UNIVERSITÄT PASSAU
*Fakultät für Informatik und Mathematik*

**Dissertation**

# New Techniques for Polynomial System Solving

**Ehsan Ullah**

Eingereicht an der Fakultät für Informatik und Mathematik
der Universität Passau als Dissertation zur Erlangung des
Grades eines Doktors der Naturwissenschaften

---

Submitted to the Department of Informatics and Mathematics
of the Universität Passau in Partial Fulfilment of the Requirements
for the Degree of a Doctor in the Domain of Science

Betreuer / Advisor:
**Prof. Dr. Martin Kreuzer**
Universität Passau

Februar 2012

# New Techniques for Polynomial System Solving

**Ehsan Ullah**

Erstgutachter: **Prof. Dr. Martin Kreuzer**
Zweitgutachter: **Prof. Dr. Lorenzo Robbiano**
Mündliche Prüfer: **Prof. Dr. Franz Brandenburg**
**Prof. Dr. Tobias Kaiser**

Der Fakultät für Informatik und Mathematik
der Universität Passau
vorgelegt im Februar 2012

*For my family, who supported me all the way since the beginning of my studies*
*and*
*for those, who always offered me unconditional love, support and prayers.*

# Acknowledgement

It is a pleasure to thank those who have helped and encouraged me throughout the long and difficult process of completing this thesis. First and foremost I am heartily thankful to my supervisor, Prof. Dr. Martin Kreuzer, whose very good supervision, encouragement, guidance and support from the initial to the final level enabled me to develop an understanding of the subject. I appreciate all his contributions of time, ideas, and suggestions to make my Ph.D. experience productive and stimulating. He is the one who can be blamed for creating a wonderful working environment in our research group of Symbolic Computations.

It is an honor for me to have a very experienced professor, Prof. Dr. Lorenzo Robbiano, as a second reviewer of this thesis. I appreciate his contribution of time, ideas, and suggestions. I am thankful to Prof. Dr. Schwartz for giving me a chance for completing my Ph.D. at Faculty of Informatics and Mathematics of the Universität Passau. I would like to thank Prof. Dr. Franz Brandenburg and Prof. Dr. Tobias Kaiser for extending my knowledge about complexity theory and commutative algebra. It is an honor for me to be a doctoral student in the research group of Symbolic Computation and obtain a Ph.D. degree from Universität Passau.

The generous support from Higher Education Commission (HEC) of Pakistan is greatly appreciated. Without their support, my ambition to study abroad can hardly be realized. Besides, I would like to acknowledge Deutsche Akademische Austausch Dienst (DAAD) for organizing my research program in Germany. I gratefully thank Prof. Dr. Tobias Kaiser, Prof. Dr. Gerhard Rosenberger and Dr. Andreas Dolzmann for recommending my applications for the further financial support. I am also very thankful to IBM and AMPL teams for providing their commercial softwares CPLEX and AMPL free of charge for academic research.

I am indebted to all my colleagues who have shared their expertise with me. In

particular, I would like to thank Stefan Kasper, Jan Limbek, Xingqiang Xiu and Stefan Schuster for their advise and fruitful discussions during the development of the packages for ApCoCoA and proof reading this thesis.

Lastly, I would like to thank my whole family for all their prayers, love, encouragement and supporting me spiritually. I owe my deepest gratitude to my parents who raised me with a love of knowledge and supported me in all my pursuits. I have no suitable word that can fully describe their everlasting love to me.

Finally, I offer my regards and blessings to all of those who supported me in any respect during the completion of my studies.

**Ehsan Ullah**

February 2012,
Universität Passau, Germany

# Contents

# List of Tables

# Abbreviations

| | |
|---|---|
| CTC | Courtois Toy Cipher |
| HFE | Hidden Fields Equations |
| NulLA | Nullstellensatz Linear Algebra |
| LA | Linear Algebra |
| EF | Echelon Form |
| SEF | Sparse Echelon Form |
| SGE | Structured Gaußian Elimination |
| MLA | Mutant Linear Algebra |
| MLA2 | Improved Mutant Linear Algebra |
| BBA | Border Basis Algorithm |
| MBBA | Mutant Border Basis Algorithm |
| MBBA2 | Improved Mutant Border Basis Algorithm |
| IP | Integer Programming |
| MILP | Mixed Integer Linear Programming |
| MINLP | Mixed Integer Non-Linear Programming |
| IPC | Integer Polynomial Conversion |
| RPC | Real Polynomial Conversion |
| LPC | Logical Polynomial Conversion |
| CNF | Conjunctive Normal Form |
| SS | Standard Strategy |
| LPS | Linear Partner Strategy |
| DPS | Double Partner Strategy |

| | |
|---|---|
| QPS | Quadratic Partner Strategy |
| CPS | Cubic Partner Strategy |
| HIC | Hybrid Integer Conversion |
| HRC | Hybrid Real Conversion |
| SNF | Smith Normal Form |
| RSC | Real Standard Conversion |
| RFC | Real Fourier Conversion |

# Chapter 1

# Introduction

Solving systems of polynomial equations over finite fields is one of the most important research problems which gives rise to applications in many areas such as cryptography, coding theory, robotics, computational geometry, etc. For instance, the intractability of solving this problem assesses the security of a type of public-key cryptosystems called multivariate algebraic cryptosystems [69]. Multivariate algebraic cryptosystems are believed to be secure against attacks with quantum computers, thus giving rise to one of the candidates for post-quantum cryptography [25]. Moreover, the security of asymmetric as well as symmetric cryptosystems is connected to the problem of solving systems of polynomial equations over finite fields. This was firstly noticed by Claude Shannon who remarked in his seminal paper [160]:

> *Thus, if we could show that solving a certain system requires at least as much work as solving a system of simultaneous equations in a large number of unknowns, of a complex type, then we would have a lower bound of sorts for the work characteristic.*

It is well-known that any encryption map between finite dimensional vector spaces over a finite field is polynomial. Thus, it is natural to represent the task of breaking a cryptosystem by the problem of solving a multivariate polynomial system of equations over a finite field. This type of attacks is known as *algebraic attacks* and is studied in *algebraic cryptanalysis*. It has been shown that the main task for carrying out a successful algebraic attack on a cipher (or for examining the security of a cipher) is to solve a multivariate polynomial system over a finite field. Therefore, in this thesis we study new techniques that can be used in the context of polynomial systems derived from algebraic attacks to examine the security of different ciphers.

In particular, the topic of this thesis is to solve the following well-known problem. Let $p$ be a prime number, let $q = p^e$ for some $e > 0$, let $K = \mathbb{F}_q$ be the finite field with $q$ elements, and let $f_1, \ldots, f_\ell \in K[x_1, \ldots, x_n]$ be non-zero polynomials. Find the $K$-rational solutions of the system of polynomial equations:

$$\begin{aligned} f_1(x_1, \ldots, x_n) &= 0 \\ &\vdots \\ f_\ell(x_1, \ldots, x_n) &= 0 \end{aligned}$$

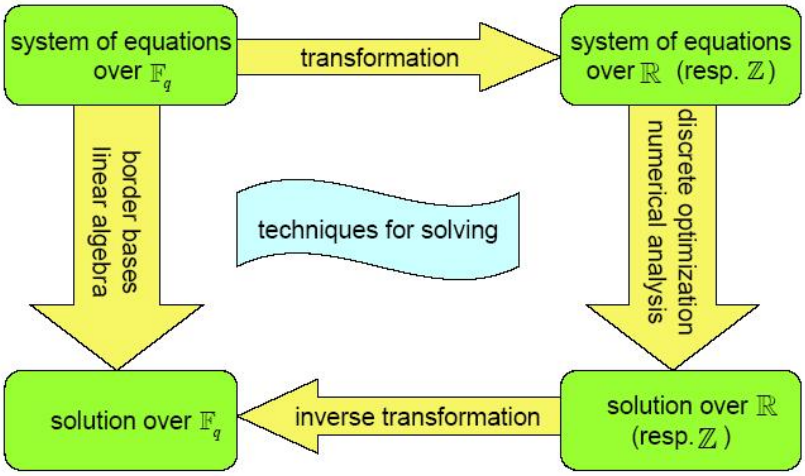
There are several algorithms which solve systems of multivariate polynomial equations over finite fields. Our work focuses mainly on two aspects. Firstly, we investigate linear algebra techniques for polynomial system solving. The reason for choosing linear algebra as a solving tool is that the most efficient and effective algorithms for polynomial system solving, by computing a Gröbner basis, use linear algebra. Furthermore, we can benefit from the full potential of the linear algebra techniques developed in the last fifty years. This motivates further investigations in the field of linear algebra techniques for polynomial system solving. Our intention is to look for new linear algebra techniques and to study the impact of various strategies for improving these techniques. In particular, we study techniques coming from combinatorial optimization and border basis theory. Furthermore, we develop variants of these techniques which look for new low degree polynomials in the ideal generated by the original polynomials.

Secondly, we focus on using highly developed techniques from several other areas such as discrete optimization and numerical analysis. It is well-known that the problem of solving a system of multivariate polynomial equations, even over a finite field, is **NP**-hard. On the other hand, the mixed integer programming problem (a problem from discrete optimization), solving a system of polynomial equations numerically (a numerical analysis problem), and solving a system of linear Diophantine equations for non-negative integer solutions (a number theory problem) are also **NP**-hard problems. Inspired by the possibility that solution of either one of them could be used for solving the others, since all **NP**-complete problems are polynomially equivalent, we began this investigation.

The reasons for choosing discrete optimization, numerical analysis and linear Diophantine system solving as the basis of solving tools are as follows. Research efforts of the past fifty years have led to the development of discrete optimization and numerical analysis as mature disciplines of applied mathematics. After formulating the solution of a system of polynomial equations as a discrete optimization problem or a numerical

analysis problem, we can apply standard IP and numerical solvers inside our algebraic techniques. In this way we can get the full advantage of several algorithms available in highly developed mature disciplines of applied mathematics for solving our main problem of finding $K$-rational solutions for polynomial systems.

The solution process can be separated into two steps. Firstly, we apply a conversion algorithm (and later the inverse conversion) at formulation level. Secondly, we employ a solver for the solution of the reformulated problem. Our main work focuses on the first step. We developed conversion algorithms which suggest that a clever formulation may accelerate the performance of a solver dramatically, especially by exploiting the structural properties of the system. The following figure illustrates our approach.




As we have seen above, we have techniques from different disciplines at our disposal for solving systems of polynomial equations. One advantage of using these techniques is that they will be automatically improved with developments in their respective disciplines. We have implemented all these techniques in C++ and CoCoAL. A part of these implementations is also available online in packages `CharP` (see Appendix B) and `glpk` (see Appendix C) of the computer algebra system ApCoCoA [12]. To study techniques using numerical analysis, we have developed `Bertini` [21] and `HOM4PS` [127] interfaces to ApCoCoA (see Appendix A). Finally, the efficiency of the developed techniques is examined using standard cryptographic examples such as Small Scale AES, CTC and HFE. Our experimental results show that all solving techniques we present are highly competitive to state-of-the-art algebraic techniques. In the following we consider these disciplines one by one and describe previous work as well as our contribution and proposals for future research. This thesis can be divided into the following four main parts.

### Techniques From Linear Algebra

One of the most useful applications of Gröbner bases is to compute the solution set of a system of polynomial equations. Buchberger's Algorithm [34] was the first algorithm for computing Gröbner bases. Due to complexity issues of the standard Buchberger algorithm, several variants of this algorithm such as F4 [73], F5 [74] and XL (extended linearization) [59] have been proposed. Furthermore, several optimized versions of these variants make them even more powerful. Actually, these variants reduce a polynomial system solving problem to a linear algebra problem. The success achieved by these algorithms motivates further investigations in the field of linear algebra techniques for polynomial system solving.

Unfortunately, Gröbner bases are not always well suited for solving systems of polynomial equations. Border bases are a natural generalization of Gröbner bases that are known to deform smoothly with the input and provide a more flexible concept than Gröbner bases (see [117]). One of the most useful applications of border bases is to solve zero-dimensional systems of polynomial equations (see, e.g., [13, 139, 145, 117]). The preference of border bases over Gröbner bases partly arises from the iterative generation of linear syzygies, inherent in the Border Basis Algorithm, which allows for successively approximating the basis degree by degree (see [111]). Moreover, the Border Basis Algorithm is a linear algebra algorithm.

We focus mainly on linear algebra algorithms which use the multivariate polynomials to "enlarge" the system by generating additional equations having the same set of solutions. The enlarged system can be thought of as a *large* system of linear equations. Using linear algebra techniques, such as Gaussian elimination, on the matrix representation of this linear system, a solution can be obtained. For this purpose we study some techniques from combinatorial optimization, algorithms for computing border basis and J. Ding's [67] concept of mutants.

In [67], J. Ding observed that during the linear algebra step (Gaußian elimination) some *special* polynomials of degree lower than expected appear and he called them *mutants*. The *mutant strategy* aims to distinguish mutants from the other polynomials and give them priority in the process of generating new equations. Later on, the mutant strategy was further optimized and become the *improved mutant strategy* in [142]. The improved mutant strategy is based on the mutant concept and a new enlargement method called *partial enlargement strategy*. It introduces a heuristic strategy of only choosing the minimum number of mutants, which is called *necessary mutants*. The mutant strategy, along with its improvements, can be used to improve various algorithms

for solving systems of polynomial equations which use linear algebra.

In [68, 142, 140, 143], J. Ding et al. proposed the MutantXL, the MXL2 and MXL3 algorithms as variants of the XL Algorithm which are based on the mutant strategy. Note that the XL Algorithm is the first algorithm that is equipped with the mutant strategy. Therefore, there is a natural need to develop mutant variants of other linear algebra algorithms such as the Border Basis Algorithm for polynomial system solving.

In combinatorial optimization, systems of polynomial equations have been used to model combinatorial problems. This well-known method, which Alon referred to as "the polynomial method" (see [7, 8]) recently regained strong interest. In [61, 63, 64] infeasibility of certain combinatorial problems is established using Hilbert's (complex) Nullstellensatz and the authors provide an algorithm `NulLA` to establish infeasibility by using a linear relaxation. Furthermore, in [64] J.A. De Loera et al. reviewed a methodology to solve systems of polynomial equations and inequalities. They discussed techniques that use the algebra of multivariate polynomials with coefficients over a field to create large-scale linear algebra or semidefinite programming relaxations of many kinds of feasibility or optimization questions.

Our contribution in developing linear algebra techniques consists mainly on the following two points.

**Techniques From Combinatorial Optimization:** We study some techniques from combinatorial optimization to transform a polynomial system solving problem into a (sparse) linear algebra problem. In particular, we study the concept of transforming infeasibility proofs to large systems of linear equations and extend the ideas of J.A. De Loera et al. [64] to develop an algorithm called the Linear Algebra (LA) Algorithm aimed at solving systems of polynomial equations over finite fields.

**Optimizations Using Mutants:** We use J. Ding's concept of mutants [68] to optimize the LA Algorithm and the Border Basis Algorithm [111], in terms of memory and time consumption. We believe that the mutant variants of the LA Algorithm and especially the Border Basis Algorithm are highly competitive to state-of-the-art algorithms for solving systems of polynomials equations.

