looking into what happens for \(t > 1\), though, we want to formally define the problem we are trying to solve, which differs substantially from the traditional covering code construction problem:

**Original problem:** Optimal binary covering code construction.

*Input:* \(n \in \mathbb{N}, r \in [n]\).

*Output:* \(C \subseteq \{0, 1\}^n\) such that \(\forall c \in \{0, 1\}^n, d_H(c, C) \leq r\), and \(C\) has minimum possible size.

**New problem:** Optimal grid covering construction.

*Input:* \(n \in \mathbb{N}, t \in \Theta(1), r \in [n]\).

*Output:* \(V \subseteq \mathbb{N}^t\) such that \(\forall v \in \mathbb{N}^t, d_M(v, V) \leq r\), and \(V\) has minimum possible cost.

By *cost*, denoted by \(K(V)\), we mean the sum of the grid entries corresponding to the elements of \(V\). For the covering code \(C\) corresponding to the vectors of weights in \(V\), \(K(V) = |C|\).

If \(r = \frac{n}{t}\) for a constant \(c \in \mathbb{N}\), the first problem (unless \(c \leq 2\)) needs exponential time, since, by the sphere covering bound, the size of the optimal code is exponential in the input size. The second problem, on the contrary, with a covering radius linear in \(n\) appears to be solvable in polynomial time, because the elements of the grid have a cost forcing us to pick certain centres, and the number of elements in the grid is polynomial. Furthermore, if we only care about the asymptotical size of the resulting code (i.e., up to polynomial factors), the elements with largest cost are going to outweigh the
4. Piecewise Linear Codes

rest, which we are going to use to “fill the holes”. The most costly element is the centre \( c = \left( \frac{n}{2t}, \ldots, \frac{n}{2t} \right) \) of the grid: to find the asymptotically optimal grid covering, we start by covering \( c \), corresponding to \( \prod_{i=1}^{t} \left( \frac{n}{2t} \right) \) codewords; as we move towards the border of the grid, we get smaller entries, reaching eventually 1 (for all-0 and all-1 words, placed in the corners of the grid). To cover the centre \( c \) and its \( \frac{n}{t} \)-neighbourhood in the most economical way, we pick elements as far away as possible from it: the ones on the surface of the Manhattan ball of radius \( \frac{n}{t} \) centred in \( c \).

Now that we have a clear definition of the problem, we can try to compute the (asymptotically) optimal piecewise linear code for \( t = 2 \).

4.2.2 2 Dimensions

For \( n = \frac{n}{2} + \frac{n}{2} \), the \( \frac{n}{t} \)-ball centred in the centre of the grid covers more than half of the grid. It looks like a square, tilted by 45°.

![Figure 4.4: The ball of elements with Manhattan distance at most \( \frac{n}{t} \) from \( \left( \frac{n}{4}, \frac{n}{4} \right) \) in the 2-dimensional grid.](image)

In order to cover the central element and its surroundings, we have to choose an entry on each of the four portions of the surface of the ball contained in the grid (the four diagonal segments in the figure).

An entry in position \((a, b)\), with \( a, b \in \left[ \frac{n}{2} \right] \), contains the value \( \binom{n/2}{a} \cdot \binom{n/2}{b} \); thanks to the fact that \( \binom{n}{i} = \binom{n}{n-i} \), we can start the search in any of the four portions of the ball, since the elements on each segment have cost-wise symmetric counterparts on the other three segments. We choose, w.l.o.g., to start from the lower-right segment, connecting the points \( \left( \frac{n}{2}, \frac{n}{2} \right) \) and \( \left( \frac{n}{2}, \frac{n}{2} \right) \).

We are searching, in the lower-right fourth of the grid, for a vector \((a, b)\) such
that: \( a + b = \frac{n}{4} + \frac{n}{4} + \frac{n}{3} = \frac{5n}{6} \), i.e., the Manhattan distance from the centre of the grid is maximised. The number \( 5/6 \) recurs in higher dimensions, too: the sum of the vector entries on the portion of the ball close to \((\frac{n}{4}, \ldots, \frac{n}{4})\) has to be equal to \( \sum_{i=1}^{t} \frac{n}{2t} + \frac{n}{3} = \frac{n}{2} + \frac{n}{3} = \frac{5n}{6}, \forall t \in \mathbb{N} \).

Let’s now look closely at the portion of the ball’s surface near the lower-right corner of the grid.

We can choose between the elements, going from top-right to bottom-left:

\[
\begin{align*}
\binom{n/3}{n/3}, & \quad \binom{n/2}{n/3-1} \cdot \binom{n/2}{1}, & \quad \binom{n/2}{n/3-2} \cdot \binom{n/2}{1}, & \quad \ldots, & \quad \binom{n/2}{1} \cdot \binom{n/2}{n/3-1}, & \quad \binom{n/2}{n/3}.
\end{align*}
\]

The following important observation allows us to find the smallest.

