MASTER'S THESIS

Department of Mathematics

University of Trento

Italy

On two algebraic decision problems and their instances

Alex Pellegrini

Supervisors :

Dr. Alessio Meneghetti, University of Trento, Italy

Associate Prof. Dr. Massimiliano Sala, University of Trento, Italy

March 2019

To my family relatives and friends

Contents

List of Acronyms

MLD Maximum Likelihood Decoding

SAT Boolean Satisfiability Problem

3-SAT SAT with clauses with at most 3 literals

MQ Boolean Multivariate Quadratic Problem

CNF Conjunctive Normal Form

XOR Logical Exclusive OR

XOR-SAT Conjunction of XOR clauses

3-XOR-SAT XOR-SAT with clauses with at most 3 literals

RHS Right Hand Side

LHS Left Hand Side

Table of Notations

$\mathbb{F}_2$	The field of characteristic 2 or the "bits" field.
$\mathbb{F}_2^n$	The n-dimensional vector space over $\mathbb{F}_2$
$\mathbb{K}$	A field
$\mathcal{M}_{m,n}(\mathbb{K})$	The set of $m \times n$ matrices with entries in $\mathbb{K}$
$\alpha, \beta, \varphi, \psi, \omega$	Problem reductions
$\mathcal{O}$	Big-O notation
v	Binary vector
$C[n, k, d]$	Binary code of length n, dimension k and distance d
c	A codeword
$\mathcal{V}_{\mathbb{E}}(I)$	Variety of the ideal I over $\mathbb{E}$
$\mathcal{G}$	Groebner Basis
H, H_1	Parity check matrices
A, B	SAT CNF formulae
f, f_j, g_h	Polynomials in $\mathbb{F}_2[a_1, \ldots, a_l]$
P, S	Polynomial systems

Introduction

In this thesis we survey three NP-complete problems and we work to find explicit reductions between such problems. This goes along with a computation complexity analysis of our transformations in order to make sure everything is performed efficiently (i.e. polynomial time) from the point of view of the computability. We are given the following Maximum Likelihood Decoding (MLD for short), Satisfiability (SAT and 3-SAT) and Multivariate Quadratic equations (MQ) problems. Here we give the formal definition of such problems and in Chapter 1 we will explain them more in detail.

Problem 0.1 (MAXIMUM-LIKELIHOOD DECODING)**.** *Given a binary $m \times n$ matrix H, a vector $s \in \mathbb{F}_2^m$ and an integer $w > 0$, is there a vector $v \in \mathbb{F}_2^n$ of weight at most w, such that $Hv^\top = s$?*

Problem 0.2 (3-SATISFIABILITY)**.** *Given a set U of variables and a collection $\mathcal{C}$ of OR clauses over U such that each clause has at most 3 literals, is there a truth assignment for U such that every clause in $\mathcal{C}$ is satisfied?*

Problem 0.3 (MQ-PROBLEM)**.** *Consider the field $\mathbb{F}_2$ and the polynomial ring $\mathbb{F}_2[x_1, \ldots, x_n]$ over $\mathbb{F}_2$ with n variables. Given $P = \{f_1(x_1, \ldots, x_n), \ldots, f_n(x_1, \ldots, x_n), x_1^2 - x_1, \ldots, x_n^2 - x_n\}$ where the degree of each polynomial is at most 2, check whether the algebraic variety associated to the ideal generated by the polynomials in P is empty or not.*

The main goal of this work is to provide explicit polynomial transformations between MLD and MQ problems. The first aim was to

- Give a reduction (φ) from an instance MLD to an instance of SAT,
- Give a reduction (ψ) from an instance of SAT to an instance of 3-SAT,
- Give a reduction (ω) from an instance of 3-SAT to MQ

see figure (1) for a graphical interpretation. We do not claim these reductions as original.
Our main result is finding two direct transformations (α and β) between MLD and MQ. We are going to introduce all the transformations we found.

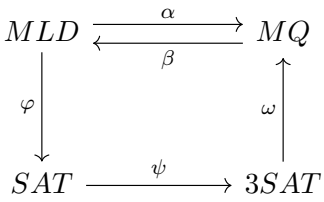

Figure 1: Diagram of transformations

By constructing the reduction β we define an infinite family of binary codes which we inspect in Chapter 5. The decoding of these codes would allow the solution of any MLD instance, namely, the decoding of any binary code. This is due to the fact that given a binary code, we can create an instance I_{MLD} of MLD by taking syndrome and a weight, we can apply reduction α to obtain an instance $\alpha(I_{MLD})$ of MQ. Now applying reduction β we get an instance $I'_{MLD} = \beta(\alpha(I_{MLD}))$, with a code belonging to our family.

The structure of the dissertation is the following. In the first chapter we give an introduction to the essential topics of computational complexity theory needed for this thesis. This chapter introduces the reader to the concepts of problem's tractability, to the complexity classes P and NP and gives some definition useful to get a better grasp of the Satisfiability problem.
In the second chapter we provide a complete and explicit description of a reduction from MLD problem to SAT, along with its computational complexity analysis. This also comprehends the description of a natural reduction that takes SAT into 3-SAT.
In the third chapter, we give an explicit description of the reduction from 3-SAT to MQ problem along with its complexity analysis.
The fourth chapter describes a direct reduction from MLD to MQ. We give a description, in Chapter 5, of a direct reduction from MQ to MLD. In this chapter we provide also a description and an analysis of the parameters of the family of codes we define as a part of reduction β. In the appendices we bring the MAGMA implementation of some of the reductions we introduced.

Chapter 1

Introduction to computational complexity theory

This chapter is intended to provide the reader with an introduction as well as some formal definition in order to understand the computational behaviour of a selected problem. A problem can be seen as a question to be solved and we define it as a family of instances P where:

Definition 1.1. *An **instance** $P(n)$ of a problem P is a set of concrete setups for the inputs, identified by a size parameter n, that are allowed to be passed to the problem.*

Definition 1.2. *The n parameter introduced in Definition 1.1 is also called **complexity parameter**.*

Thus for every value $n \in \mathbb{N}^+$ we have that $P(n)$ is an instance of the problem P. For example, we consider the problem P of **square matrix maximum rank**. Given $n \in \mathbb{N}^+$ we consider $P(n)$ as the instance corresponding to the matrices in $\mathcal{M}_{n,n}(\mathbb{K})$, with $\mathbb{K}$ a field. Therefore given a matrix $M \in \mathcal{M}_{n.n}(\mathbb{K})$ we ask whether M has maximum rank or not. This can be regarded as a decision problem which we define as:

Definition 1.3. *A **decision problem** is a problem which can be posed as a yes/no question on the set of possible inputs. For example "Does a satisfying assignment for a given CNF formula exist?" or "Is the given number a prime number?".*

Hence we can say that for each n there exists a function $f : (\mathbb{F}_2)^n \to \mathbb{F}_2$ that outputs the solution according to $P(n)$

Morever we define the cost of a problem as a function like:

$$cost : \{Problems\} \to \mathbb{N}^{+\mathbb{N}^+}$$

that is a function that returns a function from $\mathbb{N}^+$ to $\mathbb{N}^+$. Therefore given a problem P we obtain that $cost(P) = c_P : \mathbb{N}^+ \to \mathbb{N}^+$.

To better understand how this *cost* function is defined we need to itroduce the notion of algorithm, that is a set of rules or steps that need to be performed starting from the input in order to obtain an output, i.e. the solution of an instance of a problem. Consider again the matrix maximum rank problem described above. An process that allows to solve such problem, i.e. to decide whether an input matrix has maximum rank, is to compute the determinant of such matrix $det(M)$ and then check if it is 0 or not. In case $det(M) = 0$ we know that there are linearly dependent columns so the matrix is not full rank. These two steps form an algorithm. Actually the computation of $det(M)$ can be divided in more steps. The following pseudo code gives a more concrete view of the algorithm we are talking about.

```
function IsFullRank(M)                                  1
        d := ComputeDeterminant(M);                     2
        full := d != 0;                                 3
        return full;                                    4
end function;                                           5
```

Bear in mind that this is only one algorithm that can solve the instances of our problem. There could be eventually many other. So depending on the algorithm the problem we are tackling could seem very hard or very easy. Actually the hardness of a problem is completely independent form the choice of the algorithm used to solve it. By the way we need some tools now in order to determine the goodness of an algorithm. We can consider, as a time metric, the number of steps/operation an algorithm has to perform in order to produce the output. In the case of the maximum rank problem we need to estimate the number of aritmetic operation we need to compute the determinant and then to compare the result with 0. First of all we introduce a notation to describe an algorithm's running time depending on the size of the input.

Definition 1.4. *Let $f, g : \mathbb{N}^+ \to \mathbb{R}^+$, and assume $g(n)$ is the running time of an algorithm on an input of size n. We denote the asymptotic running time of an algorithm by $O(f(n))$. This is called Big-O notation, meaning that there exist some $c \in \mathbb{R}^+$ such that for all $n \in \mathbb{N}^+$, $g(n) \leq c \cdot f(n)$, i.e. $c \cdot f(n)$ bounds $g(n)$ from above.*

Let $A_{P(n)}$ be the set of all the possible algorithms solving the instance $P(n)$ and pick $a \in A_{P(n)}$ so let us set $c(a) = \mathcal{O}(f(n))$ for some $f : \mathbb{N}^+ \to \mathbb{R}^+$. Assume in our matrix maximum rank problem the `ComputeDeterminant(M)` applies the Gaussian elimination method to achieve its goal. It then performs a number of multiplications, divisions and subtractions to compute the result, (the cost of such operations may vary depending on the base field one is working on) but we assume that every operation has cost 1. The actual number of operation performed in Gaussian elimination method on

an $n \times n$ matrix is around n^3. The cost of comparing `d != 0` is again 1. So the total cost of our whole algorithm is upperbounded by $f(n) = n^3$ and therefore the cost is $\mathcal{O}(n^3)$. Now how can we determine the actual complexity of a problem? This is the same as asking "how is the *cost* function defined?". We can now define the complexity of P as follows

$$c_{P(n)} = \min_{a \in A_{P(n)}} \{c(a)\}$$

In other words the complexity of a problem is defined to be the complexity of the fastest algorithm solving all of its instances. Thus in the case of the rank problem we have that $c_{P(n)} \leq \mathcal{O}(n^3)$.

We pose our interest also on the 3-SAT problem which basically restricts the clauses of a CNF formula to be the disjunction of at most 3 literals. SAT was the first problem to be shown to be NP-complete, see the Cook-Levin theorem. In the next section we will introduce complexity classes as well as the notion of reductions between problems, see also [1].

1.1 The complexity classes P and NP and problem reductions

First of all we need to define what efficient means when it comes to talk about an algorithm.

Definition 1.5. *We say an algorithm is **polynomial time** or efficient if its running time is upper bounded by $O(n^k)$ for some constant k. A problem is **tractable** iff there exists a polynomial- time algorithm that solves all instances of it. A problem is **intractable** iff there is no known polynomial-time algorithm for solving all instances of it*

We are now ready to introduce the two complexity classes P and NP.

Definition 1.6. *We define **NP** (Nondeterministic polynomial time) as the set of all decision problems of which the correctness of any proposed solution to an instance of the problem can be checked efficiently.*

Definition 1.7. *We define **P** (Polynomial time) as the set of all decision problems that are in NP and are tractable. This imples $P \subset NP$*

We introduce also the notion of reduction in order to better explain the concept of NP-completeness.

Definition 1.8. *We define a reduction that takes a problem A into a problem B as a process that can efficently map an instance I_A of A into an instance I_B of B and solutions of I_A and I_B can efficently mapped into solution of the other.*

Definition 1.9. *We define the class* ***NP-complete*** *as the set of all problems that are in NP and any problem in NP reduces to them in polynomial time. A problem is* ***NP-hard*** *if an NP-complete problem reduces to it.*

In the next sections we are going to introduce the problems defined in the introduction chapter.

1.2 Boolean satisfiability problem, SAT

Let us now introduce the so called boolean satisfiability problem (SAT), which is the problem of determining if there exists a solution to a boolean formula.