The Linear Algebra (LA) Algorithm is a *specialization* of prior algorithms from combinatorial optimization used by J.A. De Loera et al. [61, 63, 64] and based on fast large-scale (sparse) linear algebra computations over a finite field. Although such techniques have been known in combinatorial optimization, they have not been used

widely for polynomial system solving over finite fields. The key issue that we investigate here is the use of the so-called "polynomial method" for solving over finite fields. We are particularly interested in whether this can be accomplished in practice for large systems of polynomial equations over finite fields. We explicitly formulate and explain the LA Algorithm. The calculations reduce to (sparse) matrix manipulations, mostly rank computations. It turns out that such techniques are particularly effective when the number of solutions is finite, when the under lying field is a finite field, or when the system is well structured.

The main drawback of the LA Algorithm is that it can solve only systems having a unique solution. In order to solve systems with a finite number of solutions, we study another linear algebra algorithm, namely the Border Basis Algorithm (BBA) [111]. The core of the LA algorithm is identical to the $L$-stable span (or the $U$-stable span) procedure used in the BBA, which intimately links both algorithms. The difference is of a technical nature: whereas the LA Algorithm establishes infeasibility, the classical BBA, as presented in [111], computes the actual border bases of the ideal. The most time consuming part of the BBA is computing a stable span. Moreover, the complexity of the BBA relies on this step.

Although the LA Algorithm and the BBA can be used for solving systems of polynomial equations over finite fields [59, 58, 73, 74, 117, 150], theoretical complexity estimates have shown that this kind of algorithms is infeasible for many realistic applications. This is due to the fact that, in many practical cases, the computations made by these algorithms lead to constructing a huge system of polynomial equations, and consequently a huge matrix, which requires a lot of time and memory resources.

A big challenge is to improve these algorithms in a way which uses only limited available memory and time resources for solving a multivariate polynomial system with as large number of equations and variables as possible. Actually, these algorithms find additional polynomials of not much larger degree in the ideal generated by the polynomials of the system by multiplying them by terms. They apply linear algebra (Gaußian elimination) after linearizing the system. One of the strategies to improve the efficiency of these algorithms is to find better linear algebra techniques. This mainly reduces the time consumption. On the other hand, strategies improving the enlargement step of the polynomial system, by reducing the matrix size, will affect both time and memory consumption.

We improve the enlargement step of the LA Algorithm and the BBA. This leads us developing two kinds of hybrid techniques. The first kind of hybrid techniques combines

ideas studied by De Loera et al. [61, 64] for transforming combinatorial infeasibility proofs to large systems of linear equations and ideas of J. Ding et al. [67, 140, 142, 143] involving the concept of mutants. The second kind of hybrid techniques uses the concept of mutants to optimize the BBA for solving systems of polynomial equations over finite fields.

We modify the LA algorithm such that, instead of enlarging the system blindly and increasing the degree, we first use the mutants, if any, at the lowest possible degree to enlarge the system. We call this new algorithm the Mutant Linear Algebra (MLA) Algorithm. Furthermore, we modify the LA Algorithm using improved mutant strategy. This results in solving systems with fewer number of enlarged polynomials than the MLA Algorithm. We called this new algorithm the Improved Mutant LA (MLA2) Algorithm.

We also use the mutant strategies to improve the BBA. In particular, we explicitly explain a way to compute a stable span using the mutant strategy and the improved mutant strategy. We call the versions of the BBA using the mutant strategy and the improved mutant strategy respectively the MBBA and the MBBA2. Our experimental results show that the mutant variants of the LA Algorithm and the BBA can indeed outperform their original versions and can solve multivariate systems at a relatively lower degree. They provide improvements in terms of time and memory consumption.

The MXL3 is an algorithm for computing a Gröbner basis in order to solve systems with finite number of solutions. It uses mutant strategies in the setting of the XL Algorithm for computing a Gröbner basis. The results in [143] show that in both classical cryptographic challenges as well as randomly generated polynomial systems, MXL3 performs better than the Magma's implementation of F4 in terms of memory and time consumption. Since the preference of border bases over Gröbner bases partly arises from the iterative generation of linear syzygies, inherent in the BBA, which allows for successively approximating the basis degree by degree, we believe that the MBBA2 can be at least as good as the MXL3. The linear algebra step of the LA Algorithm and the computation of a stable span can be found with the help of all the sparse linear algebra techniques. In this sense, the mutant variants of the BBA could outperform the best known solution algorithms for polynomial systems and deserve further efficient implementation and experimentation. Furthermore, the flexible partial enlargement strategy introduced in [36] can further improve the MBBA2. Therefore, in the spirit of the developed techniques, it is obviously possible to launch research projects for further investigation, experimentation, and to benchmark the implementation.

### Techniques Using Discrete Optimization

From now on, we restrict our attention to solving polynomial systems defined over $\mathbb{F}_2$. Although the generalization to other finite base fields is straightforward, we want to concentrate on the fundamental principles in the most important case. We address techniques using *Mixed Integer Linear Programming (MILP)* and *Mixed Integer Nonlinear Programming (MINLP)*. After formulating the solution of a system of polynomial equations as an integer programming problem, we can apply standard IP solvers inside our algebraic techniques. In this sense, the process of solving consists of the following two steps.

- Applying a conversion algorithm for transferring (formulating) the problem of solving a system of polynomial equations into a MILP or MINLP problem.
- Using an IP solver to solve the transformed (formulated) MILP or MINLP problem.

A conversion algorithm can be further separated into the following two steps.

- **Transformation to $\mathbb{R}$ or $\mathbb{Z}$:** Transform a system of polynomial equations over $\mathbb{F}_2$ into a system of polynomial equations over $\mathbb{R}$ (resp. $\mathbb{Z}$).
- **Modeling a MILP (or MINLP) Problem:** Model the transformed system as a MILP or MINLP problem.

Some methods for representing polynomials over $\mathbb{F}_2$ as polynomials over $\mathbb{R}$ can be found in the literature, but they have not been used for our purpose. In [24], an overview of possible representations is listed. Later, this study was extended slightly in [124], but the main idea behind the representation methods was basically unaltered. Very recently, some methods have been proposed for transferring the problem of solving a system of polynomial equations over $\mathbb{F}_2$ into a MILP problem. In [117], M. Kreuzer provided a conversion algorithm based on converting polynomial equations over $\mathbb{F}_2$ into polynomial equations over $\mathbb{Z}$. In [31], J. Borghoff et al. provided another conversion method based on converting polynomial equations over $\mathbb{F}_2$ into polynomial equations over $\mathbb{R}$. J. Borghoff et al. studied their method for systems of polynomial equations coming from *Bivium Cipher*, but an algorithm for general systems of polynomial equations is missing.

Recently, solving a system of polynomial equations over $\mathbb{F}_2$ by converting it to a set of propositional logic clauses achieved a lot of success. The first study of efficient

methods for converting boolean polynomial systems to CNF clauses was presented in [18]. Later this study was extended slightly in [19, 48] and [166] but the procedure was basically unaltered. The latest effort is due to P. Jovanovic and M. Kreuzer [106]. They examined different conversion strategies, i.e. different ways to convert the polynomial system into a satisfiability problem.

Our contribution in this thesis focuses on developing and improving the conversion algorithms. In the following we elaborate our contribution in detail. First of all, we review the algorithm provided by M. Kreuzer in [117] and the suggestions by J. Borghoff et al. in [31]. We provide an algorithm for solving general systems of polynomial equations over $\mathbb{F}_2$ according to the suggestions of J. Borghoff et al. [31]. Furthermore, we compare both methods with the help of experimental results and suggest the settings (while modeling a MILP problem) for the best performance of these algorithms.

As explained above, a conversion algorithm consists of two steps, namely transformation to a set of equations over $\mathbb{R}$ or $\mathbb{Z}$ and modeling a MILP problem. While modeling a MILP problem we need to replace certain nonlinear terms with new 0-1 variables. The idea is to introduce new 0-1 variables to take the place of the nonlinear terms, simultaneously introducing auxiliary constraints to insure that the new variables will assume the appropriate values. A big challenge is to improve the above conversion algorithms such that they provide better MILP models by replacing nonlinear terms with new 0-1 variables in a more economical way.

To develop several strategies for modeling a MILP problem, we use the so called *transformed linear approach* (see [17, 171, 175]) which involves some standard procedures for linearizing nonlinear 0-1 polynomial functions into linear 0-1 polynomials. We study these standard approaches to achieve more economical constraints while replacing nonlinear terms with new 0-1 variables. Thus the purpose is to give procedures for achieving improved linear representations of nonlinearities occurring in the above described conversion algorithms. The experimental results show that the conversion algorithms equipped with new developed strategies perform better than their standard versions in many cases.

Next we present a new conversion method based on propositional logic and pseudo-boolean optimization. In particular, first we review some concepts from propositional logic and then exploit the connection between propositional clauses and 0-1 inequalities to model the polynomial system over $\mathbb{F}_2$ as a MILP problem. This enables us to export several strategies from propositional logic for modeling a MILP problem. Therefore, the new conversion method also has the ability to exploit several strategies

for formulating more economical MILP models. The experiments show that our new polynomial conversion technique is at least as good as the previously known techniques and provides better results in most cases.

The connection between propositional clauses and linear 0-1 inequalities not only provides a new conversion method, but also provides strategies to optimize the methods proposed by M. Kreuzer [117] and J. Borghoff et al. [31]. As described above such strategies can be used to achieve more economical constraints while replacing nonlinear terms with new 0-1 variables. This leads us towards the development of new hybrid conversion methods which seem to outperform their original versions proposed by M. Kreuzer [117] and J. Borghoff et al. [31].

Finally, to conclude our discussion of techniques using MILP, we present a comparison of all techniques. We present experimental results which show that our newly developed techniques and strategies result in a substantial speed up of IP solvers. In extreme cases the gain resulting from our techniques and strategies can be striking. Furthermore, we note that some IP solvers like CPLEX can be parallelized. Thus we can benefit from parallelization capabilities of IP solvers to solve systems of polynomial equations. In this sense we have developed a number of techniques and strategies for solving systems of polynomial equations that can be parallelized. In addition to this, internal parameters of CPLEX can be fine-tuned to reach the optimum in different way.

We also highlight a new technique, using non-convex MINLP, for solving systems of polynomial equations which was never used before. This technique should mark a first step and offers several future research directions. Based on the above-mentioned conversion algorithms, we develop two approaches for transferring the problem of solving a system of polynomial equations over $\mathbb{F}_2$ into a non-convex MINLP problem. The first approach, which is based on converting polynomial equations over $\mathbb{F}_2$ into polynomial equations over $\mathbb{R}$, does not seem to be effective. But the second approach, which is based on converting polynomial equations over $\mathbb{F}_2$ into polynomial equations over $\mathbb{Z}$, seems to be rather efficient and deserves to be the subject of further investigations. Moreover, we also generalize this approach to an arbitrary finite field. Using concrete examples, we illustrate the performance of these techniques. Furthermore, we suggest settings (while modeling a non-convex MINLP problem) for the optimal performance of these techniques.

In the spirit of the techniques studied above, it is obviously possible to generate a number of further variations of the conversion algorithms which have the potential to

speed up IP solvers. We have realized that there is a strong need to consult literature available on transformation of 0-1 programs into 0-1 linear programs to make the conversion methods more effective and take advantage of the full potential of fifty years research on MILP. Thus the conversion methods deserve further investigation and experimentation. Furthermore, MILP and MINLP are very fast developing disciplines of mathematics and the conversion methods we present can take full advantage of any new development.

### Techniques Using Linear Diophantine System Solving

This part of our work presents a new technique, based on linear Diophantine system solving, for solving systems of polynomial equations over $\mathbb{F}_2$. While studying techniques using discrete optimization, we have seen different conversion techniques for formulating the problem of solving a system of polynomial equations over $\mathbb{F}_2$ into a system of linear equalities and inequalities over $\mathbb{Z}$. This formulation suggests to apply a linear Diophantine system solving algorithm for finding a non-negative integer solution satisfying the system of linear equalities and inequalities.

After formulating the task, we apply the straightforward approach for solving systems of linear Diophantine equations for non-negative integer solutions. First we find a general integer solution using the Smith normal form. Then we obtain a (unique) minimal non-negative integer solution from general integer solution using MILP. We illustrate the performance of this technique using some concrete examples. Although solving systems of linear Diophantine equations for non-negative integer solutions is an **NP**-hard problem, our experimental results show that this technique seems to be rather efficient and deserves to be the subject of further investigations. We believe that the latest developments for solving systems of linear Diophantine equations such as the ones in [83, 71] can perform even better. Furthermore, we highlight some ideas to spark further research in this direction. Finally, we remark that the resulting linear Diophantine systems are highly sparse and motivate developing sparse linear Diophantine system solving algorithms.

### Techniques Using Numerical Analysis

Next we address approaches based on numerical methods for solving systems of polynomial equations over $\mathbb{F}_2$. Since numerical methods operate on the set of real numbers, we first convert the system over $\mathbb{F}_2$ into a system over $\mathbb{R}$ using different conversion techniques and then we apply a numerical solver. We develop conversion techniques

such that we can use numerical solvers, for instance, homotopy continuation methods and variants of Newton's method.

Recently, in [124] Lamberger et al. used the ideas mentioned in [24] for representing a system of polynomial equations over $\mathbb{F}_2$ as a system of polynomial equations over $\mathbb{R}$. Then they apply two numerical algorithms: the DIRECT algorithm by D.R. Jones et al. [105] and interior-reflective Newton method by Coleman and Li [52, 53]. They used these two algorithms to attack a reduced version of the stream cipher *Trivium* called *Bivium A*. Their experimental results show that the DIRECT algorithm does not yield any success, whereas the interior-reflective Newton method needs 75% of the original solution for choosing a good starting point. Therefore, both methods do not really seem to be practical. But the authors believe that we can do better if we use the available knowledge in the field of numerical analysis.

Unfortunately, the known conversion techniques result in an increase in the size of the system over $\mathbb{R}$ in terms of the number of equations, the number of terms, and the number of variables. Therefore, one of the strategies to improve solving using numerical methods is to find better conversion algorithms. Another strategy is to search for a suitable numerical solver. Several algorithms such as variants of Newton's method and homotopy continuation methods are available as numerical solvers which have not been used for our purpose. Finally, the performance of a numerical solver may vary depending on the polynomial system to be solved.

In the spirit of the above observations, we realize that there is a strong need to launch further investigations in this direction. First we study suggestions by Lamberger et al. [124]. Then we investigate the combination of several conversion techniques and numerical solvers for solving systems of polynomial equations. Finally, we generalize the approach in [24, 124] and extend this work further. In particular, we present a proposal for choosing starting points and some ad-hoc tricks to obtain better results.