**Observation 4.5** For every \( i < \frac{k}{2} \), \( \binom{n}{k-i} \cdot \binom{n}{i} < \binom{n}{k-i-1} \cdot \binom{n}{i+1} \).

**Proof**

\[
\begin{align*}
\binom{n}{k-i} \cdot \binom{n}{i} & < \binom{n}{k-i-1} \cdot \binom{n}{i+1} \\
\Leftrightarrow \frac{n!}{(n-k+i)!(k-i)!} \cdot \frac{n!}{i!(n-i)!} & < \frac{n!}{(n-k+i+1)!(k-i-1)!} \cdot \frac{n!}{(i+1)!(n-i-1)!} \\
\Leftrightarrow \frac{1}{(k-i)(n-i)} & < \frac{1}{(n-k+i+1)(i+1)} \\
\Leftrightarrow (n-k+i+1)(i+1) & < (k-i)(n-i).
\end{align*}
\]
Assume w.l.o.g. that $k$ is even. Since $i \leq \frac{k}{2} - 1$,

\[
(n - k + i + 1)(i + 1) \leq \left(n - \frac{k}{2}\right) \cdot \frac{k}{2}, \quad \text{and} \\
(n - i)(k - i) \geq \left(n - \frac{k}{2} + 1\right)\left(\frac{k}{2} + 1\right).
\]

\[\square\]

For $i > k/2$ we get the contrary: monotonically decreasing values.

The smallest of the elements on the ball’s boundary is therefore the one in the top-right (or bottom-left) corner, containing the quantity $\binom{n/2}{n/3}$, since the value of these products has a symmetric behaviour: increasing from the extremes $\binom{n/2}{n/3}$ to the centre, until achieving the maximum in $\binom{n/2}{n/6}$.

**Figure 4.6:** An asymptotically optimal grid cover for $n = \frac{n}{2} + \frac{n}{3}$ and radius $\frac{n}{2}$. The cover contains the pairs $(0, \frac{n}{3})$, $(\frac{n}{6}, 0)$, $(\frac{n}{2}, \frac{n}{2})$, $(\frac{n}{3}, \frac{n}{3})$. We only show the balls around $(0, \frac{n}{3})$ and $(\frac{n}{2}, \frac{n}{6})$ for clarity.

Before computing the cost of the cover we just found, let us derive an approximation for the binomial coefficient $\binom{n/2}{n/3}$ that we will use several times throughout our investigations.
4.2. Covering the Grid

**Approximation 4.6** For $j, k \in \Theta(1)$ such that $k < j$,

$$\left(\frac{n/k}{n/j}\right) \approx \frac{1}{\sqrt{n}} \cdot \left(\frac{j^{1/k}}{k^{1/(j-k)(j-k)/kj}}\right)^n.$$

**Proof**

$$\left(\frac{n/k}{n/j}\right) = \frac{(n/k)!}{(n/j)! (n/k-n/j)!} \approx \frac{1}{\sqrt{n}} \cdot \frac{\left(\frac{n}{k}\right)^{n/k}}{\left(\frac{n}{j}\right)^{n/j} \left(\frac{n(j-k)}{kj}\right)^{n(j-k)/kj}} \quad (4.1)$$

$$= \frac{1}{\sqrt{n}} \cdot \frac{j^{-n/k}}{j^{-n/(j-k)k(j-k)/kj} k^{-n(j-k)/kj}} \quad (4.2)$$

$$= \frac{1}{\sqrt{n}} \cdot \left(\frac{j^{1/k}}{k^{1/(j-k)(j-k)/kj}}\right)^n,$$

where (4.1) uses Stirling’s approximation (ignoring constants): for $n \to +\infty$, $n! \sim \sqrt{2\pi n} \cdot n^n e^{-n}$. (4.2) relies on the fact that since $\frac{1}{k} = \frac{1}{j} + \frac{j-k}{kj}$, all the exponentials with $e$ or $n$ as a base get canceled out.

Then, if we let $\hat{\mathcal{V}} := \left\{\left(0, \frac{n}{6}, \frac{n}{2}\right), \left(\frac{n}{6}, \frac{n}{2}, 0\right), \left(\frac{n}{2}, 0, \frac{5n}{6}\right)\right\}$,

$$K(\hat{\mathcal{V}}) = 4 \cdot \left(\frac{n/2}{n/6}\right) = O\left(n^{-1/2} \left(\frac{\sqrt{6}}{\sqrt{2}\sqrt{4}}\right)^n\right) = O\left(\frac{1.37473^n}{\sqrt{n}}\right).$$

We improved on the $O^*(1.5691^n)$ result we got for $t = 1$, but we’re still far from the binary entropy bound of $O^*(1.0582^n)$.

4.2.3 3 Dimensions

We can improve again in three dimensions. But before doing so, we will make the setting more intuitive by approximating the $t$-dimensional grid as a $t$-cube in the Euclidean space. The sides of the cube range from 0 to $\frac{1}{t}$ and the central element is $(\frac{1}{2t}, \ldots, \frac{1}{2t})$.