Definition 1.10. *A* ***boolean literal*** x *is an object which can take only the two values true or false. We write the negation of* x *as* $\overline{x}$*.*

Definition 1.11. *The logical OR between two or more literals, e.g.* $x_1 \vee x_2$*, is called a* ***disjunction****.*

Definition 1.12. *A* ***clause*** *is the disjunction of a finite set of literals.*

Example 1.13. *Given the literals* $x_1, x_2, \ldots, x_n$*, an example of clause is* $(x_1 \vee x_2 \vee \ldots \vee x_n)$*. The clause is TRUE if at least one literal takes the value TRUE.*

Definition 1.14. *A* ***conjunctive normal form formula****, or CNF for short, is the conjunction of a finite set of clauses.*

According to Definition 1.14 a CNF formula expresses the requirement that all the clauses must be TRUE.

The corresponding decision problem is Problem (0.2)

Example 1.15. *Consider the following CNF formula*

$$(x_1 \vee \overline{x_2} \vee x_3) \wedge (x_2) \wedge (x_4 \vee \overline{x_1}).$$

This is a conjunction of 3 clauses wchich are made up by the litarls x_1, x_2, x_3, x_4 *and by their negations. A satisfying assignment of the literals would be* $x_1 = F, x_2 = T, x_3 = T, x_4 = F$*, indeed:*

$$(F \vee F \vee T) \wedge (T) \wedge (F \vee T) = T$$

1.3 Maximum likelihood decoding problem, MLD

This is one of the central algorithmic problem in coding theory. Berlekamp, McEliece, and van Tilborg have shown this problem to be NP-complete in [2]. We introduce some notation, see also [3], in order to make the reader more comfortable with the formal definition of the problem.

Definition 1.16. *The* ***distance*** $d(v_1, v_2)$ *between two vectors* $v_1, v_2 \in \mathbb{F}_2^n$ *is defined as the number of coordinates in which they differ.*

Definition 1.17. *Given* $v \in \mathbb{F}_2^n$*, the* ***weight*** $w(v)$ *of* v *is defined as the number of zero coordinates in* v*.*

Let $C \subseteq (\mathbb{F}_2)^n$ be a linear binary code, i.e. a linear subspace of $(\mathbb{F}_2)^n$ with length n and dimension k.

Definition 1.18. *The* ***distance*** d *of the linear code* C *is the minimum weight of its nonzero codewords, or equivalently, the minimum distance between distinct codewords.*

We denote $C[n, k, d]$ a binary linear code with length n, dimension k and distance d.

Definition 1.19. *The* ***generator matrix*** G *of* C *is a matrix whose rows form a basis for the linear code* C*.*

Definition 1.20. *The annihilator space* $C^\perp$ *of* C *with respect to the bilinear form* $< \cdot, \cdot >$ *is also called the* ***dual code*** *of* C*.*

Definition 1.21. *The generator matrix* H *of* $C^\perp$ *is also called the* ***parity-check matrix*** *of* C*.*

Note 1.22. *A codeword* c *is iin* C *if and only if the matrix-vector product* $Hc^\top = 0$*.*

Assume a "codeword" $c \in C$ is transmitted over a noisy channel and the vector r is received. The difference between the two vectors is called error vector e, i.e. $e = r + c$. If H is the parity check matrix of the code C and keeping in mind it represents a linear transformation, then:

$$Hr^\top = H(c+e)^\top = Hc^\top + He^\top = 0 + He^\top = He^\top \tag{1.1}$$

Definition 1.23. $Hr^\top$ *defined as in Equation* (1.1) *is called* ***syndrome*** *of* r*.*

Note 1.24. *The syndrome of* r *depends only on the error vector* e*.*

Decoding methods based on this facts are known as syndrome decoding. The classical way precomputes a table mapping $He^\top$ to e. Such table is called standard array and once the receiver detects e it is enough to compute $r - e$ to get the correct codeword.

Let us make an example to make this procedure more clear.

Example 1.25. *Let* C *be the* $[4, 2]$ *linear code given by:*

$$G = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix} \Longrightarrow H = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix}$$

The vectors v *and* y *are said to be equivalent with respect to* C *if and only if* $v - y \in C$*. Let's prove that this is an equivalence relation:*

1. $v - v = 0 \in C$,

2. *if* $v - y \in C$ *then* $y - v = -(v - y) \in C$ *since it is a scalar multiple of an element of* C,

3. *if* $v - y \in C$ *and* $y - z \in C$ *then* $v - z = (v - y) + (y - z) \in C$

and the equivalence classes are called the cosets of C. We precompute the syndromes table, what we above called the standard array, which looks like:

Synd.		Error			
0	0	0	0	0	0
1	1	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0

Table 1.1: Standard array

Let us explain how the table (1.1) has been constructed. The right side of first row consists in the 0 vector, which is the no-error vector, while the left side contains its syndrome. Actually each of the standard array consists of a full coset of C but here it is enough only to consider the vector with lowest weight (which corresponds to the error vector and is called coset leader) of each coset and its syndrome. In the following theorem we give a proof that this gives the correct error vector.

Theorem 1.26. *Let C be an [n,k]-code in* $\mathbb{F}_q^n$ *with codewords* c_i *where* $0 \leq i \leq q^k - 1$. *Let* l_j *where* $0 \leq j \leq q^{n-k} - 1$ *be a set of coset leaders for the cosets of this code. If* $r = l_j + c_i$ *then*

$$d(r, c_i) \leq d(r, c_h) \quad \textit{for all} \quad 0 \leq h \leq q^k - 1$$

Proof. We can write the distances in terms of the weights, that means:

$$d(r, c_i) = d(l_j + c_i, c_i) = w(l_j)$$

and also

$$d(r, c_h) = w(l_j + c_i - c_h)$$

Now by linearity of C we have that $c_i - c_h \in C$ so that l_j and $l_j + c_i - c_h$ are in the same coset but the weight of l_j is minimal. □

If the coset leader is unique then c_j *is the closest codeword. If it is not unique instead we have that* c_j *is as close as any other codeword. The following theorem gives a result about uniqueness.*

Theorem 1.27. *Consider a code C with minimum distance d. Let v be any vector such that*

$$w(v) \leq \left[\frac{d-1}{2}\right]$$

Then v is a unique element of minimum weight in its coset and therefore it is always the coset leader.

Proof. Assume there exists another vector y with the same weight as v in the same coset, therefore $v - y \in C$, but

$$w(v-y) \leq w(v) + w(y) \leq 2 \cdot \left[\frac{d-1}{2}\right] \leq d-1$$

which contraddicts the distance of the code, unless $v = y$. □

Now assume the codeword $c = (1,1,1,0)$ is sent and $r = (0,1,1,0)$ is received. We compute the syndrome of r i.e.:

$$Hr^\top = (1,1) = He^\top$$

Looking-up in S we obtain the error vector $e = (1,0,0,0)$, therefore we can recover the codeword c $r - e = (1,1,1,0) = v$.

The corresponding decision problem is Problem (0.1)

1.4 Multivariate quadratic equation system problem, MQ

A MQ-system of equations over $\mathbb{F}_2$ is a set of m polynomial equations of degree at most 2 in $\mathbb{F}_2[x_1, \ldots, x_n]$, i.e. $P : \mathbb{F}_2^n \to \mathbb{F}_2^m$ of the form:

$$P = \begin{cases} p^1(x_1, \ldots, x_n) = 0 \\ p^2(x_1, \ldots, x_n) = 0 \\ \qquad\qquad \vdots \\ p^m(x_1, \ldots, x_n) = 0 \end{cases} \tag{1.2}$$

where for every $h \in \{1, \ldots, m\}$

$$p^{(h)}(x_1, \ldots, x_n) = \sum_{1 \leq i \leq j \leq n} \gamma_{ij}^{(h)} x_i x_j + \sum_{1 \leq i \leq n} \beta_i^{(h)} x_i + \alpha^{(h)} \tag{1.3}$$

where $\gamma_{ij}^{(h)}, \beta_i^{(h)}, \alpha^{(h)} \in \mathbb{F}_2$.

The corresponding problem is Problem (0.3) and is show to be NP-complete [4]. Many cryptosystem rely their security on such problem see for example [5],[6],[7],[8] and [9] and nowadays the best known algorithm to compute a solution of these system is to exploit the results due to Bruno Buchberger. Let us introduce some definitions that will be useful during this dissertation.

Definition 1.28. *Let A be acommutative ring. A subset $I \subseteq A$ is an* ***ideal*** *of A if*

- $0 \in I$;
- $f, g \in I$, *then* $f + g \in I$;
- $f \in I$ *and* $g \in A$ *then* $fg \in I$.

Let $I \subseteq A$ be an ideal, if there is $S \subseteq I$, $\#S < +\infty$ such that

$$I = \left\{ \sum_{i=1}^{\#S} \lambda_i p_i | \lambda_i \in A, p_i \in S \right\}$$

then I is ***generated*** *by set, we write $I = \langle S \rangle$ and we also say that I is* ***finitely generated*** *since S is a finite set.*

Denote by $\mathcal{R} = \mathbb{K}[x_1, \ldots, x_m]$ the polynomial ring in m variables over the field $\mathbb{K}$. Denote by $\mathcal{N}$ the set of monomials in the variables $x_1, \ldots, x_m$, that is $\mathcal{N} = \{x_1^{a_1} \cdots x_m^{a_m} | (a_1, \ldots, a_m) \in \mathbb{N}^m\}$.

Definition 1.29. *Let I be an ideal of $\mathcal{R}$ and $\mathbb{E}$ be an extension field of $\mathbb{K}$, we denote by $\mathcal{V}_{\mathbb{E}}(I)$ the set of all the zeroes of I in $\mathbb{E}^m$:*

$$\mathcal{V}_{\mathbb{E}}(I) = \{A \in \mathbb{E}^m | f(A) = 0 \; \forall f \in I\}.$$

$\mathcal{V}_{\mathbb{E}}(I)$ is also called the ***variety*** *of I over $\mathbb{E}$.*

Definition 1.30. *A* ***monomial ordering*** *$\prec$ is a binary relation on $\mathcal{N}$ such that:*

1. *$\forall m_1 \neq m_2 \in \mathcal{N}$, either $m_1 \prec m_2$ or $m_2 \prec m_1$.*
2. *$\forall m_1, m_2, m_3 \in \mathcal{N}$, if $m_1 \prec m_2$ and $m_2 \prec m_3$ then $m_1 \prec m_3$.*
3. *$\forall m_1, m_2, m_3 \in \mathcal{N}$, if $m_1 \prec m_2$ then $m_1 \cdot m_3 \prec m_2 \cdot m_3$.*
4. *$1 \prec m \quad \forall m \in \mathcal{N}$, $m \neq 1$.*

Given $f \in \mathcal{R}$ we denote the leading monomial of f by lm(f).

Definition 1.31. *Let I be an ideal and $\prec$ be a monomial ordering. We say that $\mathcal{G} = \{g_1, \ldots, g_h\}$ is a Groebner Basis for I, if $\mathcal{G} \subseteq I$ and if for all $f \in I$, there exists $g_i \in \mathcal{G}$ such that* $\mathrm{lm}(f)$ *divides* $\mathrm{lm}(g)$

Note 1.32. *Note that if $\mathcal{G}$ is a Groebner Basis for I then $I = \langle \mathcal{G} \rangle$.*

During this work we will compute the Groebner Basis of a system of polynomials, generate an ideal using such basis and then compute the associate variety.

Chapter 2

MLD to SAT transformation

SAT was the first problem that has been shown to be NP-complete by the Cook-Levin theorem in [10] and [11]. We also know that MLD is NP-complete as well, and this means that an instance of MLD can be reduced into an instance of SAT in polynomial time. What we do is to find a reduction from MLD to SAT, i.e. we want to give the explicit transformation φ of diagram 1. The first step is therefore to encode Problem 0.1 into Problem 0.2. What we want to find is a propositional formula that reflects exactly Problem 0.1, to do this, we split the problem in more parts and find a SAT representation for each of these parts, and then put all the formulae together. Now we introduce an instance of the MLD problem with a fancy code and we are going to apply each step of the transformation we made during the description. We divided the transformation in the folloing steps:

- Encode the parity check matrix constraint $Hv^\top = s$.
- Encode the vector weight computation, i.e. compute $w(v)$
- Encode the weight check, i.e. checking that $w(x) \leq t$

Consider the instance of the MLD problem with the following settings:

$$n = 4, \mathcal{H} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix}, s = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, t = 1$$

The linear code we take as an example defined by $\mathcal{H}$ is a $[4, 2, 1]$ code. In the definition of MLD we have that the number of rows of the parity check matrix is independent from the number of columns and viceversa, this doesn't happen when it comes to linear codes since the number of rows is bounded by the number of columns. We are going through the transformation and show how to create the corresponding MQ instance for this simple example.