As for numerical solvers, we investigate the use of homotopy continuation methods and variants of Newton's method. The reason for choosing homotopy continuation methods is their powerful feature of path-following which achieved a lot of success recently. Furthermore, parallel capabilities of continuation methods make them even more powerful. Newton's method and its variants are well-known tools in the numerical analysis community for approximating real solutions of systems of polynomial equations. Since we may assume that our special system of polynomial equations has a unique real solution or only few real solutions, we may hope for the convergence of Newton's method to a real solution, when using path-following techniques, we may

hope to get a real solution by tracking only one or a few paths.

We present experimental results which show that our new refinements of the techniques suggested by Lamberger et al. [124] result in a substantial better performance of Newton methods. For instance, according to our new starting point selection proposal, the interior-reflective Newton method needs 50% of the original solution for choosing a good starting point, whereas Lamberger et al. [124] suggested to choose 75% of the original solution. Starting points has a high influence on the convergence of Newton methods, especially if the dimension of the system is high. Since we have used Newton methods which are locally convergent and locally convergent methods require a good starting point, we are not able to solve large systems, i.e. systems involving many equations and indeterminates without providing much information as a part of starting point.

Homotopy continuation methods do not need any initial starting point but they are computationally infeasible for large systems. Actually, to find real solutions, homotopy continuation offers only the option of finding all roots, real and complex, and then casting out the complex ones. Unfortunately, there is no other way for finding all real solutions directly. Furthermore, the theory underlying homotopy continuation depends on working over the complex projective spaces and the real solutions of the start system may lead to complex solutions of the target system and vice versa. One more hurdle that we need to face is that homotopy continuation deals only square systems. In our case, the so-called process of randomization for obtaining a square system from an overdetermined system makes the system even more difficult for solving.

All numerical techniques investigated provide more or less the same results. The methods examined in this part of our work were not able to produce a powerful technique. However, they will be certainly help to get a better understanding of using numerical methods for solving systems over finite fields.

### Organization of the Thesis

This section presents an outline of the thesis. Throughout this thesis we follow the notation and terminology introduced in the books [120, 121] unless mentioned otherwise. This thesis consists of seven chapters and four appendices. Chapter 1, the present chapter, consists of the introduction and an outline of the thesis.

In Chapter 2 we introduce some basic concepts and notation useful for the remainder of this thesis. In particular, we recall the mathematical tools necessary for understanding polynomial system solving over finite fields. The readers familiar with

this topic can skip this chapter and continue with Chapter 3. We start by introducing finite fields in Section 2.1. After having a short look on their existence and uniqueness properties we recall the representation of elements in finite fields. In Section 2.2 we move on to the problem of solving a system of multivariate polynomial equations over finite fields. The main task for a successful algebraic attack on a cipher (or for examining the security of a cipher) is to solve a multivariate polynomial system over a finite field. This is the topic in Section 2.3 which addresses applications of polynomial system solving over finite fields in cryptography and cryptanalysis.

After Chapter 2, the reader should be sufficiently warmed up to enter the hunt for new techniques for polynomial system solving over finite fields. Our journey through the land of new techniques starts in Chapter 3 which addresses linear algebra techniques to solve polynomial systems having a unique $K$-rational solution. The concept of transforming infeasibility proofs to large systems of linear equations, recently studied by De Loera et al. [61, 64] to resolve the combinatorial feasibility problem, is reviewed in Section 3.1. In particular, we recall an algorithm aimed at proving combinatorial infeasibility based on the observed low degree of Hilbert's Nullstellensatz certificates for polynomial systems arising in combinatorics, and based on fast large-scale linear algebra computations over a finite field. Based on the ideas reviewed in Section 3.1, we explicitly formulate and explain the Linear Algebra (LA) Algorithm, which is an algorithm for solving systems of polynomial equations over finite fields, in Section 3.2. Section 3.3 reports on experiments with the LA Algorithm using different linear algebra libraries and some self implemented code. Then it is time to think about optimizations and variants of the LA Algorithm. In Section 3.4 we first recall the concept of mutants by J. Ding et al. [67, 140, 142, 143], and then highlight our first hybrid algorithm, called the MLA Algorithm, that uses the mutant strategy to improve the LA Algorithm. The second hybrid algorithm, called the MLA2 Algorithm, is presented in Section 3.5. It uses the improved mutant strategy to speed up the LA Algorithm. Section 3.6 reports on experiments with the mutant variants of the LA Algorithm.

The main drawback of the LA Algorithm and its mutant variants is that they can solve only systems having a unique solution. In order to solve systems with a finite number of solutions, we investigate some linear algebra technique from border basis theory in Chapter 4. To this end we review the main technique for computing border bases in Section 4.1. In particular, we recall a version of the Border Basis Algorithm which is actually called the *Improved Border Basis Algorithm* in [111]. In Section 4.2 we propose two hybrid algorithms, called MBBA and MBBA2, which combine border

basis theory and the concept of mutants to accelerate the computation of border bases over finite fields. These two hybrid algorithms use the mutant strategy and the improve mutant strategy respectively. The efficiency of these newly developed hybrid techniques is examined using standard cryptographic examples in Section 4.3.

Chapter 5 starts the second leg of our journey. After formulating the solution of a system of polynomial equations as an integer programming problem, we apply standard IP solvers inside our algebraic techniques. Several techniques and strategies are developed and their efficiency is examined using standard cryptographic examples at the end of each section. Section 5.1 serves as a foundation for coming sections. We discuss our approach to use techniques from integer linear programming, review some necessary theoretical concepts, and discuss some standard techniques used for modeling and solving mixed integer linear problems. We study recent suggestions, by M. Kreuzer [117] and J. Borghoff et al. [31], of transferring the problem of solving a system of polynomial equations over $\mathbb{F}_2$ into a mixed integer linear programming problem in Section 5.2. Section 5.3 is devoted to studying strategies that enable the transformation of a 0-1 polynomial programming problem into a 0-1 linear programming problem to be effected with a reduced number of constraints. In particular, we investigate more economic ways of transferring 0-1 programs into 0-1 linear programs and generalize the approach in Section 5.2. We present a new conversion method based on propositional logic and pseudo-boolean optimization in Section 5.4. This new method also enables us to export several strategies from propositional logic to model our MILP problem. In Section 5.5 we develop new hybrid techniques for modeling a MILP problem. These hybrid conversion techniques combine the ideas studied in the previous sections and can be equipped with several strategies to achieve efficiency. To conclude this chapter, we give a brief conclusion and present a comparison, using plots and tables, of all techniques studied in this chapter in Section 5.6. The experimental results are presented for polynomial systems coming from the CTC and the small scale AES cipher.

Chapter 6 marks a first step towards developing techniques using MINLP and linear Diophantine system solving, and offers several future research directions. In Section 6.1 we review some necessary concepts from the theory of MINLP with a focus on non-convex MINLP. Afterwards, we reformulate the polynomial conversion methods presented in Chapter 5 to transfer the problem of solving a system of polynomial equations over $\mathbb{F}_q$ into a non-convex MINLP problem. We try to see what can be achieved if we employ a MINLP solver instead of a MILP solver. Towards the end of this section, we present experimental results using the open-source solver COUENNE

[56] which solves non-convex MINLP problems. In Section 6.2 first we review some necessary concepts from number theory with focus on methods for solving systems of linear Diophantine equations. Afterwards, we reformulate the polynomial conversion methods presented in Chapter 5 to transfer a system of polynomial equations over $\mathbb{F}_2$ into a system of linear Diophantine equations. After reformulation, we apply the straightforward approach for solving systems of linear Diophantine equations for finding non-negative integer solutions. Furthermore, we highlight some ideas to spark further research in this direction. Finally, we illustrate the performance of this technique using some concrete examples.

We address some approaches to apply numerical methods for solving systems of polynomial equations over $\mathbb{F}_2$ in Chapter 7. Section 7.1 presents a quick overview of various Newton methods used in this chapter. We use iterative (Newton) methods which need an initial starting guess. Section 7.2 presents a quick overview of homotopy continuation methods used later in this chapter. In particular, we focus on path following techniques for finding the unique isolated solution of a system of nonlinear polynomial equations given that we know that such a solution exists. In Section 7.3, we reconsider the conversion techniques of Chapter 5 to obtain a system of nonlinear polynomial equations over $\mathbb{R}$ which can be solved using different algorithms from numerical analysis. In particular, we investigate the use of homotopy continuation methods and Newton's method. We generalize the approach in [24, 124] and extend this work further. Furthermore, we present some ad-hoc tricks. Finally, the performance of the techniques is investigated using some concrete examples.

For using homotopy continuation methods in our algebraic settings, we have developed two ApCoCoA interfaces `bertini` and `hom4ps`. These interfaces are introduced in Appendix A. They are able to call the full functionality of `Bertini` [21] and `HOM4PS` [127] for computations with homotopy continuation methods inside ApCoCoA. Appendix B introduces the functions which implement the linear algebra techniques of Chapters 3 and 4. These functions are available as a part of the package `CharP` of ApCoCoA. The functions which implement the conversion algorithms of Chapter 5 are introduced in Appendix C. These functions are available as a part of ApCoCoA package `glpk`. Finally, Appendix C is provided to give a brief description about the implementations of the conversion algorithms of Chapters 6 and 7. Each function in the above packages is explained with its syntax and an example describing its usage.

# Chapter 2

# Preliminaries

In this chapter we introduce some basic concepts and notations useful for the remainder of this thesis. In particular, we develop the mathematical tools necessary for understanding polynomial system solving over finite fields. We start with introducing finite fields, then concentrating on the general problem of systems of multivariate polynomial equations over finite fields. Furthermore, we discuss complexity of polynomial system solving over finite fields and its applications to cryptography and cryptanalysis.

## 2.1 Finite Fields

As finite fields are a very basic building block for many cryptographic protocols that play an essential role in modern life, we start with introducing them. Loosely speaking, a (finite) field consists of a (finite) set of elements, and two operations, namely addition (denoted "+") and multiplication (denoted "$\cdot$"). These operations need to fulfil certain criteria. Details can be found in any book of algebra such as [130].

**Definition 2.1.1.** A **ring** $(R, +, \cdot)$ is a set $R$, together with two binary operations, denoted by $+$ and $\cdot$, such that $(R, +)$ is an abelian group. $\cdot$ is associative and the distributive laws hold.

Recall that a ring is called a *ring with identity* if the ring has a multiplicative identity. A ring is called *commutative* if $\cdot$ is commutative. In this thesis by a ring we shall always mean a *commutative ring with identity element*. An *ideal* $I$ of a ring $R$ is a subring of $R$ such that for all $a \in I$ and $r \in R$ we have $ar \in I$ and $ra \in I$.

**Definition 2.1.2. A field** $(K, +, \cdot)$ is a ring such that $(K \setminus \{0\}, \cdot)$ is a group. If a field $(K, +, \cdot)$ contains only finitely many elements, it is called a **finite field**.

Note that for brevity, we usually write $xy$ instead of $x.y$. If it is clear from the context which addition and multiplication we use with the field, we also write $K$ instead of $(K, +, \cdot)$. Our first examples of finite fields are the residue class fields $\mathbb{Z}/\langle p \rangle$, where $\langle p \rangle$ is a principal ideal generated by a prime $p$.

**Definition 2.1.3.** Let $p$ be a prime number, let $\mathbb{F}_p$ be the set $\{0, \ldots, p-1\}$ of integers and let $\varphi : \mathbb{Z}/\langle p \rangle \longrightarrow \mathbb{F}_p$ be the map defined by $\varphi(\bar{a}) = a$ for $a \in \{0, \ldots, p-1\}$. Then $\mathbb{F}_p$, equipped with the field structure induced by $\varphi$, is a finite field, called the **Galois field** of order $p$.

Note that computing with elements of $\mathbb{F}_p$ means ordinary arithmetic of integers with reduction modulo $p$. We also know that every finite field has prime characteristic and the prime subfield of a finite field $K$ is isomorphic to $\mathbb{F}_p$.

Before going into further details of finite fields. We need to recall a few results from field theory. Let $K \subseteq L$ be a field extension. The extension (field) $K(\alpha)$ of $K$ obtained by adjoining the element $\alpha \in L$ is called a *simple extension* of $K$ and $\alpha$ is called a *defining element* of $K(\alpha)$ over $K$. If $\alpha$ is algebraic over $K$ then there exists a uniquely determined monic polynomial $f \in K[x]$ such that $f(\alpha) = 0$, where $K[x]$ is the polynomial ring over $K$ in one indeterminate. The uniquely determined monic polynomial $f \in K[x]$ is called the *minimal polynomial (or defining polynomial, or irreducible polynomial)* of $\alpha$ over $K$. By the degree of $\alpha$ over $K$ we mean the degree of $f$.

**Proposition 2.1.4.** *Let $\alpha \in L$ be algebraic of degree $e$ over $K$ and let $f$ be the minimal polynomial of $\alpha$ over $K$.*

*a) The extension field $K(\alpha)$ is isomorphic to $K[x]/(f)$, where $K[x]$ is the polynomial ring over $K$ in one indeterminate.*

*b) The set $\{1, \alpha, \ldots, \alpha^{e-1}\}$ is a basis of $K(\alpha)$ over $K$.*

*c) Every $\beta \in K(\alpha)$ is algebraic over $K$ and its degree over $K$ is a divisor of $e$.*

*Proof.* See [130], Theorem 1.86. □

Due to the proposition above any element of $K(\alpha)$ can be uniquely represented in the form $a_0 + a_1\alpha + \cdots + a_{e-1}\alpha^{e-1}$ with $a_i \in K$ for $0 \leq i \leq e-1$. The construction

of a simple algebraic extension without reference to a previously given larger field $L$ is given by the following theorem.

**Proposition 2.1.5.** *Let $K$ be a field, $K[x]$ be the polynomial ring in one indeterminate and let $f \in K[x]$ be an irreducible polynomial. Then there exists a simple algebraic extension of $K$ with a root of $f$ as a defining element.*

*Proof.* See [130], Theorem 1.87. □

The construction of a simple algebraic extension as given by above proposition is some times refereed as *root adjunction*. By adjoining different roots of the polynomial $f$ we can get the same simple algebraic extension as given by the following result.

**Proposition 2.1.6.** *Let $K$ be a field and let $K[x]$ be the polynomial ring in one indeterminate. Let $\alpha$ and $\beta$ be two roots of the polynomial $f \in K[x]$ that is irreducible over $K$. Then $K(\alpha)$ and $K(\beta)$ are isomorphic under an isomorphism mapping $\alpha$ to $\beta$ and keeping the elements of $K$ fixed.*

*Proof.* See [130], Theorem 1.89. □

Now the *splitting field* is the extension field to which all roots of the polynomial $f$ belong.

**Theorem 2.1.7.** *Let $K$ be a field, $K[x]$ be the polynomial ring in one indeterminate, and let $f$ be a polynomial of positive degree in $K[x]$. Then there exists a splitting field of $f$ over $K$. Any two splitting fields of $f$ over $K$ are isomorphic under an isomorphism which keeps the elements of $K$ fixed and maps roots of $f$ into each other.*

*Proof.* See [130], Theorem 1.91. □

The splitting fields are obtained from $K$ by adjoining finitely many algebraic elements over $K$, and the splitting field of $f$ over $K$ is a finite extension of $K$. We can identify isomorphic field due to Theorem 2.1.7. Therefore, we can speak of the splitting field of $f$ over $K$. Recall that any finite field with characteristic $p$ has $q = p^e$ elements for some positive integer $e$ and if $\mathbb{F}_q$ is a finite field with $q$ elements and $\mathbb{F}_p$ is a subfield of $\mathbb{F}_q$, then $\mathbb{F}_q$ is a splitting field of $x^q - x$ over $\mathbb{F}_p$. The following characterization theorem for finite fields tells us more about finite fields.