For $t = 3$, the Manhattan balls have radius $\frac{1}{3}$ and their shape is the regular octahedron’s, with edge length $\frac{\sqrt{2}}{3}$.
4. Piecewise Linear Codes

![Image of a cube with labeled vertices: (1/6, 1/6, 1/6), (0, 1/6, 0), (0, 0, 1/3), (0, 1/3, 1/6), (0, 1/6, 1/3), (0, 1/6, 1/3), (0, 1/6, 1/3), (0, 1/6, 1/3), (0, 1/6, 1/3), (0, 1/6, 1/3), (0, 1/6, 1/3), (0, 1/6, 1/3).]

**Figure 4.7:** The 3-grid, represented as a cube of side $\frac{1}{3}$.

The intersection between the ball and each facet of the cube is smaller than in the 2-dimensional case.

![Image of a 2D intersection with labeled points: (0, 1/6, 0), (0, 0, 1/3), (0, 1/3, 1/6), (0, 1/6, 1/3).]

**Figure 4.8:** The intersection of the ball centred in $(\frac{1}{6}, \frac{1}{6}, \frac{1}{6})$ with the face $w_1 = 0$.

Based on our previous considerations, we would like to pick elements for the piecewise linear code on the surface of the cube. As the picture depicts, for $t = 3$, the ball of radius $\frac{1}{3}$ centred in $(\frac{1}{6}, \frac{1}{6}, \frac{1}{6})$ is big enough to touch not only the surface but also the edges of the cube. Namely, the ball it is tangent to all points in the set

$$P = \left\{ \left( \frac{1}{6}, k, j \right), \left( k, \frac{1}{6}, j \right), \left( k, j, \frac{1}{6} \right) \mid k, j \in \left\{ 0, \frac{1}{3} \right\} \right\},$$

the midpoints of the 12 edges of the cube. By Observation 4.5, these intersection points are the least costly we can pick in order to cover the centre of the cube. We now prove that six edge midpoints cover not only the centre, but the whole cube.
Theorem 4.7 The balls centred in the midpoints of 3 pairs of distinct antipodal edges cover of the cube.

Proof For \( s \in \{0, 1\} \), let
\[
V_s = \left\{ \left( \frac{1}{6}, s, s \right), \left( s, \frac{1}{6}, s \right), \left( s, s, \frac{1}{6} \right) \right\}.
\]
We prove the theorem for the set of centres \( \tilde{V} = \{V_0, V_1\} \). By symmetry of the cube, the claim follows for any three pairs of nodes satisfying the required properties (being on the midpoint of an edge, and being pairwisely antipodal).

We will show that for all \( p = (x, y, z) \in [0, \frac{1}{3}]^3 \) with \( x + y + z \leq \frac{1}{2} \),
\[
d_M(p, V_0) \leq \frac{1}{3};
\]
by symmetry, the points \( p' = (x', y', z') \) with \( x' + y' + z' \geq \frac{1}{2} \) satisfy
\[
d_M(p, V_1) \leq \frac{1}{3}.
\]
Let us thus consider \( p = (x, y, z) \in [0, \frac{1}{3}]^3 \) with \( x + y + z \leq \frac{1}{2} \), and suppose w.l.o.g. that \( x \) is the largest coordinate.

- If \( x > \frac{1}{6} \), then \( d_M(p, (\frac{1}{6}, 0, 0)) = (x - \frac{1}{6}) + y + z \leq \frac{1}{3} \).
- If \( x \leq \frac{1}{6} \), then \( y, z \leq \frac{1}{6} \). Hence,
\[
d_M(p, (\frac{1}{6}, 0, 0)) = \frac{1}{6} - x + y + z \geq \frac{1}{6} + z \leq \frac{1}{3} \quad \square
\]

By our previous observations, \( \tilde{V} \) has asymptotically optimal cost among all covers. Approximation 4.6 yields:
\[
K(\tilde{V}) = 6 \left( \frac{n^3}{n/6} \right) = O(2^{n/3}n^{-1/2}) = O(1.2599^n n^{-1/2}),
\]
which improves again from the 2-dimensional case, where we got a cost of \( O(1.37473^n) \). Unfortunately, this is the last improvement we will get with our approach. Furthermore, the \( 2^{n/3} \) bound is quite useless, since it could have been derived with the following naive concatenation:

\( \mathcal{C} \), a covering code with covering radius \( \frac{4}{3} \) and length \( n \), can be built by concatenating \( 0^{2n/3} \) and \( 1^{2n/3} \) to all possible binary vectors \( s \) of length \( \frac{4}{3} \):

\[
\forall s \in \{0, 1\}^{n/3}, \{(1^{2n/3}s), (0^{2n/3}s)\} \in \mathcal{C}.
\]

We get \( 2 \cdot 2^{n/3} \) words in the code, which has covering radius \( \frac{4}{3} \) in the first \( \frac{2n}{3} \)-sized block of its codewords, and covering radius 0 in the last \( \frac{4}{3} \)-block (consisting of all the possible words of length \( \frac{4}{3} \) in \( \mathbb{H}_n \)). This gives us a construction matching the \( O(2^{n/3}n^{-1/2}) \) bound, up to the polynomial factor.
4.2.4 Larger Dimensions

For the fourth dimension and larger, instead of considering the grid (to which we will continue to refer to as “cube”) and the cross-polytope balls geometrically, we look at the problem from an optimisation point of view.