2.1 Parity check constraint

We have that the matrix multiplication $Hv^\top = s$ gives clauses of the following forms:

$$\begin{array}{l} \sum h_{i,j}v_j = 0 \text{ if } s_i = 0 \\ \sum h_{i,j}v_j = 1 \text{ if } s_i = 1 \end{array} \tag{2.1}$$

The above sum is over $\mathbb{F}_2$ which corresponds to the logical XOR. In the case $s_i = 0$ we can negate the last v_j and we can take the conditions to the form:

$$\sum l_j = 1 \tag{2.2}$$

in which l_j denotes a literal v_j or $\overline{v_j}$. So we from the matrix vector product above we got m XOR-SAT formulae which we need to reduce into 3-SAT formulae. In order to do this we take advantage of the following two propositions which allows us to achieve our goal. The proofs of the following two proposition can be found in [12].

Proposition 2.1. *Consider the clause $(l_1 \oplus l_2 \oplus \ldots \oplus l_n)$, then it is stisfiable iff there exist an auxiliary variable u and a truth assignment such that the two clauses $(l_1 \oplus l_2 \oplus \ldots \oplus l_{n-2} \oplus u)$ and $(\overline{u} \oplus l_{n-1} \oplus l_n)$ are satisfied.*

Proposition 2.2. *(1) A truth assignment satisfies $(l_1 \oplus l_2 \oplus l_3)$ iff each of the following clauses are satisfied:*

$$(l_1 \vee l_2 \vee l_3) \wedge (l_1 \vee \overline{l_2} \vee \overline{l_3}) \wedge (\overline{l_1} \vee l_2 \vee \overline{l_3}) \wedge (\overline{l_1} \vee \overline{l_2} \vee l_3)$$

(2) A truth assignment satisfies $(l_1 \oplus l_2)$ if and only if it satisfies:

$$(l_1 \vee l_2) \wedge (\overline{l_1} \vee \overline{l_2})$$

So applying Proposition 2.1 to Equation (2.2) we introduce some variables, we will survey exactly the complexity values in the next subsection, and we produce a conjunction of 3-XOR-SAT clauses. The next step is to apply Proposition 2.2 to each 3-XOR-SAT clause. For each of such clauses this produces at most four 3-SAT clauses and therefore we are done reducing this part of the problem to 3-SAT.

2.1.1 Computational complexity

We are going to discuss the complexity of this part of the transformation considering as a complexity parameter the size $m \cdot n$ of the matrix H. The parity check matrix we are considering lies in $\mathcal{M}_{m,n}(\mathbb{F}_2)$ therefore from each row of the matrix we obtain a XOR clause made up by at most n literals l_i with $i \in 1, \ldots, n$. Let us get a closer look to one of such clauses. By

Proposition 2.1 the clauses can be split into $n-2$ 3-XOR clauses by adding $n-2$ new auxiliary variables. Thanks to Proposition 2.2 we can take this into a 3-SAT CNF formula. Indeed for each 3-XOR clause we need to satisfy 4 3-SAT clauses made up by the same literals. Therefore the final result, in the worst case, is a CNF formula made up by $4 \cdot (n-2) = 4n-8$ clauses. Since $H \in \mathcal{M}_{m,n}(\mathbb{F}_2)$ the final result of the reduction is the conjunction of m of the formulae just discussed. We get a total of $4mn-8m$ clauses. Thus the reduction of this part of the problem runs bounded by $\mathcal{O}(mn)$.

2.1.2 The example

We need to find a vector $v = (v_1, v_2, v_3, v_4)$ such that $\mathcal{H}v^\top = s$ and such that $w(v) \leq t$. The first step is to find the SAT translation of the $\mathcal{H}v^\top = s$ constraint.

$$\mathcal{H}v^\top = \begin{cases} v_1 + v_3 = 0 \\ v_1 + v_2 + v_4 = 1 \end{cases} = \begin{cases} v_1 \oplus \overline{v_3} = 1 \\ v_1 \oplus v_2 \oplus v_4 = 1 \end{cases}$$

Therefore the system becomes the XOR-SAT formula:

$$(v_1 \oplus \overline{v_3}) \wedge (v_1 \oplus v_2 \oplus v_4)$$

Even though we didn't explain the SAT to 3-SAT reduction, using Proposition 2.1 and Proposition 2.2 we can directly take the above equation to:

$$(v_1 \vee v_3) \wedge (\overline{v_1} \vee v_3) \wedge (v_1 \vee v_2 \vee v_4) \wedge (v_1 \vee \overline{v_2} \vee \overline{v_4}) \wedge (\overline{v_1} \vee v_2 \vee \overline{v_4}) \wedge (\overline{v_1} \vee \overline{v_2} \vee v_4) \quad (2.3)$$

2.2 Weight computation

We are going to tackle the problem of computing the weight of a given vector in $\mathbb{F}_2^n$. We give a short code snippet which computes the weight the input vector.

```
counter = 0                                                    1
for i = 1,..,n do:                                             2
        counter = counter + x[i]                               3
end for                                                        4
return counter;                                                5
```

Since encoding a for loop in SAT is unfeasible we need to write and encode every cicle explicitly. Consider thus the binary representation of the counter variable in the code above. Since the input vector $v = (v_1, \ldots, v_n)$ lies in $\mathbb{F}_2^n$ then the number of 1-coordinates sums up to a maximum of n, which means that the binary representation of our counter needs $l = \lceil \log_2(n) \rceil$ bits. Let $(a_1, \ldots, a_l)$ be the binary representation of the counter. Since we are splitting the for loop and the counter could eventually increase (and

therefore change) at every step, we will need $n+1$ different such a vectors representing the counter at each step.

$$\begin{cases} (a_1^{(0)}, \dots, a_l^{(0)}) \to \text{counter}^{(0)} = 0 \\ (a_1^{(1)}, \dots, a_l^{(1)}) \to \text{counter}^{(1)} = \text{counter}^{(1)} + v_1 \\ \vdots \\ (a_1^{(n)}, \dots, a_l^{(n)}) \to \text{counter}^{(n)} = \text{counter}^{(n-1)} + v_n \end{cases} \tag{2.4}$$

The reader shoud have noticed that the sums in the above equation is performed in the ring $\mathbb{Z}$. When $v_i = 0$ then no problem arises, but when $v_i = 1$ it does so then we need to encode such sum into binary sum. For the generic $counter^{(i)}$ for $i \in \{1, \dots, n-1\}$ the addition by 1 can be encoded as follows:

$$\begin{aligned} &a_1^{(i)} \leftarrow a_1^{(i-1)} \oplus 1 \\ &a_2^{(i)} \leftarrow a_2^{(i-1)} \oplus a_1^{(i-1)} \\ &a_3^{(i)} \leftarrow a_3^{(i-1)} \oplus a_1^{(i-1)} a_2^{(i)} \\ &\vdots \\ &a_l^{(i)} \leftarrow a_l^{(i-1)} \oplus a_1^{(i-1)} a_2^{(i-1)} \cdots a_{l-1}^{(i-1)} \end{aligned} \tag{2.5}$$

Notice that we perform this operation $n-1$ times. The reason is the following: assume that $(1, 1, \dots, 1)$ is a solution of $Hv^\top = s$. Since $t < n$ and $w(1, 1, \dots, 1) = n$ this couldn't be a solution of the MLD problem. However one can easily check that performing the above for n steps would set the n-th counter to $(0, 0, \dots, 0)$ which represents the integer $0 \leq t$ therefore $(1, 1, \dots, 1)$ is accepted as a correct solution. To get rid of this solution we perform only $n-1$ steps and we set an extra variable $a^{(n)}$ in case the solution vector is the all ones vector:

$$a^{(n)} = a_1^{(n-1)} a_2^{(n-1)} \cdots a_l^{(n-1)} \cdot v_n. \tag{2.6}$$

Lemma 2.3. *Given a vector $v \in \mathbb{F}_2^n$, the variable $a^{(n)}$ defined as in Equation (2.6) evaluates $1 \iff v = (1, 1, \dots, 1)$ i.e. $w(v) = n$.*

Proof. Assume $a^{(n)} = 1$ then

$$a_1^{(n-1)} = a_2^{(n-1)} = a_l^{(n-1)} = v_n = 1.$$

The number of 1 coordinates of x until v_{n-1} is represented by the vector $\left(a_1^{(n-1)}, a_2^{(n-1)}, \dots, a_l^{(n-1)}\right) = (1, 1, \dots, 1) = 2^l - 1 = n - 1$, so all of them are 1. Since also $v_n = 1$ then $v = (1, 1, \dots, 1)$.
Assume on the contrary that $v = (1, 1, \dots, 1)$. Then $\left(a_1^{(n-1)}, a_2^{(n-1)}, \dots, a_l^{(n-1)}\right)$ represents the count of 1 coordinates until v_{n-1} which is $n - 1 = 2^l - 1 = (1, 1, \dots, 1)$. Moreover, $v_n = 1$ then obviously $a^{(n)} = 1$ □

Therefore, the generic formula to compute $a_h^{(i)}$ for $h \in \{1, \ldots, l-1\}$ would be:

$$a_h^{(i)} \leftarrow a_h^{(i-1)} \oplus \prod_{j=1}^{h-1} a_j^{(i-1)}. \tag{2.7}$$

Since x_i can assume the value 0 or 1, the encoding of the equations in 1.1 becomes:

$$a_h^{(i)} \leftarrow a_h^{(i-1)} \oplus \left(\prod_{j=1}^{h-1} a_j^{(i-1)} \right) v_i. \tag{2.8}$$

These are XOR-SAT formulae are ready to be transformed into 3-SAT CNF formulae. To do this we apply again propositions (2.1) and (2.2). We do this in the complexity analysis section. In the following we prove a proposition which will be useful during the complexity analysis of this part of the transformation.

Proposition 2.4. *Let A and B be two 3-SAT propositional formulae. Let $\mathcal{C}_A = \{A_i\}_{i=1}^n$ and $\mathcal{C}_B = \{B_j\}_{j=1}^m$ be the clauses forming A and B respectively. Then we have*

$$A \vee B = \bigwedge_{i=1\ldots n, j=1\ldots m} (A_i \vee B_j), \tag{2.9}$$

which is again a well formed SAT formula.

Proof. Assume $P = (p_1 \vee p_2 \vee p_3)$ and $Q = (q_1 \vee q_2 \vee q_3)$ are two 3-SAT clauses then

$$P \vee Q = (p_1 \vee p_2 \vee p_3) \vee (q_1 \vee q_2 \vee q_3) = (p_1 \vee p_2 \vee p_3 \vee q_1 \vee q_2 \vee q_3) \tag{2.10}$$

is a well formed SAT clause. We prove the proposition by induction on i and setting $m = 1$ (induction on j is similar). Let then $i = 2$ so $A = A_1 \wedge A_2$ and $B = B_1$ so:

$$A \vee B = (A_1 \wedge A_2) \vee B_1 = (A_1 \vee B_1) \wedge (A_2 \vee B_1). \tag{2.11}$$

The right-hand side is the conjunction of two SAT clauses by the observation made above. Assume now the claim holds for a generic n, and consider $i = n + 1$. By associativity of conjunction and distributivity of disjunction over conjunctions we get:

$$\begin{aligned} &A \vee B = (A_1 \wedge \cdots \wedge A_{n+1}) \vee B_1 = \\ &= ((A_1 \wedge \cdots \wedge A_n) \wedge A_{n+1}) \vee B_1 = \\ &= ((A_1 \wedge \cdots \wedge A_n) \vee B_1) \wedge (A_{n+1} \vee B_1) = \\ &= \textstyle\bigwedge_{i=1\ldots n} (A_i \vee B_1), \end{aligned} \tag{2.12}$$

which is a well formed SAT formula. Now pick any value for m and substitute B_1 with B in the last term of Equation (2.12) to obtain $\bigwedge_{i=1\ldots n}(A_i \vee B)$. Notice that every term $(A_i \vee B)$ can be expanded with the same procedure as above. □

2.2.1 Computational complexity

We discuss now the complexity in terms of the number n of columns of H, and $l = \log_2(n)$. The "counting" part is basically made up as follows:

$$a_k^{(i)} = a_l^{(i-1)} \oplus a_1^{(i)} a_2^{(i)} \cdots a_{k-1}^{(i)} v_i, \tag{2.13}$$

that is, an equality, an EXCLUSIVE-OR and k consecutive AND. If we set $k = l$ we obviously obtain the worst case of the problem. We can express the XOR in terms of AND and OR as follows:

$$x \oplus y \iff (x \vee y) \wedge (\overline{x} \vee \overline{y}). \tag{2.14}$$

The equality can be expressed as:

$$x = y \iff (x \wedge y) \vee (\overline{x} \wedge \overline{y}). \tag{2.15}$$

We analyse Equation (2.14) setting $x = a_l^{(i-1)}$ and $y = a_1^{(i)} \wedge a_2^{(i)} \wedge \cdots \wedge a_{l-1}^{(i)} \wedge v_i$, and therefore $\overline{y} = \overline{a_1^{(i)}} \wedge \overline{a_2^{(i)}} \wedge \cdots \wedge \overline{a_{l-1}^{(i)}} \wedge \overline{v_i}$. Due to Proposition 2.4, it becomes:

$$(x \vee y) = (a_l^{(i-1)} \vee a_1^i) \wedge (a_l^{(i-1)} \vee a_2^i) \wedge \cdots \wedge (a_l^{(i-1)} \vee a_{l-1}^i) \wedge (a_l^{(i-1)} \vee v_i)$$

It is easy to see that the above equation is already a 3-SAT formula which is made up by l clauses of 2 literals each:

$$(\overline{x} \vee \overline{y}) = (\overline{a_l^{(i-1)}} \vee \overline{a_1^{(i)}} \vee \overline{a_2^{(i)}} \vee \cdots \vee \overline{a_{l-1}^{(i)}} \vee \overline{v_i}). \tag{2.16}$$

Equation (2.16) is a well formed SAT clause made up by $l + 1$ literals ready to be transformed into $l - 1$ 3-SAT clauses.