**Theorem 2.1.8. (Existence and Uniqueness of Finite Fields)**
*Let $p$ be a prime and let $e$ be a positive integer.*

*a) There exists a finite field with $p^e$ elements.*

*b) There exists a unique (up to isomorphism) field having $p^e$ elements.*

*Proof.* See [130], Theorem 2.5. □

Uniqueness in Theorem 2.1.8 is a consequence of the uniqueness (up to isomorphism) of splitting fields. The uniqueness provides the justification for speaking of the finite field (or Galois filed) with $q = p^e$ elements, or of the finite field (or the Galois field) of order $q$. From now on, we shall denote this field by $\mathbb{F}_q$. Now we know that all finite fields of same size are isomorphic, and so we have constructed the finite field of size $p^e$ which is isomorphic to $\mathbb{F}_p[x]/\langle f(x)\rangle$ where $f(x) \in \mathbb{F}_p[x]$ is an irreducible polynomial of degree $e$. Since finite fields are of central importance in this thesis, we briefly recall some useful results for finite fields.

**Lemma 2.1.9.** *Let $\mathbb{F}_q$ be a finite field with $q$ elements, then every element $a \in \mathbb{F}_q$ satisfies $a^q = a$.*

*Proof.* If $a$ is zero, then $0^q = 0$ is trivial. If $a$ is non-zero, then the nonzero elements of $\mathbb{F}_q$ form a group of order $q-1$ under multiplication. Thus $a^{q-1} = 1$ for all $a \in \mathbb{F}_q$ with $a \neq 0$, and multiplication by $a$ yields the required result. □

A useful property of the multiplicative group $\mathbb{F}_q^\times$ of $\mathbb{F}_q$ is given by the following result.

**Lemma 2.1.10.** *Let $\mathbb{F}_q$ be a finite field. The multiplicative group $\mathbb{F}_q^\times$ of non-zero elements of $\mathbb{F}_q$ is cyclic.*

*Proof.* See [130], Theorem 2.8. □

Lemmas 2.1.9 and 2.1.10 will prove particularly useful in the context of systems of polynomial equations defined over extension fields and in the context of polynomial maps. Recall that the $p^{\text{th}}$ root of an element $a \in \mathbb{F}_q$ is uniquely determined. For instance, if we have $b, c \in \mathbb{F}_q$ such that $b^p = c^p = a$, then we have $b^p - c^p = (b-c)^p = 0$, which implies $b = c$.

**Definition 2.1.11.** The map $\varphi : \mathbb{F}_q \longrightarrow \mathbb{F}_q$ defined by $\varphi(a) = a^p$ is called the **Frobenius map**.

Note that Frobenius map is a *field homomorphism*, since for all $a, b \in \mathbb{F}_q$ we have

$$(a+b)^p = a^p + b^p,$$
$$(ab)^p = a^p b^p.$$

Since $\mathbb{F}_q$ is a finite field, $\varphi$ is bijective. In the field $\mathbb{F}_p$ every element is its own $p^{\text{th}}$ root. This can be generalized to an arbitrary finite field $\mathbb{F}_q$ as follows. For all $a \in \mathbb{F}_q$ we have the map $a \mapsto a^{p^{e-1}}$ such that $a^{p^{e-1}} = a^q = a$. Thus this map provides $p^{\text{th}}$ roots.

### Representing Elements of Finite Fields

From now on, let $p$ be a prime number, let $q = p^e$ for some $e > 0$, and let $\mathbb{F}_q$ be the finite field with $q$ elements. Recall that there are three ways of representing the elements of the finite field $\mathbb{F}_q$ with $q = p^n$ elements. For details we refer to [130], Chapter 2. Here we recall the way which is based on the fact that $\mathbb{F}_q$ is a simple algebraic extension of $\mathbb{F}_p$. Let $f(x) \in \mathbb{F}_p[x]$ be an irreducible polynomial of degree $e$, then $f(x)$ has a root $\alpha$ in $\mathbb{F}_q$ according to Proposition 2.1.4. So we have $\mathbb{F}_q = \mathbb{F}_p(\alpha)$. In this way we may view $\mathbb{F}_q$ as the residue class ring $\mathbb{F}_p[x]/\langle f(x)\rangle$ and every element of $\mathbb{F}_q$ can be uniquely expressed as a polynomial in $\alpha$ over $\mathbb{F}_p$ of degree less than $e$. Note that this representation is unique (up to isomorphism). Therefore it does not matter which irreducible polynomial $f(x) \in \mathbb{F}_p[x]$ we choose. In other words, all finite fields of the same size are isomorphic.

**Example 2.1.12.** We can represent elements of the field $\mathbb{F}_4$ as follows. The field $\mathbb{F}_4$ is a simple algebraic extension of the field $\mathbb{F}_2$ of degree 2. The extension $\mathbb{F}_4$ is obtained by adjunction of a root $\alpha$ of an irreducible polynomial of degree 2 over $\mathbb{F}_2$, say $f(x) = x^2 + x + 1 \in \mathbb{F}_2[x]$. We have $f(\alpha) = \alpha^2 + \alpha + 1 = 0 \in \mathbb{F}_4$. The multiplicative group for the non-zero elements of $\mathbb{F}_4$ is generated by the field element $\alpha$ which satisfies $\alpha^2 + \alpha + 1 = 0$. The elements of $\mathbb{F}_4$ can be represented as $\{0, 1, \alpha, \alpha^2\}$. The operation tables for $\mathbb{F}_4$ can be easily constructed with $\alpha$ playing the role of the residue class $\bar{x} \in \mathbb{F}_p[x]/\langle f(x)\rangle$.

For the other two ways of expressing the elements of $\mathbb{F}_q$ we refer to [130], Chapter 2.

## 2.2 Solving Systems of Polynomial Equations Over Finite Fields

After having a short look at finite fields, we move on to the problem of solving a system of multivariate polynomial equations over finite fields. From now on, let $p$ be a prime number, let $q = p^e$ for some $e > 0$, and let $K = \mathbb{F}_q$ be the finite field with $q$ elements, and let $K[x_1, \ldots, x_n]$ be the ring of polynomials over the field $K$. Let $f_1, \ldots, f_\ell \in K[x_1, \ldots, x_n]$ be a set of non-zero polynomials. Let $I$ be the ideal generated by the polynomials $f_1, \ldots, f_\ell$. We are interested in finding $K$-rational solutions of the following system of polynomial equations.

$$\begin{aligned} f_1(x_1, \ldots, x_n) &= 0 \\ &\vdots \\ f_\ell(x_1, \ldots, x_n) &= 0 \end{aligned}$$

**Definition 2.2.1.** Let $K \subseteq L$ be a field extension, and let $S \subseteq L^n$. Consider the set of all polynomials $f \in K[x_1, \ldots, x_n]$ such that $f(a_1, \ldots, a_n) = 0$ for all points $(a_1, \ldots, a_n) \in S$. This set forms an ideal of the polynomial ring $K[x_1, \ldots, x_n]$. This ideal is called the **vanishing ideal** of $S$ in $K[x_1, \ldots, x_n]$ and is denoted by $\mathcal{I}(S)$.

Considering $K^n$ as a finite point set, the polynomials $f_1, \ldots, f_\ell$ can be modified by adding elements of the vanishing ideal

$$\mathcal{I}(K^n) = \{g \in K[x_1, \ldots, x_n] \mid g(a_1, \ldots, a_n) = 0 \text{ for all } (a_1, \ldots, a_n) \in K^n\}.$$

**Definition 2.2.2.** Let $K$ be the field as above and let $K[x_1, \ldots, x_n]$ be the ring of polynomials over the field $K$. Then the **field polynomials** of the ring $K[x_1, \ldots, x_n]$ are defined as the set

$$\{x_1^q - x_1, \ldots, x_n^q - x_n\}.$$

The vanishing ideal of $K^n$ in $K[x_1, \ldots, x_n]$ is $\mathcal{I}(K^n) = \langle x_1^q - x_1, \ldots, x_n^q - x_n \rangle$. From now on, we shall call this ideal the **field ideal** of the ring $K[x_1, \ldots, x_n]$.

If we look for solutions in $K^n$, we may include equations of the field ideal while solving the system of polynomial equations $f_1 = 0, \ldots, f_\ell = 0$. This gives us the

following system of equations.

$$\begin{aligned} f_1(x_1,\ldots,x_n) &= 0 \\ &\vdots \\ f_\ell(x_1,\ldots,x_n) &= 0 \\ x_1^q - x_1 &= 0 \\ &\vdots \\ x_n^q - x_n &= 0 \end{aligned}$$

Recall that by condition $d$) of the finiteness criterion (see [120], Proposition 3.7.1), an ideal $J$ of the ring $K[x_1,\ldots,x_n]$ is called *zero-dimensional* if the $K$-vector space $K[x_1,\ldots,x_n]/J$ is finite dimensional. Now consider the ideal $J = I + \langle x_1^q - x_1, \ldots, x_n^q - x_n\rangle$. The $K$-vector space $K[x_1,\ldots,x_n]/J$ is generated by the finite set $\{\bar{x}_1^{\alpha_1}\ldots\bar{x}_n^{\alpha_1} \mid \alpha_i < q\}$, where $\bar{x}_i$ is the residue class of $x_i$ in $K[x_1,\ldots,x_n]/\langle x_1^q - x_1,\ldots,x_n^q - x_n\rangle$. Therefore appending the field polynomials to the ideal $I$ will assure that the ideal $I$ is zero-dimensional. Next we see that the ideal $J$ is a radical ideal. One way of proving the ideal $J$ a radical ideal is using *Seidenberg's Lemma* which we quote from [120], Proposition 3.7.15.

> *Let $K$ be a field, let $P = K[x_1,\ldots,x_n]$, and let $I \subseteq P$ be a zero-dimensional ideal. Suppose that, for every $i \in \{1,\ldots,n\}$, there exists a non-zero polynomial $g_i \in I \cap K[x_i]$ such that $\gcd(g_i, g_i') = 1$. Then $I$ is a radical ideal.*

Note that after appending field polynomials, by Seidenberg's Lemma the ideal $I$ becomes radical ideal because for every $i \in \{1,\ldots,n\}$ we have a field polynomial $g_i = x_i^q - x_i \in J \cap K[x_i]$ such that $\gcd(g_i, g_i') = \gcd(x_i^q - x_i, qx_i^{q-1} - 1) = \gcd(x_i^q - x_i, -1) = 1$. Another way of proving the ideal $J$ to be radical is the following.

**Lemma 2.2.3.** *Let $f_1,\ldots,f_\ell \in K[x_1,\ldots,x_n]$ be a set of polynomials. Let $I = \langle f_1,\ldots,f_\ell\rangle$ and $J = I + \langle x_1^q - x_1,\ldots,x_n^q - x_n\rangle$. Then $J$ is a radical ideal.*

*Proof.* To show that the ideal $J$ is a radical ideal. We need to show $\sqrt{J} = J$. Since by definition, any ideal is contained in its radical, we only need to prove $\sqrt{J} \subseteq J$. Let $f \in \sqrt{J}$. By definition of radical ideal, for some integer $s \geq 0$, $f^s \in J$. Let $\varphi$ be the Frobenius homomorphism. Thus we have $\varphi(f^s) = \varphi(f)^s \in \varphi(I)$. Now to show that $f \in J$ we only need to show that $\varphi(f) \in \varphi(I)$.

Since $\varphi$ is the Frobenius homomorphism therefore for any $g \in K[x_1,\ldots,x_n]$, we

have $\varphi(g)^q = \varphi(g)$. Now without loss of generality we can assume $s < q$ in $\varphi(f)^s$. Since $\varphi(f)^s \in \varphi(I)$, therefore $\varphi(f)^s\varphi(f)^{q-s} = \varphi(f)^q = \varphi(f) \in \varphi(I)$. □

In this way, the ideal $J$ becomes a zero-dimensional radical ideal. Now we can translate strong version of Hilbert's Nullstellensatz over finite fields as follows.

**Theorem 2.2.4. (Hilbert's Nullstellensatz in Finite Fields)**
*Let $f_1, \ldots, f_\ell \in K[x_1, \ldots, x_n]$ be a set of polynomials. Let $I = \langle f_1, \ldots, f_\ell \rangle$ and $J = I + \langle x_1^q - x_1, \ldots, x_n^q - x_n \rangle$ be ideals of $K[x_1, \ldots, x_n]$, then*

$$\mathcal{I}(\mathcal{Z}(I)) = J$$

*Proof.* Applying Hilbert's Nullstellensatz (see [120], Theorem 2.6.16) to $J$ and using Lemma 2.2.3, we have $\mathcal{I}(\mathcal{Z}(J)) = J$. Since $\mathcal{Z}(\langle x_1^q - x_1, \ldots, x_n^q - x_n \rangle) = K^n$, therefore $\mathcal{Z}(J) = \mathcal{Z}(I) \cap K^n$. Thus, we have $\mathcal{I}(\mathcal{Z}(I)) = J$. □

Let $R$ be the following residue class ring

$$K[x_1, \ldots, x_n]/\langle x_1^q - x_1, \ldots, x_n^q - x_n \rangle,$$

where we reduce everything modulo the field polynomials. So far we have discussed polynomial ideals in the polynomial ring $K[x_1, \ldots, x_n]$. Another representation can be achieved by defining them in the residue class ring $R$. Using this representation we can represent polynomials in a more convenient way in computer and benefit from specialized algorithms or implementations over the ring $R$. Furthermore, for our purpose, some times it is convenient to work in the ring $R$ as we see in Chapters 3 and 4. The elements of $R$ are residue classes. We can represent residue classes by polynomials using Macaulay's Basis Theorem (see [120], Theorem 1.5.7). We let $\mathbb{T}^n$ denote the set of terms of $K[x_1, \ldots, x_n]$.

**Remark 2.2.5. (Representing Elements of $R$ by Polynomials)**
Macaulay's Basis Theorem tells us how to compute effectively in $R$. Let $\sigma$ be a (degree compatible) term ordering, then for the field ideal we have $\mathrm{LT}_\sigma(\langle x_1^q - x_1, \ldots, x_n^q - x_n \rangle) = \langle x_1^q, \ldots, x_n^q \rangle$. Hence we can represent every element uniquely as a finite linear combination of the residue classes of the elements of $\mathbb{T}^n \setminus \mathrm{LT}_\sigma\{\langle x_1^q - x_1, \ldots, x_n^q - x_n \rangle\} = \{\bar{x}_1^{\alpha_1} \ldots \bar{x}_n^{\alpha_1} \mid \alpha_i < q\}$, where $\bar{x}_i$ is the residue class of $x_i$.