Following once again the strategy of covering the most expensive element first, we are interested in the most economic points on the surface of the ball centred in \((\frac{1}{2t}, \ldots, \frac{1}{2t})\) contained in the cube. A point’s cost can be deduced from its coordinates, as we saw previously, via the cost function:

\[
K\left(\left(\frac{n}{x_1}, \ldots, \frac{n}{x_t}\right)\right) = \prod_{i=1}^{t} \left(\frac{n}{n/x_i}\right)^{\frac{1}{n^{1/2}}} \prod_{i=1}^{t} \left(\frac{x_i^{1/t}}{t^{1/t}(x_i/t)(x_i/t)}\right)^{n}
\]

with the convention that if the coordinate of a point is 0, we can express it as \(\frac{n}{\infty}\), since

\[
\lim_{x \to +\infty} \left(\frac{x^{1/t}}{t^{1/t}(x-t)(x-t)/tx}\right) = 1.
\]

Up to polynomial factors, which we ignore, the cost function of a polynomial sized set of points \(V\) corresponds to the maximum cost of a point in \(V\). We now wish to find a point \(p\) on the surface of minimal cost and then see whether we can cover the grid with \(p\) and points in symmetric positions, like we did before.

As a start, we consider the portion of the grid included between the point \((\frac{1}{2t}, \ldots, \frac{1}{2t})\) and \((\frac{1}{t}, \ldots, \frac{1}{t})\). That is, points where all coordinates are larger than \(\frac{1}{2t}\). This way we are considering only a \(2^{-t}\) fraction of the cube, but by symmetry all such fractions have equal size and intersect equally with a Manhattan ball placed in the centre.

The analysis of the least costly entries is not as immediate as in the previous cases. The optimisation problem has the following form:

\[
\text{minimize } K\left(\left(\frac{n}{x_1}, \ldots, \frac{n}{x_t}\right)\right)
\]

such that:

\[
\forall i \in \{1, \ldots, t\} \text{, } t \leq x_i \leq 2t, \text{ and }
\]

\[
\sum_{i=1}^{t} \frac{1}{x_i} = \sum_{i=1}^{t} \frac{1}{2t} + \frac{1}{3} = \frac{5}{6}.
\]

The first constraint restricts the domain to the \(2^{-t}\)-fraction of the cube we want to look at, and the second constraint makes sure we only consider the surface of the ball with radius \(\frac{n}{3}\) centred in \((\frac{n}{2t}, \ldots, \frac{n}{2t})\).
The problem can be simplified by substituting the constraint in the objective function, and considering for \( n \) a constant \( g \gg \max\{x_1, \ldots, x_t\} \): the value of \( n \) does not influence the coordinates where the function achieves its minimum. We will hence consider Manhattan balls of radius \( \frac{1}{3} \) in the \( t \)-dimensional cube \([\frac{1}{2}]^t\). We obtain:

\[
\text{minimize } K\left(\left(\frac{1}{x_1}, \ldots, \frac{1}{x_{t-1}}, \frac{5}{6} - \frac{1}{t} \sum_{i=1}^{t-1} \frac{1}{x_i}\right)\right)
\]

such that:

\[
\forall i \in [t], t \leq x_i \leq 2t, \text{ and } \frac{5}{6} - \frac{1}{t} \leq \frac{1}{2} \sum_{i=1}^{t-1} \frac{1}{x_i} \leq \frac{5}{6} - \frac{1}{2t}.
\]

We notice that the function \( K \) is convex: each entry of the product is the convex, positive function \( \left(\frac{g}{x}\right)^{\frac{1}{t}} \), for \( x \geq t \). Hence the product is overall positive and convex (monotonically decreasing in the \( 2^{-t} \) portion of the ball we are looking at).

In order to avoid continuity problems with \( K \), when feeding it to a convex optimisation program, we can extend the binomial function to the real domain using, instead of the factorial function, Euler’s Gamma function

\[
\Gamma(x) = \int_0^\infty y^{x-1}e^{-y}dy,
\]

which is both convex and logarithmically convex on the \( \mathbb{R}^+ \) domain.