Therefore Equation (2.14) can be translated into $2l - 1$ 3-SAT clauses. Now we try to express eqation 1.14 into 3-SAT, so set $A = a_l^{(i-1)} \oplus a_1^{(i)} \wedge a_2^{(i)} \wedge \cdots \wedge a_{l-1}^{(i)} \wedge v_i$ and $B = a_l^{(i)}$. We've just seen that A is a 3-SAT formula made up by $2l - 1$ 3-SAT clauses, so we can rewrite it as:

$$A = \bigwedge_{k=1}^{2l-1} A_k,$$

and thus

$$\overline{A} = \bigvee_{k=1}^{2l-1} \overline{A_k}.$$

where a generic clause $A_k = (a_{k_1} \vee a_{k_2} \vee a_{k_3})$, and $\overline{A_k} = a_{k_1} \wedge a_{k_2} \wedge a_{k_3}$ becomes a formula made up by 3 atomic (single literal) clauses. Observe that

$$(B \wedge A) = B \wedge A_1 \wedge A_2 \wedge \cdots \wedge A_{2l-1}$$

is a 3-SAT formula made up by $2l$ clauses, and:

$$(\overline{B} \wedge \overline{A}) = \bigvee_{k=1}^{2l-1} \overline{A_i} \wedge \overline{B}$$

due to Proposition 2.4. Therefore,

$$(B \wedge A) = (B \wedge A_1 \wedge A_2 \wedge \cdots \wedge A_{2l-1}) \vee (\overline{A_1} \wedge \overline{B}) \vee \cdots \vee (\overline{A_{2l-1}} \wedge \overline{B})$$

Notice that every element in the RHS in the form $(\overline{A_i} \wedge \overline{B})$ is a SAT formula made up by 4 clauses. Therefore using Proposition 2.4 to expand it we obtain a SAT formula with $2l \cdot 4^{2l-1} = \frac{l \cdot n^4}{2}$. In terms of 3-SAT clauses we need to count the number of literals with repretitions appearing in such formula. The first part of the RHS contains B which is atomic and A_k which is made up by at most 3 literals, thus this part contains at most $3l$ literals. The elements of the form $(\overline{A_i} \wedge \overline{B})$ are made up by 4 literals. This means that every clause appearing in the result of Proposition 1.4 is made up by $\leq 3 + 2l - 1 = 2l + 2$ literals. This takes us to have at most $2l$ 3-SAT clauses for each clause in Proposition 2.4. This means that the total number of 3-SAT clauses needed to translate Equation (2.15) is upper bounded by $\frac{l \cdot n^4}{2} \cdot 2l = l^2 n^4$. We have l equation like in (2.13) and n registers therefore this part of the transformation runs in $\mathcal{O}(l^3 n^5)$.

2.2.2 The example

In this part we want to give a SAT representation of the weight calculator introduced in (2.5). In this case we have $l = \log_2 4 = 2$, therefore we need to compute the four registers $a^{(i)} = (a_1^{(i)}, a_2^{(i)})$ for $i \in \{1, 2, 3\}$. For a generic i we get:

$$\begin{aligned} a_1^{(i)} &= a_1^{(i-1)} \oplus v_i, \\ a_2^{(i)} &= a_2^{(i-1)} \oplus a_1^{(i-1)} v_i \end{aligned}$$

Using again Propositions (2.1) we obtain:

$$(\overline{a_1^{(i)}} \oplus a_1^{(i-1)} \oplus v_i) \wedge (\overline{a_2^{(i)}} \oplus a_2^{(i-1)} \oplus a_1^{(i-1)} v_i) \tag{2.17}$$

Now, applying proposition (2.2) to the first clause of (2.17) we obtain

$$(\overline{a_1^{(i)}} \vee a_1^{(i-1)} \vee v_i) \wedge (\overline{a_1^{(i)}} \vee \overline{a_1^{(i-1)}} \vee \overline{v_i}) \wedge (a_1^{(i)} \vee a_1^{(i-1)} \vee \overline{v_i}) \wedge (a_1^{(i)} \vee \overline{a_1^{(i-1)}} \vee v_i). \tag{2.18}$$

When we apply proposition (2.2) to the second clause we get:

$$\begin{aligned} &(\overline{a_2^{(i)}} \vee a_2^{(i-1)} \vee (a_1^{(i-1)} \wedge v_i)) \wedge \\ &(\overline{a_2^{(i)}} \vee \overline{a_2^{(i-1)}} \vee (\overline{a_1^{(i-1)}} \vee \overline{v_i})) \wedge \\ &(a_2^{(i)} \vee a_2^{(i-1)} \vee (\overline{a_1^{(i-1)}} \vee \overline{v_i})) \wedge \\ &(a_2^{(i)} \vee \overline{a_2^{(i-1)}} \vee (a_1^{(i-1)} \wedge v_i)) \end{aligned} \tag{2.19}$$

Finally, we have the extra formula introduced in Equation (2.6) that excludes $v \neq (1,1,1,1)$, which is

$$a_1^{(3)} \wedge a_2^{(3)} \wedge v_4. \tag{2.20}$$

2.3 Weight check

The sub-problem of determining wether the just computed weight is less or equal than a certain integer value is better described as follows:

Problem 2.5. *Given $t \in \mathbb{Z}$ is there a vector $c \in \mathbb{F}_2^n$ s.t. $w(c) \leq t$?*

Thus, given an integer $t = (t_1, t_2, \ldots, t_l)$ and the register containing the binary representation of the weight count $a = (a_1, a_2, \ldots, a_l)$ we want to check whether $t > a$. Assuming t_l and a_l are the most significant bits, we can write such constraint as:

$$(t_l > a_l) \vee [(t_l = a_l) \wedge (t_{l-1} > a_{l-1}) \vee [(t_{l-1} = a_{l-1}) \wedge (t_{l-2} > a_{l-2}) \vee [\ldots] \tag{2.21}$$

We've already seen that the above equation can be encoded into a SAT formula. Recall that the equalities like $t_l = a_l$ can be expressed as $(t_l \wedge a_l) \vee (\overline{t_l} \wedge \overline{a_l})$ while $t_l > a_l$ can be expressed as $t_l \wedge \overline{a_l}$, this is due to the fact that these are boolean literals.

2.3.1 Computational complexity

The complexity parameter here is the number of bits l of the binary representation of an in teger in $\{1, \ldots, n\}$. The innermost parenthesis is a SAT formula made up by 4 clauses. By proposition 1.4 every OR in this formula multiplies by 2 the total number of clauses, while each AND adds 2 clauses, which means that the total number of clauses sums up to:

$$((2^2) \cdot 2 + 2) \cdot 2 + 2) \ldots \cdot 2 + 2 = 2^l + 2^{l-1} + \cdots + 2 \leq l \cdot 2^l = l \cdot n \tag{2.22}$$

2.3.2 The example

At this point, the register $(a_1^{(4)}, a_2^{(4)})$ is the binary representation of the weight of the solution vector x and we want to compare it to $t = (t_1, t_2)$. So the comparison, according to Equation (2.21) is

$$(t_2 > a_2) \vee ((t_2 = a_2) \wedge (t_1 \geq a_1)),$$

which as a proper SAT formula becomes

$$(t_2 \wedge \overline{a_2}) \vee ((\overline{t_2} \vee a_2) \wedge (t_2 \vee \overline{a_2}) \wedge (t_1 \vee \overline{a_1})).$$

Since $t = 2$ we have $t_1 = 0$ and $t_2 = 1$ so we can evaluate the above formula on those values to get:

$$\overline{a_2} \vee (a_2 \wedge \overline{a_1}) = (\overline{a_2} \vee a_2) \wedge (\overline{a_2} \vee \overline{a_1}). \tag{2.23}$$

2.4 Natural reduction from SAT to 3-SAT

We can transform a SAT formula F into a well formed 3-SAT formula by "splitting" F and adding new literals. We consider each clause in F. In case the clause is made up by at most 3 literals we can leave it unchanged. In the case the number of literals is larger than 3 we proceed in a different way. If a clause contains 4 literals, we can split it as follows:

$$(v_1 \vee v_2 \vee v_3 \vee v_4) \rightarrow (v_1 \vee v_2 \vee u) \wedge (\overline{u} \vee v_3 \vee v_4), \tag{2.24}$$

where u is an auxiliary literal whose value depends on the values of the original literals. In case the clause is made up by 5 literals we can add 2 new literals and split it in this way:

$$(v_1 \vee v_2 \vee v_3 \vee v_4 \vee v_5) \rightarrow (v_1 \vee v_2 \vee u) \wedge (\overline{u} \vee v_3 \vee t) \wedge (\overline{t} \vee v_4 \vee v_5). \tag{2.25}$$

Therefore, assuming a clause contains n literals, this reduction introduces $n-3$ auxiliary literals and generates $n-2$ new 3-SAT clauses. It is straightforward to see that this method of generating 3 literals clauses works, as shown in the following lemma.

Lemma 2.6. *Let A and B be two boolean formulae and let z be a new variable. Then*

$$A \vee B$$

is satisfiable if and only if z can be assigned a value such that

$$(A \vee z) \wedge (\overline{z} \vee B)$$

is satisfiable with the same assignment to the variables appearing in A and B.

Proof. Assume first that $A \vee B$ is satisfiable. Then we can assume that A evaluates to $true$. Then we can set $z = false$ and the thesis easily follows. In case B is $true$ instead we can set $z = true$ to obtain the thesis. When both A and B are $true$ no matter what is the value of z.
Assume on the other hand that $(A \vee z) \wedge (\overline{z} \vee B)$ is satisfiable and assume by contraddiction that $A \vee B$ is not satisfiable. This means that $A = B = false$ therefore we must have $z = \overline{z} = true$ which is impossible. □

2.4.1 The example

In our example we need to apply such transformation only on the clauses of the CNF formula shown in Equation (2.19). We have that, according to the transformation explained in this section, the second and third clause of

Equation (2.19) become, for each $i \in 1, \ldots, 4$,

$$(\overline{a_2^{(i)}} \vee \overline{a_2^{(i-1)}} \vee (\overline{a_1^{(i-1)}} \vee \overline{v_i})) = (\overline{a_2^{(i)}} \vee \overline{a_2^{(i-1)}} \vee u_1^{(i)}) \wedge (\overline{u_1^{i}} \vee \overline{a_1^{(i-1)}} \vee \overline{v_i})$$

and

$$(a_2^{(i)} \vee a_2^{(i-1)} \vee (\overline{a_1^{(i-1)}} \vee \overline{v_i})) = (a_2^{(i)} \vee a_2^{(i-1)} \vee u_2^{(i)}) \wedge (\overline{u_2^{(i)}} \vee \overline{a_1^{(i-1)}} \vee \overline{v_i}) \tag{2.26}$$

Where $u_1^{(i)}$ and $u_2^{(i)}$ are new auxiliary literals according to the transformation we built. While taking advantage of Proposition 2.4 and applying it to the first and the fourth clauses of Equation (2.19), they become:

$$(\overline{a_2^{(i)}} \vee a_2^{(i-1)} \vee (a_1^{(i-1)} \wedge v_i)) = (\overline{a_2^{(i)}} \vee a_2^{(i-1)} \vee a_1^{(i-1)}) \wedge (\overline{a_2^{(i)}} \vee a_2^{(i-1)} \vee v_i)$$

and

$$(a_2^{(i)} \vee \overline{a_2^{(i-1)}} \vee (a_1^{(i-1)} \wedge v_i)) = (a_2^{(i)} \vee \overline{a_2^{(i-1)}} \vee a_1^{(i-1)}) \wedge (a_2^{(i)} \vee \overline{a_2^{(i-1)}} \vee v_i) \tag{2.27}$$

2.5 Conclusions

In this chapter we showed how to build a complete reduction from MLD to the 3-SAT problem. We split the starting problem into 3 smaller pieces, plus a piece that explains a SAT to 3SAT reduction, which are:

1. The parity check constraint, i.e. find a vector x such that $\mathcal{H}v^\top = s$, and its reduction to SAT is shown in Section (2.1).