So the elements of $R$ are residue classes where each residue class will be represented by a polynomial in the following way. We assume that $t = x_1^{\alpha_1} \ldots x_n^{\alpha_n} \in \mathbb{T}^n$ is a term

in $R$, then $\alpha_i < q$, $1 \leq i \leq n$. The residue class ring $R$ is a finite dimensional $K$-vector space and its dimension is equal to the cardinality of the set of all terms in $R$. For simplicity, we will call $R$ a polynomial ring in $n$ indeterminates $x_1, \ldots, x_n$ where each term is reduced modulo the so-called field polynomials $x_1^q - x_1, \ldots, x_n^q - x_n$. In view of this, we can uniquely represent a residue class (an element of $R$) by a polynomial, where each $x_i$, $1 \leq i \leq n$, has a power less than $q$. In particular, every residue class of $R$ has a unique polynomial representation of the form $f = \sum_{\alpha \in \mathbb{N}^n} c_\alpha t_\alpha$ where $t_\alpha = x_1^{\alpha_1} \ldots x_n^{\alpha_n}$, such that $\alpha_i < q$, $1 \leq i \leq n$, and only finitely many elements $c_\alpha \in K$ are different from zero. Since each element (residue class) in $R$ is represented by a polynomial, we can define the degree, leading term, and leading variable of this polynomial in the natural way.

Our goal in this thesis is to find the zero set of the zero-dimensional radial ideal $J = I + \langle x_1^q - x_1, \ldots, x_n^q - x_n \rangle$. Moreover, we are interested in finding only one zero of the ideal $J$, since usually we may assume that it has only one zero. Solving systems of polynomial equations over the finite field $\mathbb{F}_2$ has special importance due to its applications in cryptography and cryptanalysis, and due to the fact that it can be generalized to any finite field. Especially, to the finite field $\mathbb{F}_{2^e}$, for some integer $e > 0$. For instance, for a novel way of transforming a polynomial system over $\mathbb{F}_{2^e}$ to a (larger) system over $\mathbb{F}_2$ we refer to [106], Section 3.

## 2.3 Applications

In this section, we see some applications of polynomial system solving over finite fields in *cryptography* and *cryptanalysis*. The two branches of cryptography are *Asymmetric Cryptography (or Public Key Cryptography)* and *Symmetric Key Cryptography*. Currently, the security of most algorithms that we know in Asymmetric Cryptography for encryption or signatures rely on the (not proved) intractability of the *factorization (IFP) or discrete log problem (DLP)*. Due to the improvements in algorithms for solving IFP and DLP, parameters of these cryptosystems are required to be modified in order to achieve a reasonable level of security. For instance, 156 and 200-digit RSA numbers have already been factorized. In 1999, Peter Shor discovered polynomial time algorithms to solve the IFP and DLP on a 'hypothetical' quantum computer. Once quantum computers have been developed, cryptosystems based on these problems will not remain secure any more.

So today one of the problems of Asymmetric Cryptography is to find new and effi-

cient algorithms for encryption or signatures that are as secure on quantum computers as well as on conventional computers. *Multivariate cryptography* is one of the main fields of research for the development of *multivariate algebraic cryptosystems* which are believed to be secure against attacks with quantum computers. Furthermore, the development of *Gröbner Basis cryptosystems* is an active area of research in the Gröbner Basis community. It is believed that if such cryptosystems are developed successfully, they will not be threatened by the development of quantum computers. Thus, the security of future cryptosystems seems to be related to the problem of solving systems of multivariate quadratic equations over a finite field.

### 2.3.1 Algebraic Attacks

Cryptography is the study of methods of sending messages in disguised form so that only the intended recipients can remove the disguise and read the message. The message we want to sent is called the *plaintext*, and the disguised message is called the *ciphertext*. The process of converting a plaintext to a ciphertext is called *enciphering or encryption*, and the reverse process is called *deciphering or decryption*. The plaintext and ciphertext are broken up into message units. We refer to [114] for a detailed study of the subject. In the following, we follow [117] to describe some important applications.

**Definition 2.3.1.** A **cryptosystem**, also called as cipher or encryption scheme, has the following basic components:

1. A set of **plaintext units** $\mathcal{P}$ which is also called the **message space**.
2. A set of **ciphertext units** $\mathcal{C}$ which is also called the **ciphertext space**.
3. A set $\mathcal{K}$ called the **key space**.
4. An **encryption map**, $\varepsilon_k : \mathcal{P} \longrightarrow \mathcal{C}$, for every element $k \in \mathcal{K}$.
5. A **decryption map**, $\delta_k : \mathcal{C} \longrightarrow \mathcal{P}$, for every element $k \in \mathcal{K}$.
6. Finally a map $\eta : \mathcal{K} \longrightarrow \mathcal{K}$ such that $\delta_{\eta(k)} \circ \varepsilon_k = \mathrm{id}_{\mathcal{K}}$ for every element $k \in \mathcal{K}$. For every element $k \in \mathcal{K}$ the pair $(k, \eta(k))$ is called a **key pair**.

As we know, the two major cryptosystems that have been used in modern cryptography are known as *symmetric cryptosystems* and *asymmetric cryptosystems*. In other words, if we can compute $\eta_k$ efficiently using the knowledge of the key $k$ and the

encryption map $\varepsilon_k$, the cryptosystem is called symmetric. Otherwise, the system is called asymmetric. If the cryptosystem is asymmetric the knowledge of the key $k$ is used depending on weather the cipher is a *block cipher* such as *AES* or a *stream cipher* such as *DES*. In a block cipher, after breaking the plaintext into plaintext units, the encryption is done using a fixed key $k$. In a stream cipher, we generate a sequence $k_1, k_2, \ldots$ of keys called the *key stream*. The key steam is generated using some chosen function and then $k_1, k_2, \ldots$ are used for the encryption of the individual plaintext units.

Many cryptographic protocols that play an essential role in modern life are build on cryptosystems. In his seminal paper [160], C.E. Shannon, who is also known as the father of information theory, remarked:

> *Thus, if we could show that solving a certain system requires at least as much work as solving a system of simultaneous equations in a large number of unknowns, of a complex type, then we would have a lower bound of sorts for the work characteristic.*

In the following, let $p$ be a prime number, let $q = p^e$ for some $e > 0$, and let $K = \mathbb{F}_q$ be the finite field with $q$ elements.

**Definition 2.3.2.** A **polynomial map** is a map $f : K^n \longrightarrow K^m$ such that for all points $(x_1, \ldots, x_n) \in K^n$,

$$f(x_1, \ldots, x_n) = (f_1(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n)),$$

for suitable polynomials $f_1, \ldots, f_m \in K[x_1, \ldots, x_n]$.

The set of all zero of $f$ is precisely the set of all solutions of the simultaneous equations $f_1 = \cdots = f_m = 0$. Polynomial maps can be defined on any non-empty subset of $K^n$. We consider the sets $\mathcal{P}$ and $\mathcal{C}$ are (subsets of) finite dimensional vector spaces over a finite fields, usually of characteristic 2.

**Remark 2.3.3.** Over the field $K$, for every map $f : K^n \longrightarrow K^m$ there exist polynomials $f_1, \ldots, f_m \in K[x_1, \ldots, x_n]$ such that

$$f(x_1, \ldots, x_n) = (f_1(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n)),$$

for all $(x_1, \ldots, x_n) \in K^n$. The polynomials $f_i$ are not uniquely determined. In other words, the map $f$ is a polynomial map. Since we are interested in finding $K$-rational

solutions, we consider $K^n$ as finite point set and modify the polynomials $f_i$ by adding elements of the field ideal (vanishing ideal $\mathcal{I}(K^n)$). The vanishing ideal is generated by the field polynomials. Every time we represent an encryption map (or a family of encryption maps) via polynomials $f_1, \ldots, f_m$, we modify them using field polynomials.

In [117], we have the following example which gives us a non-standard look at the RSA cryptosystem.

**Example 2.3.4.** Consider the RSA cryptosystem. Choose two prime numbers $p = 3$ and $q = 5$ such that $n = 15 = p.q$. Knowing the factorization of $n$, we can easily compute $\varphi(n) = (p-1)(q-1) = n + 1 - p - q$. Next we randomly choose an integer $e = 5$ known as public exponent between 1 and $\varphi(n)$ which is prime to $\varphi(n)$. The secret exponent is $d = 5$ such that $de \equiv 1(\mod 8)$ with $8 = \varphi(n)$. We represent the plaintest and ciphertext units as tuples $(a_0, a_1, a_2, a_3) \in \mathbb{F}_2^4$ corresponding to elements $a_0 + 2a_1 + 4a_2 + 8a_3 \in \mathbb{Z}/(15)$. By a straightforward calculation we can represent $\varepsilon_5(a_0, a_1, a_2, a_3) = (a_0 + 2a_1 + 4a_2 + 8a_3)^5$ by the tuple $(c_0(a_0, a_1, a_2, a_3), \ldots, c_3(a_0, a_1, a_2, a_3)) \in \mathbb{F}_2^4$ where

$$\begin{aligned}
c_0 &= a_0a_1a_3a_3 + a_0a_1a_2 + a_0a_2 + a_0a_3 + a_2a_3 + a_0 + a_3 \\
c_1 &= a_0a_1a_2a_3 + a_0a_1a_2 + a_0a_1a_3 + a_0a_2a_3 + a_0a_1 + a_1 + a_2 + a_3 \\
c_2 &= a_1a_2a_3 + a_0a_1 + a_1a_2 + a_1a_3 + a_1 + a_2 \\
c_3 &= a_0a_1a_2a_3 + a_0a_1a_2 + a_1a_2a_3 + a_0a_1 + a_0a_2 + a_0a_3 + a_2a_3 + a_3
\end{aligned}$$

Now consider the ciphertext $(1, 1, 0, 0)$. We can recover the plaintext from this cipher by solving the polynomial system $c_0 - 1 = 0, c_1 - 1 = 0, c_2 = 0, c_3 = 0$ for $\mathbb{F}_2$-rational solutions. The most obvious way to solve this system is to compute a Gröbner basis. The reduced Gröbner basis of the ideal $I = \langle c_0 - 1, c_1 - 1, c_2, c_3, a_0^2 - a_0, \ldots a_3^2 - a_3 \rangle$ is $\{a_0 - 1, a_1 - 1, a_2, a_3\}$, therefore plaintext unit was $(1, 1, 0, 0)$ which agrees with $3^5 \equiv 3(\mod 15)$.

**Remark 2.3.5.** We do not know a standard way to express the RSA cryptosystem and many others as a systems of polynomial equations. Therefore, a natural question is to ask about the construction of the polynomials $f_1, \ldots, f_m$ which represent the encryption map $\varepsilon_k$. The encryption map $\varepsilon_k$ carry some specific information with it which is exploited while construction the polynomials $f_1, \ldots, f_m$. This suggests that the polynomials $f_1, \ldots, f_m$ are constructed on case-by-case basis. For instance, for the

construction of polynomials $f_1, \ldots, f_m$ from the so-called Courtois Toy Cipher (CTC) see [57].

A partial answer to the above question is to use the *Buchberger-Möller Algorithm* (see [121], Theorem 6.3.10 and Corollary 6.3.11 for general setting and [117], Proposition 3.1 for cryptanalysis setting) that yields all polynomials which model the encryption map $\varepsilon_k$ for the given plaintext units and keys. But this is possible in practice if the space of plaintext units $\mathcal{P}$ (and possibly the key space) is not too large. For large real-world cryptosystems, we can generate polynomial relations between the plaintext and key bits tuples, and the ciphertext tuple.

Furthermore, for more details and a description of several attack scenarios using the algebraic representation of the encryption $\varepsilon_k$ and decryption $\delta_k$ maps we refer to the article titled "Algebraic Attacks Galore!" [117] by M. Kreuzer. We can summarize the discussion above as follows. *The main task for a successful algebraic attack on a cipher (or for examining the security of a cipher) is to solve a multivariate polynomial system over a finite field.* Therefore, in this thesis we develop new techniques that can be used in the context of polynomial systems derived from algebraic attacks to examine the security of different ciphers.

### 2.3.2 Cryptographic Polynomial Systems

To test the performance of the developed techniques we consider systems of polynomial equations coming from applications in cryptography and cryptanalysis. In his diploma thesis [131] J. Limbeck implemented the polynomials representing the encryption functions of a number of important cryptosystems, e.g. DES, AES, CTC, Serpent, Keeloq, HFE and a number of variations of these. These implementations in ApCoCoA [12] are available from the author upon request. We use some of these implementations to generate systems of polynomial equations. The systems used for experiments are available at `http://apcocoa.org/polynomialsystems/`. In particular, we consider the following cryptosystems.

#### Courtois Toy Cipher (CTC)

In May 2006, Nicolas Courtois published in [57] the specifications of the so-called Courtois Toy Cipher (CTC) along with a way to express this cipher as a multivariate equation system over $\mathbb{F}_2$. He claims to have broken this cipher by solving the associated equation system faster than exhaustive key search. In particular, he claims to have

broken a 255-bit block size and six round instance of CTC in under one hour on his notebook computer. Nicolas Courtois calls his attack "fast algebraic attack against block ciphers". However he did not publish the details of his attack as he was afraid his attack could be extended to break AES quite quickly: "In order to protect the United States government, the financial institutions, mobile phone operators, and hundreds of millions of other people that use AES, from attackers". Our assumption is that CTC can be broken with algebraic attacks effectively as it was designed for that purpose. But as the actual attack of Nicolas Courtois is unpublished the second purpose is to attack CTC and report the results of observations on these experiments. This will also serve our purpose to examine the performance of the developed techniques. Furthermore, this will contribute to a better understanding of CTC and thus algebraic attacks on block ciphers.

Given the CTC (Courtois Toy Cipher) cryptosystem and a plaintext-ciphertext pair, we can construct an overdetermined algebraic system of equations in terms of the indeterminates representing key bits and certain intermediate quantities (see [57] for the construction of polynomials). Then the task is to solve the system for the key bits. The size of the system depends mainly on two parameters: the number B of simultaneous S-boxes and the number N of encryption rounds used. Throughout this thesis we denote a particular instance of CTC by CTC(B,N). The polynomial systems used for experiments are available at `http://apcocoa.org/polynomialsystems/`.

### Hidden Field Equations (HFE)

Hidden Fields Equations (HFE) [150] is a public key cryptosystem which was introduced at Eurocrypt in 1996 and proposed by J. Patarin following the idea of the Matsumoto and Imai system. HFE is also known as HFE trapdoor function. It is based on polynomials over finite fields $\mathbb{F}_q$ of different size to disguise the relationship between the private key and public key. HFE is in fact a family which consists of basic HFE and combinatorial versions of HFE. The HFE family of cryptosystems is based on the hardness of the problem of finding solutions to a system of multivariate quadratic equations (the so-called MQ problem) since it uses private affine transformations to hide the extension field and the private polynomials. Hidden Field Equations also have been used to construct digital signature schemes, e.g. Quartz and Sflash. The polynomial systems used for experiments are available at `http://apcocoa.org/polynomialsystems/`.