Running an optimisation program in the 4-dimensional grid gives us the following result: \( K(V) \) is minimised for \( \tilde{V} = (\frac{1}{4}, \frac{1}{4}, \frac{1}{8}, \frac{5}{24}) \). We can see that the intuition of maximising the number of entries equals to \( \frac{1}{t} \), captured in Observation 4.5, finds a confirmation in \( \tilde{V} \): since three entries can not be equals to \( \frac{1}{4} \) by our previous considerations (the ball does not intersect the cube’s edges in 4 dimensions), only two are. The penultimate entry is equals to \( \frac{1}{2} \), and the last simply exploits the rest of the available Manhattan distance left.

The result, though, is quite disappointing, since we get

\[
K(\tilde{V}) = O(1.3309^n n^{-1/2}),
\]

more than the cost of the optimum in 3 dimensions!

The likely explanation for this behaviour is the much faster decrease of the volume of the balls than the one of the cube: for a cross-polytope (radius 1)
and a cube with edge length 1, the volume of the first tends to zero as we enlarge the dimensions, whereas the volume of the cube keeps constant:

\[
\text{Vol}_1(\ell_1\text{-ball with radius 1}) = \frac{2^t}{t!} \\
\text{Vol}_1(\text{cube with side 1}) = 1.
\]

This is not a problem in and of itself, for we do not care about the number of balls we pick, but only about the total cost; the problem is that small balls need to get closer to the cube’s centre in order to cover it, and the trade-off we gain by decreasing their coordinates does not pay off.

For instance, a consequence of this shrinkage is that the ball centred in \( c = (\frac{1}{t}, \ldots, \frac{1}{t}) \) touches the edges of the cube only up to the 3-dimensional case, because the distance between \( c \) and any edge is

\[
d_M\left(\left(\frac{1}{t}, \ldots, \frac{1}{t}, \frac{1}{2t}\right), c\right) = \sum_{j=1}^{t-1} \frac{1}{2t} = \frac{t-1}{2t}.
\]

In four dimensions, we have that

\[
d_M\left(\left(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{8}\right), c\right) = \frac{3}{8} > \frac{1}{3},
\]

and the Manhattan distance to the edges tends to \( \frac{1}{t} \) as the dimension grows. Therefore, as we enlarge (but keep constant) the dimension \( t \), the most economical element on the ball has to contain more and more coordinates different from \( \frac{1}{t} \).

**Observation 4.8** In dimension \( t \), the number of coordinates of a vector on the surface of a Manhattan ball of radius \( r \) equals to \( \frac{1}{t} \) is at most \( 2t \cdot r \).

**Proof** Let \( k \) be the maximum number of coordinates equal to \( \frac{1}{t} \). Then:

\[
d_M\left(\left(\frac{1}{t}, \ldots, \frac{1}{t}, \ldots\right)_{k \text{ times}} \left(\frac{1}{2t}, \ldots, \frac{1}{2t}\right)\right) \geq \frac{k}{2t}.
\]

Therefore, since the total distance from the ball’s surface to the centre can be at most \( r \),

\[
k \leq 2t \cdot r.
\]

For \( r = \frac{1}{3} \), we have that

\[
k \leq \frac{2t}{3}.
\]
4.3 Disproving the Conjecture

As we enlarge \( t \), more coordinates need to be smaller than \( \frac{1}{t} \), thus more factors in the cost’s product are not equals to 1. On the other hand, the binomial functions in the product get smaller arguments. We will soon prove that neither of the two factors prevails, and as we enlarge \( t \), we keep getting the same value for the cost of the cheapest cover \( K(\tilde{V}_t) \).

**Lemma 4.9** For \( t \in \Theta(1) \), consider the \( \left[ \frac{1}{t} \right]^t \) grid.

Let \( S \) be the surface of a Manhattan \( t \)-ball of radius \( \frac{1}{3} \) centred in \( \{ \frac{1}{t}, \ldots, \frac{1}{t} \} \), and \( D \) be the \( 2^{-t} \)-portion of the grid where each coordinate is at least as large as \( \frac{1}{2t} \).

Then, for each three sets of indices \( I_0, I_1, I_2 \subset [t] \) such that \( |I_0 \cap I_1| = 0, |I_0| = \left\lceil \frac{2t}{3} \right\rceil, |I_1| = t - \left\lceil \frac{2t}{3} \right\rceil - 1, I_2 = [t] \setminus (I_0 \cup I_1) \),

the cheapest (i.e., smallest) entries in \( S \cap D \) are defined by the set

\[
\gamma^* = \left\{ (x_1, \ldots, x_t) \mid \forall i \in I_0, x_i = \frac{1}{t}; \forall j \in I_1, x_j = \frac{1}{2t}; \forall k \in I_2, x_k = \frac{1}{2t} + \frac{1}{3} - \left\lfloor \frac{2t}{3} \right\rfloor \right\}.
\]