2. The computation of the weight of such vector v is made in Section (2.2)

3. The check of the weight computed in Section (2.2) against t is performed in Section (2.3)

4. In Section (2.4) we shown an reduction from SAT to 3SAT, i.e. we depicted ψ.

With those pieces we can take a whole MLD instance into a 3-SAT instance. From the point of view of the computational complexity of this transformation we have that the whole process is dominated by the complexity of computing the weight of the solution vector v. This part of the process takes $\mathcal{O}(l^3 n^5)$ to be carried out. Therefore, as a conclusion, we can say that this transformation is upper bounded by the complexity of φ, that is $\mathcal{O}(l^3 n^5)$.

2.5.1 The example

Here we give the 3-SAT version of the initial instance of the MLD problem. It is all a matter of making a logical AND of the Equations (2.3), (2.23),(2.26), (2.20),(2.18) and (2.27) and the explicit result is:

$$
\begin{aligned}
&(v_1 \vee v_3)\wedge\\
&(\overline{v_1} \vee \overline{v_3})\wedge\\
&(v_1 \vee v_2 \vee v_4)\wedge\\
&(v_1 \vee \overline{v_2} \vee \overline{v_4})\wedge\\
&(\overline{v_1} \vee v_2 \vee \overline{v_4})\wedge\\
&(\overline{v_1} \vee \overline{v_2} \vee v_4)\\
&\overline{a_1^{(0)}}\wedge\\
&\overline{a_2^{(0)}}\wedge\\
&(\overline{a_1^{(i)}} \vee a_1^{(i-1)} \vee v_i)\wedge\\
&(a_1^{(i)} \vee \overline{a_1^{(i-1)}} \vee \overline{v_i})\wedge\\
&(a_1^{(i)} \vee \overline{a_1^{(i-1)}} \vee \overline{v_i})\wedge\\
&(\overline{a_1^{(i)}} \vee \overline{a_1^{(i-1)}} \vee v_i)\wedge\\
&(\overline{a_2^{(i)}} \vee \overline{a_2^{(i-1)}} \vee u_1^{(i)})\wedge\\
&(\overline{u_1^{i}} \vee \overline{a_1^{(i-1)}} \vee \overline{v_i})\wedge\\
&(\overline{a_2^{(i)}} \vee \overline{a_2^{(i-1)}} \vee u_2^{(i)})\wedge\\
&(\overline{u_2^{(i)}} \vee \overline{a_1^{(i-1)}} \vee \overline{v_i})\wedge\\
&(\overline{a_2^{(i)}} \vee a_2^{(i-1)} \vee a_1^{(i-1)})\wedge\\
&(a_2^{(i)} \vee \overline{a_2^{(i-1)}} \vee v_i)\wedge\\
&(a_2^{(i)} \vee \overline{a_2^{(i-1)}} \vee a_1^{(i-1)})\wedge\\
&(\overline{a_2^{(i)}} \vee \overline{a_2^{(i-1)}} \vee v_i)\wedge\\
&(\overline{a_1^{(3)}} \vee \overline{a_2^{(3)}} \vee \overline{v_4})\wedge\\
&(\overline{a_2} \vee a_2)\wedge\\
&(\overline{a_2} \vee \overline{a_1})
\end{aligned}
\tag{2.28}
$$

Clauses indexed by i must be added for every $i \in \{1, 2, 3\}$. We omitted this for reading semplicity and space reasons.

Chapter 3

A 3-SAT to MQ problem reduction

In this chapter we explicit a reduction from 3-SAT to a system of boolean quadratic equations. In other words we want to explicit the transformation ω of diagram 1. Given an instance of 3-SAT, we use, on each clause, the transformation $T : \mathcal{C} \longrightarrow \mathbb{F}_2[x_1, \ldots, x_n]$ given by:

$$\begin{array}{l} T(x_i) = 1 - x_i \\ T(\overline{x_i}) = x_1 \\ T(\lambda_1 \vee \lambda_2 \vee \lambda_3) = T(\lambda_1) \cdot T(\lambda_2) \cdot T(\lambda_3) \end{array} \tag{3.1}$$

The degree of the resulting polynomials could be 3 in the case of clauses with 3 literals. Let us make an example. Consider the simple formula

$$F = (x_1) \wedge (x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee \overline{x_3} \vee x_4)$$

We apply T to each clause above obtainging:

$$\omega(F) = \begin{cases} T(x_1) = (1 - x_1) \\ T(x_2 \vee \overline{x_3}) = (1 - x_2)x_3 = x_2x_3 + 1 \\ T(\overline{x_2} \vee \overline{x_3} \vee x_4) = x_2x_3(1 - x_4) = x_2x_3x_4 + x_2x_3 \end{cases}$$

So now we want to construct a polynomial system P, with quadratic boolean polynomials, therefore the first and the second equations can simply be put inside P while the third equation needs some further work since its degree is 3. All we need to do is to add a new auxiliary variable z, such that $z = x_2x_3$. P becomes:

$$P = \begin{cases} x + 1 = 0 \\ z + 1 = 0 \\ zx_4 + z = 0 \\ z + x_2x_3 = 0. \end{cases}$$

With the change of variables above, we took P to be a system of quadratic equations over $\mathbb{F}_2$. Therefore ω, along with an eventual change of variables,

forms a reduction from 3-SAT to MQ. Since in a 3-SAT formula there are clauses with at most 3 literals, the maximum degree of an equation created through T is 3. A variable substitution as in the example introduces only one variable and a new quadratic equation.

3.1 Computational complexity

The transformation T described in Equation (3.1) generates a new equation from each clause in the CNF formula. The degree of such equations depends on the number of literals contained in a clause. Since we are working with 3-SAT CNF formulae, a clause can be made up by up to three literals. By definition of T we have to distinguish in two cases:

- in case the clause is made up by one or two we have that T produces a linear equation and a quadratic equation namely,
- in the case of a clause with 3 literals T produces a degree 3 equation.

In any case there will be only one monomial of degree 3, exactly as we did in the above example. The complexity parameter we are going to consider in this transformation is the number of clauses that make the initial 3-SAT CNF formula. Therefore, a conjunction of m clauses, in the worst case we produce m new variables and a total of $2m$ equations. Therefore ω runs in polynomial time in terms of clauses of the initial formula.

3.2 The example

In this section we take the results from chapter 2 and we apply the transformation we just described in this chapter. In the case a clause is indexed using $i \in \{1, 2, 3\}$ we apply the transformation just one time for a generic i for the sake of simplicity. Therefore, the resulting system according to the

CNF formula in (2.28) is the following:

$$S = \begin{cases} (v_1+1)(v_3+1)=0 \\ v_1v_3=0 \\ (v_1+1)(v_2+1)(v_4+1)=0 \\ (v_1+1)v_2v_4=0 \\ v_1(v_2+1)v_4=0 \\ v_1v_2(v_4+1)=0 \\ a_1^0=0 \\ a_2^0=0 \\ a_1^{(i)}(a_1^{(i-1)}+1)(v_i+1)=0 \\ a_1^{(i)}a_1^{(i-1)}v_i=0 \\ (a_1^{(i)}+1)(a_1^{(i-1)}+1)v_i=0 \\ (a_1^{(i)}+1)a_1^{(i-1)}(v_i+1)=0 \\ a_2^{(i)}a_2^{(i-1)}(u_1^{(i)}+1)=0 \\ u_1^{(i)}a_1^{(i-1)}v_i=0 \\ (a_2^{(i)}+1)(a_2^{(i-1)}+1)(u_2^{(i)}+1)=0 \\ u_2^{(i)}a_1^{(i-1)}v_i=0 \\ a_2^{(i)}(a_2^{i-1}+1)(a_1^{(i-1)}+1)=0 \\ a_2^{(i)}(a_2^{(i-1)}+1)(v_i+1)=0 \\ (a_2^{(i)}+1)a_2^{(i-1)}(a_1^{(i-1)}+1)=0 \\ (a_2^{(i)}+1)a_2^{(i-1)}(v_i+1)=0 \\ a_2^{(3)}a_2^{(3)}v_4=0 \\ a_2^{(3)}(a_2^{(3)}+1)=0 \\ a_2^{(3)}a_1^{(3)}=0 \end{cases} \tag{3.2}$$

For the sake of reading simplicity, we omit the variables sostitutions discussed above in order to transform the 3-rd degree equations into quadratic equations. Projecting the solutions on the first four coordinates, will produce the solutions of the initial MLD instance.

Chapter 4

A direct MLD to MQ reduction

In this chapter we want to discuss the direct reduction α from MLD to MQ. Therefore starting from an instance of the MLD problem we want to build a system P of boolean quadratic equations. We want a solution to P to be solution of the original MLD instance, through an eventual transformation.

4.1 Parity check constraint

We want to find a collection of boolean quadratic equations that express the parity check constraint, that is, as we've already seen in section (2.1), $Hv^\top = s$. Observe that $Hv^\top = s$ is a set of equations of the form:

$$\begin{array}{l} \sum h_{i,j}v_j = 0 \text{ if } s_i = 0 \\ \sum h_{i,j}v_j = 1 \text{ if } s_i = 1. \end{array} \tag{4.1}$$

These are boolean linear equations, and this means that we can put such m equations into P without any change.

4.1.1 Computational complexity

This part of the transformation generates the same number of linear equations as the number of rows of the parity check matrix H. Therefore, since $H \in \mathcal{M}_{m,n}(\mathbb{F}_2)$ we obtain exactly m linear equations.

4.2 Weight Computation

This section aims at the computation of the weight of the solution vector v. We want to obtain a vector $a = w(v) = (a_1, \ldots, a_l)$, where $a_i \in \mathbb{F}_2$, which is the binary representation of the integer weight of the vector. To perform this we proceed similarly to the reduction in section 2.2.Recall that

our method creates a new register $a^{(i)} = \left(a_1^{(i)}, \ldots, a_l^{(i)}\right)$ every time we add v_i therefore we can make use of the same set of equations which are:

$$\begin{aligned} a_1^{(i)} &= a_1^{(i-1)} + v_i \\ a_2^{(i)} &= a_2^{(i-1)} + a_1^{(i-1)} \cdot v_i \\ a_3^{(i)} &= a_3^{(i-1)} + a_1^{(i-1)} a_2^{(i-1)} \cdot v_i \\ &\vdots \\ a_l^{(i)} &= a_l^{(i-1)} + a_1^{(i-1)} a_2^{(i-1)} \cdots a_{l-1}^{(i-1)} \cdot v_i \end{aligned} \tag{4.2}$$

for $i \in \{1, \ldots, n-1\}$ and setting $a_1^{(0)} = \cdots = a_l^{(0)} = 0$. Notice that the largest degree equation, which is $a_l^{(i)}$ for each i, has degree l. We need to transform these higher degree equations into a set of quadratic equations. To do this we perform changes of variables introducing new auxiliary variables and generating quadratic equations. The worst case $a_l^{(i)}$ needs the introduction of $l-2$ new variables and we get a total of $l-1$ quadratic equations. Therefore for a single register $a^{(i)}$ we obtain a set of at most l^2 quadratic equations. Since we need to produce n such registers, this process is bounded by $\mathcal{O}(n \cdot l^2)$. Notice that for $i = 0$ we added n linear equations.