#### Small Scale AES

In [50], C. Cid et al. defined small scale variants of the AES. These variants inherit the design features of the AES and provide a suitable framework for comparing different cryptanalytic methods. In particular, they provide some preliminary results and insights when using off-the-shelf computational algebra techniques to solve the systems of equations arising from these small scale variants. Without going into details, let us recall the arguments of possible configurations of the small scale AES cryptosystem presented in [50]. By AES(n,r,c,e) we denote the system such that

- $n \in \{1, \ldots, 10\}$ is the number of (encryption) rounds,
- $r$ is the number of rows in the rectangular arrangement of the input,
- $c$ is the number of columns in the rectangular arrangement of the input,
- $e$ is the size (in bits) of a word.

The word size $e$ describes the field $\mathbb{F}_{2^e}$ over which the equations are defined. For instance, $e = 4$ corresponds to $\mathbb{F}_{16}$ and $e = 8$ to $\mathbb{F}_{256}$. If we choose the parameters $r = 4$, $c = 4$ and $w = 8$, we get a block size of $4 \cdot 4 \cdot 8 = 128$ bits, and small AES becomes equivalent to full AES. For more details and a way to express this cipher as a multivariate equation system over $\mathbb{F}_2$ we refer to [50, 106]. The polynomial systems used for experiments are available at `http://apcocoa.org/polynomialsystems/`.

### 2.3.3 NP-Completeness of Polynomial System Solving Over Finite Fields

As we saw, solving a system of polynomial equations is a quite general problem which can be used for signing and encrypting. Actually, solving a system of polynomial equations even over finite fields is an **NP**-complete problem. Recall that **NP** is the complexity class of decision problems that can be solved on a non-deterministic Turing machine in polynomial time. In other words, it is the class of decision problems that can be solved by a deterministic algorithm with running time bounded by $2^{f(x)}$, where $f(x)$ is some polynomial in $x$. A decision problem in **NP** is said to be **NP**-complete if every other problem in **NP** can be reduced to it in polynomial time. Also recall that all **NP**-complete problems are polynomially equivalent i.e. one **NP**-complete problem can be reduced to another **NP**-complete problem and such reductions are polynomial time algorithms.

Typically, completeness is a sign that the problem cannot be solved satisfactorily. Solving a problem that is **NP**-complete in polynomial time, would mean that all problems in **NP** are solvable in polynomial time, i.e. **P**=**NP**. Deciding if a random system of multivariate polynomial equations even over finite fields has a solution is **NP**-complete. In particular, the **NP**-completeness of deciding whether systems of quadratic equations have a solution over finite fields is proven in [94, 151, 79]. For a generalization of this result to any domain we refer to [151]. Recall that every finite domain is also a finite field. For further reading about the completeness we refer to [82] and [128] summarizing important results in this field.

**NP**-completeness is a definition for a class of problems that are hard to solve on average. Cryptosystems are usually disguised as such problems, like the knapsack problem. In general, being **NP**-complete does not imply that a cryptosystem using this problem is automatically secure. For a counterexample see cryptosystems using the knapsack problem. Although the knapsack problem is **NP**-complete [82], most of these cryptosystems were broken, see [137] for an overview. Therefore, decryption necessarily demands that there has to be a trapdoor. This makes structure inevitable and these instances with additional structure within the class of **NP**-complete problems might be easier to solve than they seem. In general, solving a system of multivariate equations over a finite field is **NP**-complete. This means that it will be very unlikely that every system is solvable in polynomial time. Therefore for the cryptosystems whose security rely on solving a system of polynomial equations over finite fields, there is strong empirical and theoretical evidence, e.g. [59, 58], that it is also hard on average (even with embedded trapdoor) and hence can be used as a basis for a secure cryptosystems. One objective of our study is to develop techniques to examine the security of these cryptosystems.

# Chapter 3

# Techniques From Linear Algebra

In this chapter we study techniques coming from linear algebra to solve algebraic systems of equations over finite fields. One of the most useful applications of Gröbner bases is to compute the solution set of a system of polynomial equations. Buchberger's Algorithm [34] was the first algorithm for computing Gröbner bases. Due to complexity issues of the standard Buchberger algorithm, several variants of this algorithm such as F4 [73], F5 [74] and XL (extended linearization) [59] have been proposed. Furthermore, several optimized versions of these variants make them even more powerful. Actually, these algorithms reduce a polynomial system solving problem to a linear algebra problem. The success achieved by these algorithms motivates further investigations in the field of linear algebra techniques for polynomial system solving.

We study some techniques from combinatorial optimization to transform a polynomial system solving problem into a linear algebra problem. In particular, we study the concept of transforming infeasibility proofs to large systems of linear equations. This enables us to use linear algebra techniques of last fifty years for polynomial system solving. We consider the possibility of accelerating these techniques by using sparse linear algebra. Furthermore, we highlight some new hybrid techniques that combine ideas studied by De Loera et al. [61, 64] for transforming combinatorial infeasibility proofs to large systems of linear equations and ideas of J. Ding et al. [67, 140, 142, 143] involving the concept of mutants. In each case, algorithms have been developed, implemented and their performance is examined. Finally, the efficiency of the developed techniques is studied using standard cryptographic examples.

## 3.1 Proving Combinatorial Infeasibility

In combinatorial optimization, systems of polynomial equations have been used to model combinatorial problems such as the matching problem, the graph coloring problem and the stable set problem (see [63] for an extensive list of references). Therefore, we can use polynomial systems to decide whether a graph, or other combinatorial structure, has a property captured by the polynomial system and its associated ideal. We call this the *combinatorial feasibility problem*.

In this section we review the concept of transforming infeasibility proofs to large systems of linear equations. Instead of formulating combinatorial problems by systems of polynomial equations, we focus on techniques for solving systems of polynomial equations that formulate combinatorial problems. In particular, we review an algorithm aimed at proving combinatorial infeasibility based on the observed low degree of Hilbert's Nullstellensatz certificates for polynomial systems arising in combinatorics, and based on fast large-scale linear algebra computations over a finite field. The idea is to use Hilbert's Nullstellensatz to generate a finite sequence of linear algebra systems, of increasing size, which will eventually become feasible if and only if the system of polynomial equations has no solution. Note that a combinatorial problem is feasible (e.g. a graph is 3-colorable, hamiltonian, etc.) if and only if the related system of polynomial equations has a solution. We conclude this section with some observations and remarks.

It was first mentioned by D. Bayer that the 3-colorability of graphs can be modeled via a system of polynomial equations [23]. Research efforts on encoding combinatorial properties by systems of polynomial equations includes colorings [9, 62, 72, 98, 133, 134, 135, 138], stable sets [62, 129, 133, 163], matchings [76], and flows [9, 138, 148]. N. Alon [7, 8, 9] first time used the term *polynomial method* to refer to the use of systems of non-linear polynomial equations for solving combinatorial problems. This well-known method, which Alon referred to as the polynomial method (see [7, 8]) recently regained strong interest. In [61, 63, 64] infeasibility of certain combinatorial problems is established using Hilbert's (complex) Nullstellensatz and the authors provide an algorithm NulLA to establish infeasibility by using a linear relaxation. Furthermore, in [64] J.A. De Loera et al. reviewed a methodology to solve systems of polynomial equations and inequalities. They discussed techniques that use the algebra of multivariate nonlinear polynomials with coefficients over a field to create large-scale linear algebra or semidefinite programming relaxations of many kinds of feasibility or optimization questions. Actually, feasibility and optimization problems translate, either directly or via

branching, into the problem of finding a solution of a system of equations and inequalities. They have also suggested a way to use the "Fredholm Alternative Theorem" and "Farkas' Lemma" [159] to manipulate systems of polynomial equations and inequalities for finding solutions or proving that they do not exist.

Let $K$ be an algebraically closed field, and let $f_1, \ldots, f_s \in K[x_1, \ldots, x_n]$ be non-zero polynomials. The monoid of terms is $\mathbb{T}^n = \{x_1^{\alpha_1} \ldots x_n^{\alpha_n} \mid \alpha_1, \ldots, \alpha_n \in \mathbb{N}\}$ and for $d \in \mathbb{N}$, we let $\mathbb{T}^n_{\leq d}$ denote the set of terms with total degree at most $d$. We consider the following system $\mathcal{S}$ of polynomial equations.

$$\mathcal{S}: \qquad \begin{cases} f_1(x_1, \ldots, x_n) = 0 \\ \qquad \vdots \\ f_s(x_1, \ldots, x_n) = 0 \end{cases}$$

We assume that the system $\mathcal{S}$ encodes some combinatorial problem and we study the set of solutions of the system of polynomial equations $\mathcal{S}$. The system $\mathcal{S}$ has either a (feasible) solution over $K$ or no solution. Note that a combinatorial problem is infeasible if a related system of polynomial equations has no solution. Since we are interested in establishing infeasibility proofs, we are more interested in the case when the system of polynomial equations $\mathcal{S}$ has no solution. To study the solution set of $\mathcal{S}$ we first need some ingredients which are as follows. Recall that $K$ is an algebraically closed field and for simplicity we denote the set of zeros $\mathcal{Z}_K(I)$ of an ideal $I \subseteq P$ in $K^n$ by $\mathcal{Z}(I)$.

**Theorem 3.1.1. (Weak Nullstellensatz)**
*Let $K$ be a field, and let $\overline{K}$ be the algebraic closure of $K$. Let $I$ be a proper ideal of $K[x_1, \ldots, x_n]$. Then $\mathcal{Z}_{\overline{K}}(I) \neq \emptyset$.*

*Proof.* See [120], Theorem 2.6.13. □

**Corollary 3.1.2.** *Let $K$ be an algebraically closed field and let $I$ be an ideal of $K[x_1, \ldots, x_n]$. Then the following conditions are equivalent.*

*a)* $\mathcal{Z}(I) = \emptyset$.

*b)* $1 \in I$ *i.e. there exist polynomials* $g_1, \ldots, g_s \in K[x_1, \ldots, x_n]$ *such that*

$$1 = \sum_{i=1}^{s} g_i f_i.$$

*Proof.* See [120], Corollary 2.6.14. □

We will use a slightly stronger form of the statement given in Corollary 3.1.2.*b* that is more useful for our purposes and can easily be deduced using the following lemma. This stronger form allows us to perform calculations over any field $K$ even if $K$ is not algebraically closed.

**Lemma 3.1.3.** *Let $\sigma$ be a term ordering, let $K' \subseteq K$ be a field extension, let $P' = K'[x_1, \ldots, x_n]$, let $I' \subseteq P'$ be an ideal of $P'$, and let $I$ be the ideal of $P = K[x_1, \ldots, x_n]$ generated by the elements of $I'$. Then a $\sigma$-Gröbner basis of $I'$ is also a $\sigma$-Gröbner basis of $I$. In particular, we have $\mathrm{LT}_\sigma\{I'\} = \mathrm{LT}_\sigma\{I\}$. Furthermore, the reduced $\sigma$-Gröbner basis of $I'$ is also the reduced $\sigma$-Gröbner basis of $I$.*

*Proof.* See [120], Lemma 2.4.16. □

**Definition 3.1.4.** Let $K$ be a field and let $\overline{K}$ be the algebraic closure of $K$. Let $f_1, \ldots, f_s \in K[x_1, \ldots, x_n]$ be such that the system of polynomial equations $f_1 = \cdots = f_s = 0$ has no solution in $\overline{K}^n$. Then there exist polynomials $g_1, \ldots, g_s \in K[x_1, \ldots, x_n]$ such that

$$1 = \sum_{i=1}^{s} g_i f_i. \tag{3.1}$$

The polynomial identity 3.1 is called a **Nullstellensatz certificate**. We say a Nullstellensatz certificate has **degree** $d$ if $\max\{\deg(g_i) \mid i \in \{1, \ldots, s\}\} = d$.

A natural question is to ask about the degree of a Nullstellensatz certificate. There are well-known upper bounds on the degrees of Nullstellensatz certificates (see Kollár [116] and the references therein). The upper bounds for the degrees of the polynomials $g_i$ in the Nullstellensatz certificates for general systems of polynomials are doubly-exponential in the number of input polynomials and their degrees. The upper bounds provided by Kollár [116] are known to be sharp for some specially constructed systems.

**Theorem 3.1.5.** *Let $K$ be an algebraically closed field and let $f_1, \ldots, f_s \in K[x_1, \ldots, x_n]$. Let $d = \max\{\deg(f_i) \mid i \in \{1, \ldots, s\}\}$. If $f_1, \ldots, f_s$ have no common zeros, then there exist polynomials $g_1, \ldots, g_s \in K[x_1, \ldots, x_n]$ such that $1 = \sum_{i=1}^{s} g_i f_i$, where $\deg(g_i f_i) \leq \max\{3, d\}^n$.*

*Proof.* This follows directly from Definitions 1.3 and 1.4 and Theorem 1.5 of [116]. □

Beyond the very general (and sharp) bounds of Kollár for Nullstellensatz certificates, we can still hope for subexponential bounds as suggested by J.A. De Loera et al. [61]. Actually, we can profit here from the following fundamental result by D. Lazard [125] that provides a linear bound.

**Lemma 3.1.6.** *Let $K$ be a field and let $f_1, \ldots, f_s \in K[x_0, \ldots, x_n]$ be homogeneous polynomials. Let $I = \langle f_1, \ldots, f_s \rangle$. Let $\deg(f_i) = d_i$ such that $d_1 \geq d_2 \geq \cdots \geq d_s \geq 1$ and $s \geq n + 1$. Then the following conditions are equivalent:*

*a) The $s$ projective hypersurfaces defined by $f_1, \ldots, f_s$ have no point in common over the algebraic closure of $K$ (in particular, they have no point in common at infinity).*

*b) The ideal $I$ contains a power of the maximal ideal $M = \langle x_0, \ldots, x_n \rangle$; namely, for some power $p$, $x_i^p \in I$ for all $x_i$.*

*c) $M^p \subset I$ with*

$$p = d_1 + d_2 + \cdots + d_{n+1} - n \leq (n+1)(\max\{d_i \mid i \in \{1, \ldots, n+1\}\} - 1) + 1.$$

*d) The map $\phi : (g_1, \ldots, g_s) \to \sum g_i f_i$ is surjective among all polynomials of degree $p$, when for all $i$, $g_i$ is a homogeneous polynomial of degree $p - d_i$.*

*Proof.* See [125], page 169. □

**Remark 3.1.7.** J.A. De Loera et al. [61] observed that the polynomial systems that encode combinatorial problems belong to the case given by Lemma 3.1.6 in the following sense. Consider the homogenization $\bar{f}_i$ of the polynomial $f_i \in K[x_1, \ldots, x_n]$, using an extra indeterminate $x_0$. If we find a "projective" Nullstellensatz certificate $x_0^p = \sum g_i \bar{f}_i$, we obtain the Nullstellensatz certificate $1 = \sum g_i' f_i$ by substituting $x_0 = 1$. Moreover, $\deg(g_i') \leq \deg(g_i)$. J.A. De Loera et al. summarized Lemma 3.1.6 for their use as the following corollary which also agrees with Brownawell [33], Proposition 9.