**Proof** We will build the cheapest vectors step by step.

Recall that Observation 4.5 states that \( \forall i < \frac{1}{x}, \left( \begin{array}{c} n \\ k-i \end{array} \right) \cdot \left( \begin{array}{c} n \\ i \end{array} \right) < \left( \begin{array}{c} n \\ k-1 \end{array} \right) \cdot \left( \begin{array}{c} n \\ i \end{array} \right) \). The most economic vectors in \( S \cap D \) will therefore have the maximum possible number of coordinates equals to \( \frac{1}{t} \), since independently from the value \( \frac{1}{x} \) that the other coordinates have, by Observation 4.5 (with \( k = n/t + n/x, i = n/x \)), it holds that

\[
\left( \begin{array}{c} n/t \\ n/t \end{array} \right) \cdot \left( \begin{array}{c} n/t \\ n/x \end{array} \right) = \left( \begin{array}{c} n/t \\ (n/t+n/x)-n/x \end{array} \right) \cdot \left( \begin{array}{c} n/t \\ n/x+1 \end{array} \right) < \left( \begin{array}{c} n/t \\ n/t+n/x-n/x-1 \end{array} \right) \cdot \left( \begin{array}{c} n/t \\ n/x+1 \end{array} \right).
\]

Hence, in order to minimise the cost of a vector, we set \( \left\lfloor \frac{2t}{3} \right\rfloor \) coordinates to \( \frac{1}{t} \).

If \( \left\lfloor \frac{2t}{3} \right\rfloor \in \mathbb{N} \), we set all the remaining coordinates to \( \frac{1}{2t} \) and we are done. Otherwise, we still have

\[
d_{\text{left}} := \frac{1}{3} - \left\lfloor \frac{2t}{3} \right\rfloor \cdot \frac{1}{2t} \in \left( 0, \frac{1}{2t} \right)
\]

distance to exploit. Again by Observation 4.5 (with \( k = \frac{n}{t} + d_{\text{left}}, i = \frac{n}{2t} \)), the most cost-effective way to use it is to sum it to one of the coordinates and
get \( \frac{1}{2t} + d_{\text{left}} \), while the remaining coordinates are set to \( \frac{1}{2t} \):

\[
\begin{pmatrix} n/t \\ n/2t + d_{\text{left}} \end{pmatrix} \cdot \begin{pmatrix} n/t \\ n/2t \end{pmatrix} = \begin{pmatrix} n/t \\ n/t + d_{\text{left}} - n/2t \end{pmatrix} \cdot \begin{pmatrix} n/2t \\ n/2t \end{pmatrix} < \begin{pmatrix} n/t \\ n/t + d_{\text{left}} - n/2t - 1 \end{pmatrix} \cdot \begin{pmatrix} n/2t + 1 \\ n/2t \end{pmatrix}.
\]

To sum it up, the vectors we identified as most economic in \( S \cap D \) are the ones that have:

- \( \lceil \frac{2t}{3} \rceil \) many coordinates equals to \( \frac{1}{t} \);
- \( t - \lceil \frac{2t}{3} \rceil - 1 \) many coordinates equals to \( \frac{1}{2t} \);
- one coordinate equals to \( \frac{1}{2t} + \frac{3}{t} - \lceil \frac{2t}{3} \rceil \frac{1}{2t} \);

corresponding to the definition of the vectors in the set \( \mathcal{V}^* \).

Notice that if \( \lceil \frac{2t}{3} \rceil = \frac{2t}{3} \), the last mentioned coordinate of \( \mathcal{V}^* \) is just equals to \( \frac{1}{2t} \).

We now show that by picking the vectors of \( \mathcal{V}^* \) for a piecewise linear code, we cover the portion of the cube under consideration.

**Lemma 4.10** The Manhattan balls of radius \( \frac{1}{3} \) centred in the vectors of \( \mathcal{V}^* \) cover \( D \), the \( 2^{-t} \)-portion of the grid where each coordinate is at least as large as \( \frac{1}{2t} \).

**Proof** We prove that any vector \( s \in D \) has distance at most \( \frac{1}{3} \) from a vector \( v \in \mathcal{V}^* \). Let us assume for simplicity that \( \lceil \frac{2t}{3} \rceil = \frac{2t}{3} \). For

\[
s = (\epsilon_1, \ldots, \epsilon_t) \quad \forall i \in [t], \epsilon_i \in \left[ \frac{1}{2t}, \frac{1}{t} \right],
\]

let \( E_1 \) be the set of the \( \frac{2t}{3} \) largest coordinates of \( s \), and \( E_2 \) be the set of the remaining (smallest) \( \frac{t}{3} \) coordinates. Recall that every vector of \( \mathcal{V}^* \) has \( \frac{2t}{3} \) many coordinates equals to \( \frac{t}{3} \) and \( \frac{t}{3} \) many coordinates equals to \( \frac{1}{2t} \), thanks to the fact that the set \( \mathcal{V}^* \) contains all permutations of the coordinates of its vectors, we can choose a \( v \in \mathcal{V}^* \) whose entries equals to \( \frac{1}{t} \) are in the same positions as the elements of the set \( E_1 \) in \( s \). Therefore

\[
d_M(s, v) = \sum_{\epsilon \in E_1} \left( \frac{1}{t} - \epsilon \right) + \sum_{\epsilon \in E_2} \left( \epsilon - \frac{1}{2t} \right) = \frac{1}{t} - \left( \sum_{\epsilon \in E_1} \epsilon - \sum_{\epsilon \in E_2} \epsilon \right).
\]