4.3 Weight constraint

In what follows we give a representation, in terms of a single equation, of the constraint $a \leq t$ where $a = (a_1, a_2, \ldots, a_l)$ and $t = (t_1, t_2, \ldots, t_l)$ are the binary representations of the counter introduced above and of the integer in the MLD problem definition. First of all we set the following new variables:

$$g_h = (t_h + a_h), \quad h \in \{1, \ldots, l\}. \tag{4.3}$$

Furthermore we set the following l equations:

$$f_j = \left(\prod_{h=j+1}^{l} (g_h + 1) \right) g_j. \tag{4.4}$$

This set of equation $\{f_j\}_{j=1}^{l}$ aims at finding the first bit in the two binary representations above, assuming the most significant bits are t_l and a_l, that differs. Therefore at most one of them will evaluate to 1 when applied to t and a, as we see in the following lemma. If $f_j = 0$ then $t = a$, while, whenever a f_j results to 1, we want to check whether $t_j = 1$ or not. In the first case $a < t$ in the second case $a > t$.

Lemma 4.1. *Given a set of polynomials $\{f_j\}_{j=1}^{l}$ where each f_j is constructed as in (4.4), then at most one of the f_j will result equal to 1 when evaluated in $a, t \in (\mathbb{F}_2)^l$.*

Proof. Assume that $f_i = f_j = 1$, we can assume w.l.o.g. that $i < j$. This means that

$$f_j = \left(\prod_{h=j+1}^{l} (g_h + 1) \right) g_j = 1,$$

implying $g_j = 1$. By definition, f_i contains the factor $(g_j + 1) = 0$ since $i < j$. Therefore the thesis follows. □

Lemma 4.2. *$f_j = 0$ for every $j \in 1, \ldots, l$ if and only if $a = t$.*

Proof. Assume $f_j = 0$ for every $j \in 1, \ldots, l$. We prove this by induction. Assume first l = 1 We have that $f_1 = g_1 = 0$ therefore $(a_1 + t_1) = 0$ which means $a_1 = t_1$. Notice f_j are constructed backwards. Assume this holds for a generic value of l, we prove it for $l+1$. We have that $a_2 = t_2, \ldots, a_{l+1} = t_{l+1}$ which implies $g_h = 0$ for every $h \in \{2, \ldots, l+1\}$ therefore $(g_h + 1) = 1$ for every $h \in \{2, \ldots, l+1\}$. Hence, since we assume $f_1 = 0$, we have $g_1 = 0$ which means $a_1 = t_1$.
On the other hand assume $a = t$ then $g_h = 0$ for every $h \in 1, \ldots, l$ by definition of f_j this concludes the proof. □

Lemma 4.3. *Assume $f_j = 1$. Then j is the most significant bit where a and t differ.*

Proof. Assume by contradiction that $a_i \neq t_i$ for $i > j$. Then, we would have $g_i = 1$. But f_j contains the factor $(g_i + 1)$ which sould be 0, hence $f_j = 0$. □

Lemma 4.4. *The evaluation of the polynomial*

$$f = \sum_{j=1}^{l} f_j(t_j + 1) \tag{4.5}$$

in $t = (t_1, \ldots, t_l)$ is equal to 0 if and only if $a \leq t$.

Proof. Assume f = 0, this means that every factor $f_j(t_j + 1) = 0$. We have two possibilities, either $f_j = 0 \quad \forall j \in \{1, \ldots .l\}$ or there is a unique $f_j = 1$ for some value of j. In the first case by Lemma (4.2) we have that $a = t$. In the second case let $f_j = 1$, implying that $(t_j + 1) = 0$. This means $t_j = 1$ and $a_j = 0$, since otherwise, if $a_j = 1$, then $g_j = 0$ and $f_j = 0$ which leads to a contraddiction. Since by Lemma (4.3) this is the most significant bit in which a and t differ, we have that $a < t$.
Assume on the other hand that $a \leq t$. We distinguish two cases.
In case $a = t$ then, by Lemma (4.2) $f_j = 0$ for every $j \in \{1, \ldots, l\}$ and it follows that $f = 0$.
In case $a < t$ then there exist j such that $t_j = 1$ and $a_j = 0$ and $a_i = t_i$ for every $j < i \leq l$. Therefore f_j is the only equation to be 1, but $(t_j + 1) = 0$ and $f = 0$. □

Therefore, to express the constraint $a \leq t$ we just need to add the equation $\sum_{j=1}^{l} f_j(t_j + 1) = 0$.

4.3.1 Computational complexity

Let us analyise how the polynomial in Equation (4.5) is made up. We do this in tems of the input size i.e. $m \cdot n + m + l$, that is the size of the matrix H, of the syndrome s and the binary representation of t, namely. Notice that, when we are given an instance of the MLD problem, the value of the integer $t = (t_1, \ldots, t_l)$ is given and, by construction, we have that $g_k \in \mathbb{F}_2[a_h]$ for every value of $h \in \{1, \ldots, l\}$. Therefore g_h has at most two monomials, depending on the value of $t_k \in \mathbb{F}_2$. When we build f_j we have that the factor $(g_k + 1)$ for $h \in \{j + 1, \ldots, l\}$ is still made up by two monomials at most. As a consequence $f_j \in \mathbb{F}_2[a_j, \ldots, a_l]$ and has degree $\deg(f_j) = l - j + 1$. The number of monomials in it is at most 2^{l-j+1}. This means that f_1 has the largest degree that is $\deg(f_1)l + 1$. Finally f is just the sum over j of the f_j , so $f \in \mathbb{F}_2[a_1, \ldots, a_l]$ and its degree is $\deg(f) = \deg(f_1) = l + 1$.
What is left to do is to reduce f to a set of quadratic boolean equations. So for the largest degree monomial we need to add $l - 1$ new variables and obtain l quadratic equations. The number of monomials of f is at most 2^{l+1} hence the number of equations generated to reduce it to a set of quadratic equations sums up to $l \cdot 2^{l+1} = 2ln$

4.4 The example

We consider again the instance of the MLD problem we proposed in chapter 2, and we apply the direct transformation we have introduced in this chapter. We perform the two steps explained into sections (4.1) and (4.3) and give a system of boolean equations that reflect the starting instance of the MLD problem. The parity check constraints results into the following system:

$$\mathcal{H}x^\top = s \quad \Longrightarrow \quad P = \begin{cases} v_1 + v_3 = 0 \\ v_1 + v_2 + v_4 + 1 = 0 \end{cases}$$

This is a well formed system of boolean quadratic equations, so it can be left as it is. The next step is to consider the system of equations needed to compute the weight of the vector x. We can take the equations we computed

in Equation (4.2) therefore we obtain

$$P = \begin{cases} v_1 + v_3 = 0 \\ v_1 + v_2 + v_4 + 1 = 0 \\ a_1^{(0)} = 0 \\ a_2^{(0)} = 0 \\ a_1^{(1)} + a_1^{(0)} + v_1 = 0 \\ a_2^{(1)} + a_2^{(0)} + a_1^{(0)} \cdot v_1 = 0 \\ a_1^{(2)} + a_1^{(1)} + v_2 = 0 \\ a_2^{(2)} + a_2^{(1)} + a_1^{(1)} \cdot v_2 = 0 \\ a_1^{(3)} + a_1^{(2)} + v_3 = 0 \\ a_2^{(3)} + a_2^{(2)} + a_1^{(2)} \cdot v_3 = 0 \\ a_1^{(3)} \cdot a_2^{(3)} \cdot v_4 = 0 \end{cases}$$

Now we compute the equation f that has been introduced in Equation (4.5) and add it into P. Since we already have the values for t_1 and t_2 we can simplify the equations. First of all we compute the g_k's for $k \in 1, 2$ which are:

$$g_1 = (a_1^{(3)} + t_1) = a_1^{(3)}$$
$$g_2 = (a_2^{(3)} + t_2) = a_2^{(3)} + 1$$

The next step of the transformation is to compute the f_i defined in Lemma (4.1) equations, i.e.:

$$f_1 = (g_2 + 1) \cdot g_1 = a_2^{(3)} \cdot a_1^{(3)}$$
$$f_2 = g_2 = a_2^{(3)} + 1$$

Finally, we compute the boolean function f to obtain:

$$f = f_1(t_1 + 1) + f_2(t_2 + 1) = a_2^{(3)} \cdot a_1^{(3)}$$

Observe that $a > t$ only in the case $a = (a_1^{(3)}, a_2^{(3)}) = (1, 1) = 3$, hence $f = 0$ if and only if $a < 3$, which is what we are looking for. Putting everything together the system P becomes:

$$S = \begin{cases} v_1 + v_3 = 0 \\ v_1 + v_2 + v_4 + 1 = 0 \\ a_1^{(0)} = 0 \\ a_2^{(0)} = 0 \\ a_1^{(1)} + a_1^{(0)} + v_1 = 0 \\ a_2^{(1)} + a_2^{(0)} + a_1^{(0)} \cdot v_1 = 0 \\ a_1^{(2)} + a_1^{(1)} + v_2 = 0 \\ a_2^{(2)} + a_2^{(1)} + a_1^{(1)} \cdot v_2 = 0 \\ a_1^{(3)} + a_1^{(2)} + v_3 = 0 \\ a_2^{(3)} + a_2^{(2)} + a_1^{(2)} \cdot v_3 = 0 \\ a_1^{(3)} \cdot a_2^{(3)} \cdot v_4 = 0 \\ a_2^{(3)} \cdot a_1^{(3)} = 0 \end{cases}$$

The following code snippet computes a Groebner Basis of the system S and then compute the variety of the ideal generated by such basis. Once we have computed variety, we project each vector on the first four coordinates, which correspond to $v_1, \ldots, v_4$. The resulting vectors are the solutions of the initial instance of the MLD problem.

```
R<x_1,x_2,x_3,x_4,a_01,a_02,a_11,a_12,a_21,a_22,a_31,a_32>
    := BooleanPolynomialRing(12);
P := [
    x_1 + x_3,
    x_1 + x_2 + x_4 + 1,
    a_01,
    a_02,
    x_1 + a_01 + a_11,
    x_2 + a_11 + a_21,
    x_3 + a_21 + a_31,
    x_4 + a_31 + a_41,
    x_1*a_01 + a_02 + a_12,
    x_2*a_11 + a_12 + a_22,
    x_3*a_21 + a_22 + a_32,
    x_4*a_31 + a_32 + a_42,
    a_41*a_42,
    x_4*a_31*a_32
]
B := GroebnerBasis(S);
I := ideal<R|B>;
Variety(I);
[
        <0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0>,
        <0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0>,
        <1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1>
 ]
```

So projecting on the first four coordinates we obtain:

$$v_1 = (0, 0, 0, 1)$$
$$v_2 = (0, 1, 0, 0)$$
$$v_3 = (1, 0, 1, 0)$$

We have that $\mathcal{H}v_i^\top = s$ and $w(v_i) \leq 2$ for every $i \in \{1, 2, 3\}$ so these are solutions of the original MLD instance.

4.5 Conclusions

As a conclusion we can say that through this procedure we obtain a system of boolean quadratic equations whose solutions restricted to the first n coordinates give solutions to the initial MLD instance. The complexity of the total transformation α is dominated by the weight computation part which takes $\mathcal{O}(l^2 n)$.

Chapter 5

A direct reduction from MQ to MLD

The purpose of this chapter is to describe a direct reduction of MQ into MLD, that is the transformation β of diagram 1. This means that, given a system of quadratic boolean equations, we want to build a parity check matrix H for a linear code, a syndrome s and a weight t such that a solution to such instance of MLD is, through an eventual transformation, a solution also for the original system.
First of all we analyse the possible equations that could appear into the MQ system. Recall that the maximum degree of an equation is 2 and that a monomial of degree 2 is of the form $x_i x_j$. Assume we are working in an environment with n variables and a system with m equations. Consider an equation in n variables of degree 2 in $\mathbb{F}_2[x_1, \ldots, x_n]$ with the maximum number of monomials,that is the worst case for us. It looks like:

$$p = \sum_{i,j \in \{1,\ldots n\}} x_i x_j + \sum_{i \in \{1,\ldots n\}} x_i + c \tag{5.1}$$

We want to reduce Equation (5.1) to a system which contains only two types of equations, either

$$xy + z = 0 \tag{5.2}$$

or

$$a + b + c = 0. \tag{5.3}$$

Furthermore, we also want that the equations of the form as in (5.2) do not share variables. On the other hand, we want that the variables in equations of the form (5.3), are present in at least one equation of the form as in (5.2).