**Corollary 3.1.8.** *Let $K$ be an algebraically closed field and let $f_1, \ldots, f_s \in K[x_1, \ldots, x_n]$. Let $d = \max\{\deg(f_i) \mid i \in \{1, \ldots, s\}\}$. If $f_1, \ldots, f_s$ have no common zeros over $K$ and $f_1, \ldots, f_s$ have no common zeros at infinity, then there exist polynomials $g_1, \ldots, g_s \in K[x_1, \ldots, x_n]$ such that $1 = \sum_{i=1}^{s} g_i f_i$, where $\deg(g_i) \leq n(d-1)$.*

**Remark 3.1.9.** The above corollary says, the degree of a Nullstellensatz certificate for polynomial systems which encode combinatorial problems has a linear bound, which is a considerable improvement on the exponential bound predicted by Kollár. Although this linear bound is an improvement, it is still far from being computationally practical. However, in [61, 63] it is claimed that in practice the degree growth of polynomial systems for combinatorial problems is often very slow. From a computational point of view this sounds to be good news. Furthermore, for the Nullstellensatz certificates, the degrees of the polynomials $g_i$ cannot be larger than the known bounds. Thus we can design a finite (but potentially long) procedure to decide whether a system of polynomial equations has a feasible solution or no solution. Furthermore, we know bounds on the degrees of Nullstellensatz certificates for some concrete families of polynomial systems. For example, every Nullstellensatz certificate of a non-3-colorable graph has degree at least four and for every graph $\Gamma$ there exists a Nullstellensatz certificate of degree equal to the stability number of $\Gamma$, certifying that $\Gamma$ has no stable set of size greater than its stability number (see [63]). But it is still a challenge to derive degree bounds for other combinatorial problems.

The algorithm provided by J. De Loera et al. [61, 63] for establishing infeasibility certificates is called Nullstellensatz Linear Algebra (NulLA) Algorithm. The NulLA Algorithm takes as input a system of polynomial equations and outputs either a `YES` answer, if the system of polynomial equations has a solution, or a `NO` answer, along with a Nullstellensatz infeasibility certificate, if the system has no solution.

**Theorem 3.1.10. (Nullstellensatz Linear Algebra (NulLA) Algorithm)** *Let $K$ be a field and let $F = \{f_1, \ldots, f_s\} \subseteq K[x_1, \ldots, x_n]$ be a set of polynomials. Consider the following sequence of instructions.*

*1) Let $d_c = 1$ and let $D$ be a known upper bound for the degree of a Nullstellensatz certificate for $F$.*

*2) If $d_c > D$, return* `YES`*.*

*3) Let $\mu = |\mathbb{T}^n_{\leq d_c}|$, and $d = d_c + \max\{\deg(f_i) \mid i \in \{1, \ldots, s\}\}$. Let $E$ denote the equality $1 = \sum_{i=1}^{s} g_i f_i$, where $g_i = \sum_{j=1}^{\mu} c_{ij} t_j$, are polynomials of degree $d_c$ with unknown coefficients $c_{ij} \in K$ and terms $t_j \in \mathbb{T}^n_{\leq d_c}$.*

*4) For each $t \in \mathbb{T}^n_{\leq d}$, combine the terms in $E$, i.e. by comparing the coefficients of each term $t \in \mathbb{T}^n_{\leq d}$ on both sides of the equality $E$, extract a system of linear*

*equations $L$ with columns corresponding to the unknown coefficients $c_{ij}$ and rows corresponding to the terms in $\mathbb{T}^n_{\leq d}$.*

5) *If the linear system $L$ has no solution then increase $d_c$ by one and continue with step 2).*

6) *Replace the unknown coefficients in $E$ with the values of a solution of the linear system $L$ and return $E$ with a* `NO` *answer.*

*This is an algorithm which returns either a* `NO` *answer, if the system $f_1 = 0, \ldots, f_s = 0$ has a solution, or a* `YES` *answer, along with a Nullstellensatz infeasibility certificate, if the system $f_1 = 0, \ldots, f_s = 0$ has no solution.*

*Proof.* Consider the system of polynomial equations $f_1 = 0, \ldots, f_s = 0$. By Corollary 3.1.2 this system has no solution if and only if there exist polynomials $g_1, \ldots, g_s \in \overline{K}[x_1, \ldots, x_n]$ such that

$$1 = \sum_{i=1}^{s} g_i f_i. \tag{3.2}$$

Furthermore, by Lemma 3.1.3 we have $g_1, \ldots, g_s \in K[x_1, \ldots, x_n]$. Therefore the correctness follows from Corollary 3.1.2 and Lemma 3.1.3. To make sure that the process terminates even if the system $f_1 = 0, \ldots, f_s = 0$ has a solution, we use the known upper bound on the degree of Nullstellensatz certificate for $F$. The process is guaranteed to terminate because, if a Nullstellensatz certificate exists, we must find at least one set of certificate polynomials $g_i$ before reaching the known degree bound $D$.

The algorithm works as follows. A tentative degree $d_c$ on the polynomials $g_i$ for $i = 1, \ldots, s$ is fixed. Then step 3) writes an equality $E$. This equality results in a linear system $L$ in step 4). If the linear system $L$ has a solution then a certificate is constructed by replacing the indeterminate coefficients in $E$ with the values of a solution of the linear system. Otherwise, $d_c$ is increased by one. This process is continued until we have a linear system which has a solution or we exceed the degree bound $D$. In this way the process terminates after finitely many iterations. □

**Remark 3.1.11.** In the following, we collect some remarks about the NulLA Algorithm.

a) It is natural to ask about lower bounds on the degree of the Nullstellensatz certificates. Only little is known on this topic. Recently such a bound was given in [63] for those combinatorial problems where we need to decide whether

a given graph has an independent set of a given size. Recall that a stable set or independent set in a given graph $\Gamma$ is a subset of vertices such that no two vertices in the subset are adjacent. For polynomial systems coming from logic there has also been an effort to show degree growth in related polynomial systems (see [40, 103] and the references therein). Another question is to provide tighter, more realistic upper bounds for concrete systems of polynomials. It is a challenge to derive such bounds for any concrete family of polynomial systems.

b) Since we are interested in practical computational problems, it makes sense to explore refinements and variations that make NulLA robust and faster for concrete challenges. The main computational component of NulLA is to construct and solve linear systems for finding Nullstellensatz certificates of increasing degree. Furthermore, the size of the linear systems increases dramatically with the degree of the certificate. A big challenge is to improve NulLA in a way allowing it to use only the limited available memory and time resources for solving a combinatorial feasibility problem with as large number of equations and variables as possible. One of the strategies to improve the efficiency of NulLA could be to find better linear algebra techniques. This mainly reduces the time consumption. On the other hand, strategies improving the enlargement step of NulLA, constructing linear systems for finding Nullstellensatz certificates of increasing degree, will affect both time and memory consumption.

**Example 3.1.12.** Let $K = \mathbb{R}$, and let $F = \{f_1, f_2, f_3, f_4\} \subseteq K[x_1, x_2, x_3]$, be a set of polynomials, where $f_1 = x_1^2 - 1$, $f_2 = x_1 + x_2$, $f_3 = x_1 - x_3$ and $f_4 = x_2 - x_3$. Clearly the system $f_1 = f_2 = f_3 = f_4 = 0$ has no solution even over $\mathbb{C}$, and we will see that it has a Nullstellensatz certificate of degree one by following the steps of the NulLA algorithm.

1) Let $d_c = 1$.

3) Let $\mathbb{T}^3_{\leq d_c} = \{1, x_1, x_2, x_3\}$, $\mu = 4$ and $d = 3$. Write down the tentative certificate $E : 1 = \sum_{i=1}^{4} g_i f_i$, where $g_i = c_{i1}x_1 + c_{i2}x_2 + c_{i3}x_3 + c_{i3}$ for $i = 1, \ldots, 4$.

4) For each $t \in \mathbb{T}^3_{\leq d}$, combine the terms in $E$ containing $t$. We have
$1 = (c_{11}x_1^3 + c_{12}x_1^2x_2 + c_{13}x_1^2x_3) + (c_{13} + c_{14} + c_{31})x_1^2 + (c_{22} + c_{42})x_2^2 + (c_{33} + c_{43})x_3^2 + (c_{21} + c_{22} + c_{32} + c_{41})x_1x_2 + (c_{23} + c_{31} + c_{33} + c_{41})x_1x_3 + (c_{23} + c_{32} + c_{42} + c_{43})x_1x_3 + (c_{24} + c_{34} - c_{11})x_1 + (c_{24} + c_{44} - c_{12})x_2 + (c_{34} + c_{15} - c_{13})x_3 - c_{14}$.
By comparing coefficients of each term $t \in \mathbb{T}^3_{\leq d}$, on both sides of the equality $E$,

we have the following system of linear equations.
$L \ : \ c_{11} = 0, c_{12} = 0, \ldots, c_{13} + c_{21} + c_{31} = 0, c_{34} + c_{44} - c_{13} = 1, c_{31} = 0, -c_{14} = 0$

6) The linear system $L$ has a solution, therefore we construct the Nullstellensatz certificate from the solution of $L$ by replacing $c_{ij}$ with the solution values of $L$ as follows.
$$1 = (-1)(x_1^2 - 1) + \tfrac{1}{2}x_1(x_1 + x_2) - \tfrac{1}{2}x_1(x_1 - x_3) + \tfrac{1}{2}x_1(x_2 - x_3)$$

There could be two approaches to make NulLA faster. The first one is by decreasing the size of the linear system for a given degree, and the second one is by decreasing the degree of the Nullstellensatz certificate for infeasible polynomial systems. This significantly reduces the size of the largest linear system that we need to solve for proving infeasibility. Note that these approaches to reduce the degree of the Nullstellensatz certificates do not decrease the available upper bounds on the degree of the Nullstellensatz certificate required for proving feasibility, but they work only in particular instances. The systems of polynomial equations that encode combinatorial problems are very special. To exploit the special properties of these polynomial systems some optimizations are proposed by J. De Loera et al. in [61, 64]. To apply them to arbitrary polynomial systems one has to look for certain structures in the polynomials.

**Remark 3.1.13. (Optimizations of the NulLA Algorithm)**
In the following we have a short look on some optimizations proposed by J. De Loera et al. in [61, 64].

a) For some combinatorial problems such as 3-colorability, we can carry out calculations over finite fields, especially over $\mathbb{F}_2$, instead of relying on their algebraic closures. (see [61], Section 3). Finally, the degree of Nullstellensatz certificates necessary to prove infeasibility can be lower over $\mathbb{F}_2$ than over $\mathbb{R}$. For example, over $\mathbb{R}$, every odd-wheel has a minimum non-3-colorability certificate of degree six [63]. However, over $\mathbb{F}_2$, every odd-wheel has a Nullstellensatz certificate of degree three. Therefore, not only are the mathematical computations more efficient over $\mathbb{F}_2$ as compared to $\mathbb{R}$, but the algebraic properties of the certificates themselves are sometimes more favorable for computations as well.

b) By appending certain valid but redundant polynomial equations to the system $\mathcal{S}$, we can decrease the degree of the Nullstellensatz certificate necessary to prove infeasibility. Let $I$ be the ideal generated by the polynomials $f_1, \ldots, f_s$. A valid but redundant polynomial equation is any polynomial equation $g = 0$ that is

true for all the zeros of the polynomial system $\mathcal{S}$ i.e. $g \in \sqrt{I}$. A redundant polynomial equation appended to the system $\mathcal{S}$, is refereed as a *degree-cutter*. Note that appending an equation could never increase the necessary degree of a Nullstellensatz certificate.

c) The size of the linear system in step 4) of the algorithm in Theorem 3.1.10 can also be reduced by using a group action on the variables, e.g., using symmetries or automorphisms in a graph. Suppose we have a finite permutation group $\mathfrak{S}_n$ acting on the variables $x_1, \ldots, x_n$. The group $\mathfrak{S}_n$ induces an action on the set of terms of degree $d$ with variables $x_1, \ldots, x_n$. Such kind of optimizations are very special and they are only applicable to polynomial systems which encode combinatorial problems. We refer to [61], Section 3 for an explanation how symmetries can be used to reduce the size of the linear system.

d) Another approach is to decrease the minimal degree of the Nullstellensatz certificate by using the *Alternative Nullstellensatz*. By Alternative Nullstellensatz we mean the following. Let $K$ be an algebraically closed field and let $f_1, \ldots, f_s \in K[x_1, \ldots, x_n]$ be polynomials. The system of polynomial equations $F : f_1 = 0, \ldots, f_s = 0$ has no solution in $K^n$ if and only if there exist polynomials $g_1, \ldots, g_s \in K[x_1, \ldots, x_n]$ and $h \in K[x_1, \ldots, x_n]$ such that

$$h = \sum_{i=1}^{s} g_i f_i, \tag{3.3}$$

and the system $f_1 = 0, \ldots, f_s = 0$ and $h = 0$ has no solution. The Nullstellensatz certificate of Definition 3.1.4 is a special case of this Alternative Nullstellensatz 3.3, where $h = 1$. In practice, some times the minimal degree of the Alternative Nullstellensatz certificate is smaller than the minimal degree of the ordinary Nullstellensatz certificate. Some more ideas to improve NulLA involve branching, deleting equations and exploiting linear dependencies (see [61], Section 3).

In [61], the process of iterations of NulLA is claimed to terminate correctly due to the following argument.

> *The process is guaranteed to terminate because, for a Nullstellensatz certificate to exist, the degrees of the certificate polynomials $g_i$ cannot be more than known bounds.*

This does not seem to be exactly the case because of the following example. A better argument could be the following. The process is guaranteed to terminate because, if a Nullstellensatz certificate exists, we must find at least one set of certificate polynomials $g_i$ before reaching the known degree bound $D$.

**Example 3.1.14.** Let $K = \mathbb{R}$, and let $F = \{f_1, f_2, f_3\} \subseteq K[x_1, x_2]$, be a set of polynomials, where $f_1 = x_1^2 - 1$, $f_2 = x_1 + x_2$ and $f_3 = x_1 - x_2$. Clearly the system $f_1 = f_2 = f_3 = 0$ has no solution even over $\mathbb{C}$. The upper bound given by Lemma 3.1.8 is $n(d-1) = 2$, where $n = 2$ is the number of indeterminates and $d = \max(\{\deg(f_1), \deg(f_2), \deg(f_3)\}) = 2$. Now consider the following certificate for $F$.

$$\begin{aligned} 1 \;=\; & (\tfrac{45203}{2}x_1^2 + 21066x_1x_2 + \tfrac{54959}{2}x_2^2 + 11611x_1 - 51184x_2 - 1)(x_1^2 - 1) + (\tfrac{247}{2}x_1^3 \\ & -\tfrac{67921}{2}x_1^2x_2 - \tfrac{78495}{2}x_1x_2^2 + 37511x_2^3 - 63726x_1^2 + 87094x_1x_2 - \tfrac{7163}{2}x_2^2 + 37811x_1 \\ & -2237x_2 - \tfrac{39573}{2})(x_1+x_2) + (-22725x_1^3 - 9954x_1^2x_2 + \tfrac{71549}{2}x_1x_2^2 + 37511x_2^3 + 52115x_1^2 \\ & +79931x_1x_2 - \tfrac{7163}{2}x_2^2 - \tfrac{30417}{2}x_1 - \tfrac{59433}{2}x_2 + \tfrac{62795}{2})(x_1 - x_2). \end{aligned}$$

The degree of the above certificate is 3 which is larger than 2, the bound given by Lemma 3.1.8. Similarly we can also establish certificates of degree larger than 3. Note that the system $F$ also has a certificate of degree less than the bound given by Lemma 3.1.8, which is as follows.

$$1 = (-1)(x_1^2 - 1) + (\frac{1}{2}x_1)(x_1 + x_2) + (\frac{1}{2}x_1)(x_1 + x_2)$$

The following remark gives us a more clear view of the aforementioned fact.