The Manhattan distance is maximised when the expression in the parentheses is minimised, i.e., when the elements of the sets \( E_1, E_2 \) all have the same value \( \epsilon' \):

\[
d_M(s, v) \leq \frac{1}{2} - \frac{t}{3} \epsilon'.
\]
4.3. Disproving the Conjecture

The distance is maximum for the lowest possible value of $\varepsilon'$: $\frac{1}{2t}$.

$$d_M(s, v) \leq \frac{1}{2} - \left( \frac{t}{3} \cdot \frac{1}{2t} \right) = \frac{1}{3}. \quad \square$$

We can define a set analogous to $\mathcal{V}^*$, having its same size, for each portion of the cube. Let’s call these sets, in dimension $t$, $\mathcal{V}_1^t, \ldots, \mathcal{V}_{2^t-1}^t$, and let $\mathcal{V}_2^t = \mathcal{V}^*$. By reasons of symmetry, from Lemma 4.9 and Lemma 4.10, it follows that

**Lemma 4.11** $\mathcal{V}(t) = \{ \mathcal{V}_1^t, \ldots, \mathcal{V}_{2^t-1}^t \}$ is a piecewise linear covering code of covering radius $\frac{1}{3}$ in the $\left[ \frac{t}{3} \right]^t$ grid.

The next step consists in analysing the cost of our piecewise linear code: we show that $\mathcal{V}(t)$ has constant size, hence choosing one vector contained in it (which we are forced to do, in order to cover the centre of the grid) or all of them comes approximately at the same price.

**Lemma 4.12** For $t \in \Theta(1)$, to cover the $\left[ \frac{n}{t} \right]^t$ grid we need at most $\Theta(1)$ balls of radius $\frac{n}{4}$.

**Proof** Again, assume for simplicity that $\left\lfloor \frac{2t}{3} \right\rfloor = \frac{2t}{3}$. We start by computing the size of $\mathcal{V}^* = \mathcal{V}_2^t$:

$$|\mathcal{V}^*| = \sum_{i=1}^{2^t-1} \left( \frac{t}{2t/3} \right) \approx \left( \frac{3}{\sqrt{4}} \right)^t.$$  

By Lemma 4.11, the number of vectors we need is at most

$$|\mathcal{V}(t)| = 2^t |\mathcal{V}^*| = \left( 3 \sqrt{2} \right)^t = \Theta(1). \quad \square$$

Notice that, in general, for a function $f$ and $t \in \Theta(f)$, to cover the $\left[ \frac{n}{f} \right]^t$ grid we need a number of balls at most exponential in $f$.

Finally, we can state the main theorem of the chapter, and falsify Conjecture 4.3.

**Theorem 4.13** For $t \in \Theta(1)$, let $\mathcal{V}_t \in \left[ \frac{n}{t} \right]^t$ be the set of centres in the $t$-dimensional grid with asymptotically optimal cost.

Then, $\forall t$ such that $1 \ll t \in \Theta(1)$, $K(\mathcal{V}_t) \approx 2^{n/3}$.  

45
4. Piecewise Linear Codes

Proof By Lemma 4.9, we know that $V(t)$ contains the most economic elements on the surface of the $\frac{1}{k}$-ball centred in $(\frac{1}{2^t}, \ldots, \frac{1}{2^t})$, and by Lemma 4.11 they cover the grid. Since any cheaper set of centres covering the grid needs to use a subset of the vectors in $V(t)$, and, by Lemma 4.12, $V(t)$ has constant size, we can set $\tilde{V}_t = V(t)$ for any $t$, and get an asymptotically optimally sized, covering set of centres in the grid. Assuming for simplicity $\lfloor \frac{2t}{3} \rfloor = \frac{2t}{3}$, the cost is equals to

$$K(V(t)) = 2^t K(V_2^{t \lfloor \frac{t}{3} \rfloor}) = 2^t \left( \frac{n/t}{n/2t} \right)^{t/3}.$$ 

By Approximation 4.6, for a large enough $t$, $\left( \frac{n/t}{n/2t} \right) \approx 2^{n/t}$. Therefore, $\forall t$ s.t. $1 \ll t \in \Theta(1)$,

$$K(V(t)) = 2^t \cdot 2^{\frac{n}{t}} \approx 2^{n/3}. \quad \Box$$

The situation changes for $t \in \omega(1)$: Lemma 4.12 (and consequently Theorem 4.13) does not hold anymore and we have to search for the optimal configuration with more care.