Definition 5.1. *A system of equations is said in* ***standard form*** *if it contains only equations as in* (5.2) *and* (5.3)*, where the quadratic equations do not share any variable and the linear equations share every variable with at least one quadratic equation.*

Example 5.2. *The system*

$$P = \begin{cases} xy + r = 0 \\ zq + w = 0 \\ r + z + w = 0 \end{cases}$$

is in standard form.

Example 5.3. *The system*

$$P = \begin{cases} zq + w = 0 \\ r + z + w = 0 \end{cases}$$

is ***not*** *in standard form.*

Let us make a short example

Example 5.4. *Consider the system*

$$P = \{ \ xy + z + w = 0,$$

and set $t = xy$, moreover set the new variables to $z'z'' = z$ and $w'w'' = w$. The system P becomes than

$$P = \begin{cases} xy + r = 0 \\ z'z'' + z = 0 \\ w'w'' + w = 0 \\ r + z + w = 0, \end{cases}$$

which is in standard form.

Equation (5.1) has exactly $\binom{n}{2} = \frac{n\cdot(n-1)}{2}$ monomials of degree 2 and n linear monomials plus a constant term c. For each degree 2 monomial $x_i x_j$ we introduce a new variable z_{ij} in order to obtain an equation of the form $x_i x_j + z_{ij} = 0$. Therefore in this step we introduced $\frac{n\cdot(n-1)}{2}$ new variables and obtained the same number of equations of this form.
Notice that, in order to obtain a system in standard form we need that such kind of equations do not share any variable, see definition 5.1. Notice that each variable x_i is contained in $n-1$ monomials for each $i \in \{1, \dots, n\}$. Therefore, for each $i \in \{1, \dots, n\}$, introducing the variables $x_i^{(h)}$ and the equations $x_i + x_i^{(h)} = 0$ for $1 \leq h \leq n-2$ we obtain the desired result. So we introduced $n(n-2)$ new variables and equations.
Now the equation p is a linear polynomial in the variables z_{ij} and x_i for $i, j \in 1, \dots, n$ with exactly $\frac{n\cdot(n-1)}{2} + n = \frac{n\cdot(n+1)}{2}$ monomials. Here we introduce $\frac{n\cdot(n+1)}{2} - 2$ new variables in order to obtain $\frac{n\cdot(n-1)}{2} - 1$ equations as in (5.3). These new variables do not appear in any quadratic equation, therefore for each of them we need to further introduce an equation of the form (5.2) for a total of $\frac{n\cdot(n+1)}{2} - 2$ equations.

Note 5.5. *Any boolean quadratic system can be taken to standard form.*

Note 5.5 can be proved by applying the above steps to each equation of a given system. This introduces new auxiliary variables and equations.

We split the problem of fiding H in two parts. We will find a matrix H_1 for each equation of the form (5.2) and we construct a block matrix. After this step we will add a row r_j to such matrix for every linear equation of the form (5.3). The resulting matrix looks like

$$H = \begin{bmatrix} H_1 & 0 & \cdots & 0 \\ 0 & H_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & H_1 \\ & & \vdots & \\ & & r_j & \\ & & \vdots & \end{bmatrix} \tag{5.4}$$

Let's make a short example to make things more clear.

Example 5.6. *Consider the standard form system given by*

$$P = \begin{cases} xy + r = 0 \\ zq + w = 0 \\ r + z + w = 0 \end{cases}$$

We give explicitly H_1 as parity check matrix, s_1 as syndrome and t_1 as weight for the first and second equation, namely. We give a row vector r_1 and a bit δ (to be attached to the resulting syndrome) for the third equation.

$$H = \begin{bmatrix} H_1 & 0 \\ 0 & H_1 \\ r_1 & \end{bmatrix} \tag{5.5}$$

We compose syndromes like:

$$s = \begin{pmatrix} s_1 \\ s_1 \\ \delta \end{pmatrix} \tag{5.6}$$

while as a weight we take $t = 2 \cdot t_1$.

Consider equations as in (5.2), i.e. $xy + z = 0$, and let $I = \langle xy + z \rangle$ be the ideal generated by it. The variety associated to I over $\mathbb{F}_2$, $\mathcal{V}_{\mathbb{F}_2}(I)$ is

$$\begin{array}{ccc} x & y & z \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1. \end{array} \tag{5.7}$$

Our aim is to give a matrix H and a syndrome vector s such that every vector v that solves $xy + z = 0$ is a solution also of $Hv^\top = s$. Notice that for a linear system $Hv^\top = s$ every linear combination of the above vectors will be a solution of $Hv^\top = s$. Therefore the set of solution of such system is:

$$\begin{matrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} \tag{5.8}$$

In order to distinguish among the ones that solve $xy + z = 0$, we introduce the concept of weight, i.e. we want the right solutions to satisfy $\mathrm{w}(v) \leq t$ for a certain value of t. The idea is to extend the solution space to be of the form $\mathbb{F}_2^h$ for some h and charge the unwanted solutions with extra weight while leaving the desired ones unchanged or as light as possible. The solution space we designed in this case lies in $(\mathbb{F}_2)^{10}$ and in particular is the set:

$$S = \left\{ \begin{array}{ccc|cccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \right\} \tag{5.9}$$

Proposition 5.7. *S is a coset of a vector subspace of $(\mathbb{F}_2)^n$*

Proof. Translate S by the vector

$$(0, 0, 0, 0, 0, 0, 0, 1, 1, 1)$$

□

Proposition 5.8. *The vector subspace S in lemma* (5.7) *is a linear binary code generated by the matrix*

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \tag{5.10}$$

Proof. The generator matrix G spans the whole space

$$S + (0, 0, 0, 0, 0, 0, 0, 1, 1, 1).$$

□

Notice that we can consider the vector $e = (0,0,0,0,0,0,0,1,1,1)$ above as an error vector. We can reduce G to systematic form so it is easy to compute the party check matrix for such code. It is known that given a generator matrix of an $[n,k]$-code of the form $G = [\mathbb{I}_k|M]$, where $\mathbb{I}_k$ denotes the identity matrix of dimension k, we can compute the parity check matrix as $H_1 = [-M^\top|\mathbb{I}_{n-k}] = [M^\top|\mathbb{I}_{n-k}]$. Therefore H_1 becomes:

$$H_1 = \left[\begin{array}{ccc|c} 1 & 0 & 1 & \\ 1 & 0 & 1 & \\ 0 & 1 & 1 & \\ 0 & 1 & 1 & \mathbb{I}_7 \\ 1 & 1 & 1 & \\ 1 & 1 & 1 & \\ 1 & 1 & 1 & \end{array}\right] \tag{5.11}$$

Given the error vector $e = (0,0,0,0,0,0,0,1,1,1)$ we can compute its syndrome s_1

$$s_1 = H_1 e^\top = (0,0,0,0,1,1,1). \tag{5.12}$$

Let us denote by $\pi_3 : (\mathbb{F}_2)^{10} \longrightarrow (\mathbb{F}_2)^3$ the projection defined as

$$\pi_3(v_1, v_2, \ldots, v_{10}) = (v_1, v_2, v_3). \tag{5.13}$$

Proposition 5.9. *If $v \in (\mathbb{F}_2)^{10}$ is a solution of the equation $H_1 v^\top = s_1$ as in (5.7) then $\pi_3(v)$ solves $xy + z = 0$ if and only if $\mathrm{w}(v) = 3$.*
If $\pi_3(v)$ does not solve $xy + z = 0$ then $\mathrm{w}(v) \geq 5$.

Proof. The solutions of $H_1 v^\top = s$ are the rows of the set S of (5.9). The vectors v_i such that $\pi_3(v_i)$ solves $xy + z = 0$ are

$$\begin{array}{l} v_1 = (0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 1\ \ 1) \\ v_2 = (1\ \ 0\ \ 0\ \ 1\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0) \\ v_3 = (0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 1\ \ 1\ \ 0\ \ 0\ \ 0) \\ v_4 = (1\ \ 1\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0). \end{array}$$

We have that $\mathrm{w}(v_i) = 3$ for every $i \in \{1, \ldots, 4\}$. The remaining rows of S have weight larger than 5. □

Proposition 5.9 says that the first three coordinates in each vector are the values to be given to x, y and z. Thus the first three columns of H_1 are the coefficients of x, y and z in the system $H_1 v^\top = s$.

Proposition 5.10. *Using H_1, $s_1 = (0,0,0,0,1,1,1)$ as a syndrome and $t_1 = 3$ as a weight we built an instance of MLD corresponding to the MQ system containing the single equation $xy + z = 0$.*

Proof. Solving $H_1 x^\top = s_1$, selecting the ones with weight at most 3 and projecting the solution space on the first three coordinates we obtain exactly the vectors in Equation (5.7). □

We see how to treat the equation of the form (5.3) in the following example.

Example 5.11. *This example is the continuation of example 5.6. The parity check matrix H according to* (5.4) *becomes*

$$H = \left[\begin{array}{ccc|ccc|ccc|c} 1&0&1 & & & & & & & \\ 1&0&1 & & & & & & & \\ 0&1&1 & & & & & & & \\ 0&1&1 & \multicolumn{3}{c|}{\mathbb{I}_7} & \multicolumn{3}{c|}{0} & \\ 1&1&1 & & & & & & & \\ 1&1&1 & & & & & & & \\ 1&1&1 & & & & & & & \\ \hline & & & & & & 1&0&1 & \\ & & & & & & 1&0&1 & \\ & & & & & & 0&1&1 & \\ & 0 & & & & & 0&1&1 & \mathbb{I}_7 \\ & & & & & & 1&1&1 & \\ & & & & & & 1&1&1 & \\ & & & & & & 1&1&1 & \\ \hline & & & \multicolumn{9}{c}{r_1} \end{array}\right]$$

where we set $r_1 = (0,0,1,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0)$. *The* 1 *coordinates in* r_1 *are the one corresponding to* r, z *and* w. *Set also* $b = 0$ *and therefore the resulting syndrome becomes*

$$s = (s_1, s_1, \delta) = (0,0,0,0,1,1,1,0,0,0,0,1,1,1,0).$$

Set also $t = 2 \cdot 3 = 6$.

5.1 A new family of codes

The transformation depicted in this chapter takes a system of boolean quadratic polynomial equations to an instance of the MLD problem. This means that such transformation involves the construction of a new family of codes. We are going to analyse such new codes from the point of view of the length, dimension and minimum distance.

To begin with let us recall an important result in coding theory:

Theorem 5.12. *Let C be a linear code with parity check matrix H. The distance $d(C) = d$ if and only if every set of $d-1$ columns of H is linearly independent and some set of d columns of H is linearly dependent.*

Proof. Assume first that $d(C) = d$. Recall that the distance of a code is also the minimum weight of the codewords. Assume also, by contrary, that there

exists a set $\{h_{i_1}, \ldots, h_{i_{d-1}}\}$ of columns of H that are linearly independent. This means that there are some $\alpha_1, \ldots, \alpha_{d-1}$ such that:

$$\sum_{j=1}^{d-1} \alpha_j \cdot h_{i_j} = 0$$

So if we build the vector a where we set the coordinates $a_{i_j} = \alpha_j$ and set the others to 0, so $w(a) = d - 1$. According to what we said we have that $Ha^\top = 0$ so $a \in C$ but this contraddicts the minimality of the distance d. Pick now $c \in C$ such that $w(c) = d$, or in other words c has d non-zero coordinates, let them be $\{c_{k_1}, \ldots, c_{k_d}\}$. Since $Hc^\top = 0$ this means that:

$$\sum_{j=1}^{d} c_{k_j} \cdot h_{i_j} = 0$$

which is a linear dependence relation between a set of d columns of H.
On the other hand assume that every set of $d - 1$ columns of H is linearly independent and there is a set of d columns that is not. So again we can find $\alpha_1, \ldots, \alpha_d$ that defines a linear dependence relation. We can build as we did before a vector a with such coefficients and we see that it belongs to the code, it has weight $w(a) = d$ and this corresponds to the minimum distance since d is the minimum value such that we can do this procedure. □

Let us take advantage of Theorem 5.12 to compute the distance of the code we introduced. We defined the code by giving the structure of the parity check matrix H. Since this is a block matrix any set of linearly independent columns must lie in the same block. For this reason we can reduce the problem of counting the number of independent columns to a single block.

Lemma 5.13. *The code defined by the parity check matrix H_1 defined in* (5.11) *is a [10,3,4] linear code over $\mathbb{F}_2$.*

Proof. By Proposition 5.8 we get that the lenght of the code is 10 and its dimension is 3. Notice that every set of 3 columns of H_1 is linearly independent while we have that

$$\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Therefore by Theorem 5.12 the distance is 4. □

Proposition 5.14. *The code C defined by the parity check matrix H has distance $d(C) \geq 4$.*

Proof. By Lemma 5.13 every set of 3 columns of the matrix H_1 is linearly independent. Since H is a block matrix this still holds, so every set of 3 columns of H is linearly independent. This means that the distance of C is at least 4. Recall that the bottom rows of H are the r_i corresponding to the linear equations. These can only increment the number of linearly independent columns. □

5.2 Computational Complexity

We shown how to reduce a generic quadratic boolean system of equations to a standard form, containing only equations as in (5.2) and (5.3). In order to do that we introduced new auxiliary variables and generated new equations.

Summing all up we added

$$\frac{n \cdot (n-1)}{2} + n(n-2) + \frac{n \cdot (n+1)}{2} - 2 + \frac{n \cdot (n-1)}{2} - 1 \leq 4n^2$$

equations and

$$\frac{n \cdot (n-1)}{2} + n(n-2) + \frac{n \cdot (n-1)}{2} - 1 \leq 3n^2$$

new auxiliary variables. Now the actual number of quadratic equations generated, as in (5.2), is

$$\frac{n \cdot (n-1)}{2} + \frac{n \cdot (n+1)}{2} - 2 \leq 2n^2.$$

This is also the number of $H_1 \in \mathcal{M}_{7,10}(\mathbb{F}_2)$ matrices, i.e. blocks, that form the final H matrix. Therefore we can compute an upper bound for the number of columns of the matrix that is at most $10 \cdot 4n^2$. The number of rows of H depends also on the number of linear equations the system has. Recall that for each linear equation we add a row in the matrix. For the Equation (5.1) we added

$$n(n-2) + \frac{n \cdot (n-1)}{2} - 1 \leq 2n^2$$

linear equations. Hence, the number of rows in H is upper bounded by $7 \cdot 2n^2 + 2n^2 = 16 \cdot n^2$. Now assume the system is made up by m equations, the size of the final parity check matrix H is upperbounded by $40mn^2 \cdot 16mn^2$ therefore the reduction runs in time $\mathcal{O}(m^2n^4)$.

Chapter 6

Conclusions and future works

6.1 Conclusions

During our work we studied how to create an explicit polynomial-time reduction from taking an instance of the MLD problem to an instance of MQ and a reduction that does the reverse work (α and β reductions namely). Some attempts brought us to reduce MLD to an instance of SAT this into 3-SAT and then in turn into an MQ instance (reductions φ, ψ and ω namely). Trough this dissertation we provided a possible construction of such transformations, we studied a complexity bound for each and we also implemented some of them using MAGMA programming language.
According to our analysis every reduction runs in polynomial time, is therefore efficient and so the subsequent application of the reduction still remains efficient.

6.2 Future works

As a future work we plan to refine the transformations introduced as well as to study their complexities more in details. Some of the complexity bounds discussed in this dissertation are indeed bounds, which means that the actual complexity of the algorithms needs to be studied more in depth. It could be of great interest to study the resulting MQ system after an application of the transformation α on an MLD instance. As one can see the general structure of such system is pretty standard. This is due to the fact that the polynomials introduced to compute the vector weight, see section 4.2, do not depend on the solution vector x but they depend only on n. What changes instead are the m polynomials corresponding to the parity check constraint which are linear also easy to solve and the last polynomial corresponding to the weight check constraint. Hence, understanding such structure and trying to find a relatively fast way to solve them would lead to an improvement in the solving of the MLD instances and thus decoding. As a future work,

the family of codes that have been introduced in chapter 5 can be studied more in details. As we said, if en efficient decoding method can be found on this family of codes any code can be efficiently decoded. This is the same as saying that P=NP because of the NP-completeness of MLD.

Bibliography

[1] Sean Hogan. *A gentle introduction to computational complexity theory, and a little bit more.* 2011.

[2] Berlekamp Elwyn R., McEliece Robert J. , van Tilborg Henk. *On the inherent intractability of certain coding problems.* 1978.

[3] San Ling and Chaoping Xing. *Coding Theory: A First Course.* 2004.

[4] M. R. Garey and D. S. Johnson. *Computers and Intractability - A Guide to the Theory of NP-Completeness.* 1979.

[5] J. Patarin. *The Oil and Vinegar Signature Scheme.* 1997.

[6] Come Berbain, Henri Gilbert, and Jacques Patarin. *QUAD: a Practical Stream Cipher with Provable Security.* 2006.

[7] Mehdi-Laurent Akkar, Nicolas Tadeusz Courtois, Louis Goubin, and Romain Duteuil. *A Fast and Secure Implementation of SFLASH.* 2006.

[8] Jintai Ding and Dieter Schmidt. *Rainbow, a New Multivariable Polynomial Signature Scheme.* 2005.

[9] Jintai Ding and Jason E. Gower. *Inoculating multivariate schemes against differential attacks.* 2006.

[10] Stephen Cook. *The complexity of theorem proving procedures.* 1971.

[11] Leonid Levin. *Universal search problems.* 1973.

[12] Nandakishore Santhi. *Sparse Representations for Codes and the Hardness of Decoding LDPC Codes.* 2008.

[13] Bruno Buchberger. *An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal.* 1965.

Appendices

.1 SAT to 3-SAT reduction

The following code snippet reflects the reduction ψ of diagram 1. The code below itransforms a SAT CNF into a 3-SAT CNF introducing new variables and splitting the clauses exactly as by definition of ψ.

```
n := 6;
P<[x]> := BooleanPolynomialRing(n, "grevlex");
R<[y]> := PolynomialRing(P,n-3, "grevlex");
clause := [x[1],x[2],x[3],x[4],x[5],x[6]];
neg := [1,0,1,1,0,1];

sat := [[clause[1],clause[2],y[1]]];
sign := [[neg[1],neg[2],1]];
sat := sat cat [[y[i-2],clause[i],y[i-1]] : i in [3..n-2]];
sign := sign cat [[0,neg[i],1] : i in [3..n-2]];
sat := sat cat [[y[n-3],clause[n-1],clause[n]]];
sign := sign cat [[0,neg[n-1],neg[n]]];

sat;
sign;
```

- **Inputs** This piece of code takes as input `n`, a SAT clause which is denoted by `clause` and an array of negations `neg`. For example if a value of the `neg` array is 0 the literals in the same position in `sat` is negated.
- **Outputs** This prints out an array of 3-SAT clauses along with an array of their "negativities" which basically works as above.

.2 3-SAT to MQ reduction

The following code is the implementation of the ω reduction defined in chapter 3. It transforms a 3-SAT CNF formula into a system of polynomials. Some of these polynomials have degree three. This code does not reduce such polynomials to a set of quadratics.

```
n := 4;
P<[x]> := BooleanPolynomialRing(n, "grevlex");
sat := [[P.1], [P.2,P.3],[P.2,P.3,P.4]];
neg := [[1],[1,0],[0,0,1]];
for i in [1..#sat] do
    temp := 1;
    for j in [1..#sat[i]] do
        temp := temp * (sat[i][j] + neg[i][j]);
    end for;
    temp;
end for;
```

- **Inputs** The input to this code snippet are the values `n`, and the two sets `sat` and `neg` which are the CNF formula and the negation of the literals namely.
- **Outputs** The output is the set of boolean equations corresponding to each clause in the CNF formula.

.3 Parity Check Constraint

The following code is the implementation of the parity check constraint described in section 2.1. It creates a system of linear equations that reflects the constriant $Hv^\top = s$. This is part of the reduction α.

```
n := 4;
m := 2;
V := VectorSpace(GF(2),n);
H := [V![1,0,1,0],V![1,1,0,1]];
s := [0,1];
S := [];
P<[x]> := BooleanPolynomialRing(n,"grevlex");

for j in [1..m] do
    e := 0;
    for i in [1..n] do
        if H[j][i] eq 1 then
           e := e + x[i];
        end if;
    end for;
    e := e + s[j];
    S := Append(S,e);
end for;
S;
```

- **Inputs** Here the code takes as input the values `H` which represents the parity check matrix, `m,n` which are the number of rows and columns of `H` respectively and finally the value of the syndrome `s`.
- **Outputs** This piece of code prints out the set of m linear equations corresponding to the constraint $Hv^\top 0s$.

.4 Weight Computation

The code below is the implementation of the weight computation part depicted in section 2.2 and particularly 4.2. The polynomials produced by this code are those listed in (4.2).

```
l := 2;
n := 4;
P<[x]> := BooleanPolynomialRing(n,"grevlex");
A<[a]> := PolynomialRing(P, l*n, "grevlex");

S := [a[1], a[2]];
a[1];
a[2];
for i in [1..n-1] do
    for j in [1..l] do
        p := 1;
        for k in [1..j-1] do
            p := p * a[(i-1)*2 + k];
        end for;
        p := p*x[i];
        p := p + a[2*(i-1) + j];
        p := p + a[2*i + j];
        p;
    end for;
end for;
p := 1;
for j in [1..l] do
    p := p * a[2*(n-1) + j];
end for;
p := p * x[n];
p;
```

- **Inputs** The only two input values here are the value of `n` and its base 2 logarithm `l`.
- **Outputs** This code snippet prints out every single polynomial needed to represent the counting registers in (4.2) and furthermore the polynomial coresponding to Equation (2.6).

.5 Weight Check

In the following piece of code we implement the computation of the weight check polynomial introduced in Equation (4.5). This is part of the α reduction of the diagram 1.

```
n := 4;
l := 2;
t := [0,1];
A<[a]> := PolynomialRing(P, l*n, "grevlex");
f := [];
for i in [1..l] do
    temp := 1;
    for k in [i+1..l] do
        temp := temp*(a[2*(n-1) + k] + t[k] + 1);
    end for;
    f := Append(f, temp*(a[2*(n-1)+i] + t[i]));
end for;
F := 0;
for i in [1..l] do
    F := F + f[i]*(t[i] + 1);
end for;

S := Append(S,F);
S;
```

- **Inputs** In this code snippet the only input value is the base 2 logarithm of n. This value is the number of bits needed to encode in binary basis the weight of a vector in $\mathbb{F}_2^n$.
- **Outputs** The output `F` is the polynomial corresponding to the weight check equation in (4.5). The weight constraint is respected if and only if it evaluates to 0.

.6 Full reduction MLD to MQ

In the following code we put together the code snippets in appendices 3,4 and 5 to obtain a full and working implementation of the α reduction.

```
n := 4;
m := 2;
l := 2;
V := VectorSpace(GF(2),n);
Q := [V![1,0,1,0],V![1,1,0,1]];
s := [0,1];
S := [];
t := [1,0];
P<[x]> := BooleanPolynomialRing(n,"grevlex");
A<[a]> := PolynomialRing(P, l*n, "grevlex");

for j in [1..m] do
    e := 0;
    for i in [1..n] do
        if Q[j][i] eq 1 then
           e := e + x[i];
        end if;
    end for;
    e := e + s[j];
    S := Append(S,e);
end for;

S := S cat [a[1], a[2]];
for i in [1..n-1] do
    for j in [1..l] do
        p := 1;
        for k in [1..j-1] do
            p := p * a[(i-1)*2 + k];
        end for;
        p := p*x[i];
        p := p + a[2*(i-1) + j];
        p := p + a[2*i + j];
        S := Append(S,p);
    end for;
end for;
p := 1;
for j in [1..l] do
    p := p * a[2*(n-1) + j];
end for;
p := p * x[n];
S := Append(S,p);

f := [];
for i in [1..l] do
    temp := 1;
    for k in [i+1..l] do
        temp := temp*(a[2*(n-1) + k] + t[k] + 1);
    end for;
    f := Append(f, temp*(a[2*(n-1)+i] + t[i]));
```

```
end for;
F := 0;
for i in [1..l] do
    F := F + f[i]*(t[i] + 1);
end for;

S := Append(S,F);
S;
```

- **Inputs** The inputs needed in this piece of code are a matrix represented here by `Q` which is a set of `m` vectors of length `n`. The parameters `n,m` need to be given as well as inputs. Furthermore also `l` which is the basis 2 logarithm of n has to b especified. Finally the values of the syndrome `s` and the binary representation of the weight constraint `t`.
- **Outputs** The output `S` is the complete system of eqations, even though not quadratic, that represent the instance of MQ corresponding to the initial MLD instance.