We can generalise Theorem 4.13 for piecewise linear codes of arbitrary linear radii, and obtain Theorem 4.4, presented in the beginning of this chapter. Let us state it again, and then present its proof.

**Theorem 4.4** For $t, k \in \Theta(1)$, let $\tilde{V}_t \in \left[ \frac{n}{k} \right]^{t}$ be the set of centres of Manhattan balls of radius $\frac{n}{k}$ covering the $t$-dimensional grid with asymptotically optimal cost.

Then, $\forall t$ such that $1 \ll t \in \Theta(1)$, $K(\tilde{V}_t) \approx 2^{(k-2)n/k}$.

**Proof** By Observation 4.8 and slight modifications of Lemmas 4.9, 4.11 and 4.12, we are able to maintain that the asymptotically optimal vectors on the surface of a $\frac{n}{k}$-ball centred in $(\frac{1}{2^t}, \ldots, \frac{1}{2^t})$ have $\left\lfloor \frac{2t}{k} \right\rfloor$ elements equals to $\frac{1}{k}$ and the rest (besides a possible last element exploiting the remaining Manhattan distance $\frac{1}{k} - \left\lfloor \frac{2t}{k} \right\rfloor \cdot \frac{1}{2^t}$) equals to $\frac{1}{2^t}$.

Assuming $\left\lfloor \frac{2t}{k} \right\rfloor = \frac{2t}{k}$, with similar computations as in Theorem 4.13’s proof we get

$$K(\tilde{V}_t) = 2^t \left( \frac{n/t}{n/2t} \right)^{t(k-2)/k} \approx 2^{(k-2)n/k}. \quad \Box$$

Consistently with what we know, Theorem 4.4 with $k = 2$ states that in order to cover the $t$-cube with balls of radius $\frac{n}{2}$, we need at most $\Theta(1)$ codewords (two).
4.3. Disproving the Conjecture

4.3.1 Conclusion

The approach we proposed fails, since the cost tends quickly to $2^n$ as the covering radius decreases. However, a different trade-off between the dimension $t$ and the complexity might work better, e.g. $t \in \Theta(\log(n))$. Furthermore, splitting the grid in non-equally sized portions (we get a cuboidal grid) might yield different results, although it seems highly unlikely to perform better than the cubic grid, due to the regular shape of the Manhattan balls.
Appendix A

Bounds on $K(n, r)$

In this section, we present some bounds on the minimal size of binary covering codes of length $n$ and radius $r$. Such size is often referred to, in the literature, as $K(n, r)$.

We thank Professor Gerzson Kéri from MTA SZTAKI research institute for gathering up-to-date bounds on his website and authorising their use in this thesis.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$r = 1$</th>
<th>$r = 2$</th>
<th>$r = 3$</th>
<th>$r = 4$</th>
<th>$r = 5$</th>
<th>$r = 6$</th>
<th>$r = 7$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>16</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>32</td>
<td>12</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>62</td>
<td>16</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>107-120</td>
<td>24-30</td>
<td>12</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>180-192</td>
<td>37-44</td>
<td>15-16</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>342-380</td>
<td>62-78</td>
<td>18-28</td>
<td>11-12</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>598-704</td>
<td>97-128</td>
<td>28-42</td>
<td>12-16</td>
<td>7</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>14</td>
<td>1172-1408</td>
<td>159-248</td>
<td>44-64</td>
<td>16-28</td>
<td>10-12</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>15</td>
<td>2048</td>
<td>310-384</td>
<td>70-112</td>
<td>22-32</td>
<td>12-16</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>4096</td>
<td>512-768</td>
<td>115-192</td>
<td>33-64</td>
<td>13-28</td>
<td>9-12</td>
<td>4</td>
</tr>
<tr>
<td>17</td>
<td>7419-8192</td>
<td>859-1536</td>
<td>187-320</td>
<td>52-112</td>
<td>19-32</td>
<td>11-16</td>
<td>7</td>
</tr>
<tr>
<td>18</td>
<td>14564-16384</td>
<td>1702-2944</td>
<td>316-512</td>
<td>83-192</td>
<td>27-64</td>
<td>12-28</td>
<td>9-12</td>
</tr>
</tbody>
</table>

Table A.1: Best known bounds on $K(n, r)$. For space reasons, we don’t include the references to the papers in which these bounds are achieved; for a larger table with all proper references, please consult [1, Chapter 6.6]


Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor’s thesis, Master’s thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

CONSTRUCTING COVERING CODES

Author of work (in block letters):
For papers written by groups the names of all authors are required.

<table>
<thead>
<tr>
<th>Name(s):</th>
<th>First name(s):</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILIPPINI</td>
<td>LUCA TEODORO</td>
</tr>
</tbody>
</table>

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the ‘Citation etiquette’ information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date

Zürich, 24/03/2016

Signature(s)

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.