**Remark 3.1.15.** Assume that the system $\mathcal{S}$ has no solution and $D$ is the upper bound given by Lemma 3.1.5 or Lemma 3.1.8. Then there must exist at least one Nullstellensatz certificate of degree less than or equal to $D$. But there may exist certificates of degree greater than $D$. And if the system $\mathcal{S}$ has a solution then the process will terminate exactly at degree $D$. This means that the bound $D$ ensures that if there does not exist a certificate of degree less than or equal to $D$ then there does not exists a certificate of degree greater than $D$. In view of this fact the termination is guaranteed due to two reasons. Firstly, if the system $\mathcal{S}$ has no solution then we must find a linear system which has a solution. Secondly, if the system $\mathcal{S}$ has a solution then we stop exactly at degree $D$. See the proof of Theorem 3.1.10 for more details about correctness and termination of NulLA.

We end this section with a final remark.

**Remark 3.1.16.** In [61], it is stated that the iterations of the steps 2)−4) of the NulLA algorithm determine the running time of NulLA. But in our opinion the running time depends on the size of the linear systems in step 4). A smaller system of polynomial equations (having small sized linear systems) may need more iterations than a larger system of polynomial equations (having very large sized linear systems). We will see such examples in Section 3.3. Therefore, it is the size of the linear systems that determines the running time of NulLA. Furthermore, the difficulty of solving a linear system also matters. Some very well structured and sparse linear systems could be much easier to solve (see Remark 3.1.13.c).

## 3.2 The LA Algorithm

In this section we investigate an algorithm aimed at solving systems of polynomial equations over finite fields. This algorithm is based on the ideas reviewed in Section 3.1 and on fast large-scale (sparse) linear algebra computations over a finite field. Although such techniques have been known in combinatorial optimization, they have not been used for polynomial system solving over finite fields. The key issue that we investigate here is the use of techniques from Section 3.1 for solving over finite fields. We are particularly interested in whether this can be accomplished in practice for large systems of polynomial equations over finite fields.

In Section 3.1 we saw a method that generates a finite sequence of linear algebra systems to decide whether a system of polynomial equations has a solution or no solution. In particular, we saw an algorithm called the Nullstellensatz Linear Algebra (NulLA) algorithm that takes as input a system of polynomial equations and outputs either a `YES` answer, if the system of polynomial equations has a solution, or a `NO` answer, along with a Nullstellensatz infeasibility certificate, if the system has no solution. Building on this foundation we study solving systems of polynomial equations over finite fields, especially over $\mathbb{F}_2$. We explicitly formulate and explain the Linear Algebra (LA) Algorithm which is an algorithm for solving systems of polynomial equations over finite fields. The calculations reduce to (sparse) matrix manipulations, mostly rank computations. The techniques we use are a specialization of prior techniques from computational algebra (see [145, 59, 111]). It turns out this technique is particularly effective when the number of solutions is finite, when the under laying field is a finite field, or when the system is very well structured. Throughout this section, we

use the following notation and terminology unless mentioned otherwise.

Let $p$ be a prime number, let $q = p^e$ for some $e > 0$, let $K = \mathbb{F}_q$ be the finite field with $q$ elements, and let $f_1, \ldots, f_\ell \in K[x_1, \ldots, x_n]$ be non-zero polynomials. The monoid of terms is $\mathbb{T}^n = \{x_1^{\alpha_1} \ldots x_n^{\alpha_n} \mid \alpha_1, \ldots, \alpha_n \in \mathbb{N}\}$ and for $d \in \mathbb{N}$, we let $\mathbb{T}^n_{\leq d}$ denote the set of terms with total degree at most $d$. Let $I$ be the ideal generated by the polynomials $f_1, \ldots, f_\ell$. We are interested in finding $K$-rational solutions of a system of polynomial equations.

$$\begin{aligned} f_1(x_1, \ldots, x_n) &= 0 \\ &\vdots \\ f_\ell(x_1, \ldots, x_n) &= 0 \end{aligned}$$

As we saw in Section 2.2, it is safe to include equations of the field ideal while solving the system of polynomial equations $f_1 = 0, \ldots, f_\ell = 0$, since we are looking for $K$-rational solutions. From now on we let the set $F = \{f_1, \ldots, f_\ell, f_{\ell+1}, \ldots, f_m\}$ including the polynomials $f_1, \ldots, f_\ell$ and the polynomials $f_{\ell+1} = x_1^q - x_1, \ldots, f_m = x_n^q - x_n$, where $x_i^q - x_i$, for $i = 1, \ldots, n$ are the elements of the field ideal. *In particular, we assume that the system of polynomial equations $f_1 = 0, \ldots, f_m = 0$ has a unique $K$-rational solution*, say $(a_1, \ldots, a_n)$. So the ideal $I = \langle F \rangle$ is a zero-dimensional radical ideal (see Section 2.2). With this terminology in mind, we have the following lemma.

**Lemma 3.2.1.** *Let $I = \langle F \rangle$. Then there exist elements $a_1, \ldots, a_n$ in $K$ such that*

$$I = \langle x_1 - a_1, \ldots, x_n - a_n \rangle.$$

*Proof.* The claim follows from Theorem 2.2.4 and [120], Proposition 2.6.11. □

The following definition plays an important role to formulate the LA Algorithm.

**Definition 3.2.2.** Let $x_i \in \{x_1, \ldots, x_n\}$.

a) An element $k \in K$ is called an $i^{\text{th}}$ **solution coordinate** if there are polynomials $g_j \in K[x_1, \ldots, x_n]$ such that

$$x_i - k = \sum_{j=1}^{m} g_j f_j, \tag{3.4}$$

where $f_j \in F$. The identity 3.4 is called a **certificate** for the $i^{\text{th}}$ solution coordinate $k$.

b) We say a certificate has **degree** $d$ if $d = \max\{\deg(g_j) \mid j = 1, \ldots, m\}$.

**Remark 3.2.3.** In the above setting, the ideal $I$ is of the form $\langle x_1 - a_1, \ldots, x_n - a_n\rangle$. Then every reduced Gröbner basis of $I$ is of the form $\{x_1 - a_1, \ldots, x_n - a_n\}$. So to check whether an element $k \in K$ is an $i^{\text{th}}$ solution coordinate is equivalent to check the membership $x_i - k \in I$. The idea is then to find the certificate, i.e. the polynomials $g_j$ for $x_i$. Such a certificate can be found by generating a sequence of linear algebra systems of increasing size. This sequence eventually produces a linear system which has a solution.

The ideas described above provide us with the following algorithm which we call the Linear Algebra Algorithm (LA Algorithm).

**Theorem 3.2.4. (The LA Algorithm)**
*Let $F = \{f_1, \ldots, f_m\} \subseteq K[x_1, \ldots, x_n]$ be non-zero polynomials containing the elements of the field ideal such that the system of polynomial equations $f_1 = 0, \ldots, f_m = 0$ has a unique solution $(a_1, \ldots, a_n) \in K^n$. Consider the following sequence of instructions.*

*1) Let $S = \emptyset$, $X = \{x_1, \ldots, x_n\}$, and let $H = F$.*

*2) If $X = \emptyset$, return $S$. Otherwise, choose an indeterminate $x_s \in X$ and delete it from $X$. Let $d_c = 0$, $Q = K$ and $r = |H|$.*

*3) If $Q \neq \emptyset$, then choose an element $k \in Q$ and delete it from $Q$. Otherwise, increase $d_c$ by one, let $Q = K$ and choose an element $k \in Q$ and delete it from $Q$.*

*4) Let $\nu = |\mathbb{T}^n_{\leq d_c}|$, and $d = d_c + \max\{\deg(h) \mid h \in H\}$. Let $E_k$ denote the equality $x_s - k = \sum_{i=1}^{r} g_i h_i$, with $h_i \in H$ and $g_i = \sum_{j=1}^{\nu} c_{ij} t_j$, where $c_{ij} \in K$ are unknown coefficients and $t_j \in \mathbb{T}^n_{\leq d_c}$.*

*5) For each $t \in \mathbb{T}^n_{\leq d}$, combine the terms in $E_k$, i.e. by comparing the coefficients of each term $t \in \mathbb{T}^n_{\leq d}$ on both sides of the equality $E_k$, extract a system of linear equations $L_k$ with columns corresponding to the unknown coefficients $c_{ij}$ and rows corresponding to the terms in $\mathbb{T}^n_{\leq d}$.*

*6) Solve the linear system $L_k$.*

*7) If the linear system $L_k$ has no solution then continue with step 3).*

*8) Append the corresponding $k$ to $S$, substitute $x_s = k$ in $H$. Then continue with step 2), applied to polynomials in a smaller ring.*

*This is an algorithm which returns the unique solution* $(a_1, \ldots, a_n)$ *of the system of polynomial equations* $f_1 = 0, \ldots, f_m = 0$.

*Proof.* Let $I = \langle F \rangle$ be the ideal generated by $F$. The ideal $I$ is a zero-dimensional radical ideal having a unique solution $(a_1, \ldots, a_n)$ which is rational over $K$. Therefore, Hilbert's Nullstellensatz implies that the vanishing ideal of the point $(a_1, \ldots, a_n)$ in $K^n$ is $I = \langle x_1 - a_1, \ldots, x_n - a_n \rangle$. Thus, for each $x_s \in X$ there exists a set of polynomials $g_i \in K[x_1, \ldots, x_n]$, $i = 1, \ldots, m$ such that

$$x_s - a_s = \sum_{i=1}^{m} g_i f_i, \tag{3.5}$$

where $a_s \in K$ is the corresponding solution coordinate. To check the correctness, it suffices to observe the following. From steps 4)−5), the equation 3.5 holds if and only if there exists a linear system which has a solution. Since Hilbert's Nullstellensatz implies that 3.5 holds, there exists a linear system which has a solution. The only thing left to see is how to find such a linear system. For this we generate the ideal gradually degree by degree in the following way. A tentative degree $d_c$ on the polynomials $g_i$ for $i = 1, \ldots, m$ is fixed. Then step 4) writes an equality $E_k$ as a tentative certificate. This equality results in a linear system $L_k$ in step 5). If the linear system $L_k$ has no solution and there are no more elements in $Q$ then $d_c$ is increased by one. This process is continued until we find a linear system which has a solution. Now replacing the indeterminate coefficients in $E_k$ with the values of a solution of the linear system provides the corresponding certificate. Thus, the corresponding $k \in Q$ is a solution coordinate and the process terminates after finitely many iterations. □

**Remark 3.2.5.** In step 2) of the algorithm, one can choose variables randomly or depending on which variable to find first. The process of solving linear systems in step 3) takes advantage of fast linear algebra techniques. The size of linear systems determines the running time of this algorithm. The LA Algorithm has some useful features for solving systems of polynomial equations over finite fields. It can be parallelized very easily due to two reasons. Firstly, the system can be solved for an individual indeterminate. Secondly, the process of solving linear systems can be parallelized. Furthermore, it takes advantage of fast linear algebra techniques, possible sparseness and the structure of the polynomial system $F$ as well as the linear system.

There could be two approaches to make the LA Algorithm faster. The first one is

by decreasing the size of the linear system for a given degree, and the second one is by decreasing the degree of the certificate for an $i^{\text{th}}$ solution coordinate. Thus significantly reducing the size of the largest linear system that we need to solve to find an $i^{\text{th}}$ solution coordinate. Note that these approaches to reduce the degree of a certificate do not decrease the available upper bounds on the degree of the Nullstellensatz certificate required to find an $i^{\text{th}}$ solution coordinate, but they work only in particular instances. Also to apply such approaches to arbitrary polynomial systems one has to look for certain structures in the polynomials.

A closer look at this algorithm shows that different variants and optimizations of the LA Algorithm are possible. Some of the most effective ones will be discussed in Sections 3.4 and 3.5. Here we limit ourselves to pointing out some obvious opportunities for improvement.

**Remark 3.2.6. (First Optimizations of the LA Algorithm)**

a) **Deleting Equations:** The LA Algorithm spends most of its time on solving linear systems. One way of reducing the size of these linear systems is to remove all $f_i \in F$ for which there exists $h_1, \ldots, h_{i-1}, h_{i+1}, \ldots, h_h \in K[x_1, \ldots, x_n]$, such that $f_i = \Sigma_{j \neq i} h_j f_j$ and $\deg(h_j f_j) \leq \deg(f_i)$ for all $j \neq i$. This means that $f_i$ is in the ideal generated by $F \setminus \{f_i\}$. Thus, $f_i$ is a redundant polynomial. Since replacing $f_i$ with $\sum_{j \neq i} h_j f_j$ in a given certificate gives another certificate of the same degree but without $f_i$, removing $f_i$ can never increase the degree of a certificate.

b) **Exploiting Linear Dependencies:** In step 5) of the theorem, there are often many columns in the coefficient matrix of the linear system $L_k$ that are linear combinations of other columns. If we could avoid creating these columns then solving the linear system $L_k$ would be more efficient. Actually, each column of this matrix corresponds to the polynomial $t'h_i$ for some term $t' \in \mathbb{T}^n_{\leq d_c}$ and some polynomial $h_i \in H$ where $\deg(t'h_i) \leq d$. The column $t'h_i$ is thus a linear combination of the other columns of the matrix if there exists $\beta_1, \ldots, \beta_l \in K[x_1, \ldots, x_n]$ such that $t'h_i = \sum_{j=1}^{l} \beta_j h_j$ where $\deg(\beta_j h_j) \leq d$ and the term $t'$ does not appear in the polynomial $\beta_i$.

To understand Theorem 3.2.4 better, we now apply the LA Algorithm in a concrete case.

**Example 3.2.7.** Let $n = 2$, let $q = 2$, and let $F = \{f_1, f_2, f_3, f_4\} \subseteq \mathbb{F}_2[x, y]$, be the following set of polynomials $f_1 = xy + y + 1$, $f_2 = xy + x$, $f_3 = x^2 - x$ and $f_4 = y^2 - y$. The system of polynomial equations $f_1 = f_2 = f_3 = f_4 = 0$ has a unique solution over $\mathbb{F}_2$. We are looking for the unique solution and follow the steps of the LA Algorithm.

1) Let $S = \emptyset$, $X = \{x, y\}$, and $H = F$.

2) Choose $x \in X$ and set $X = \{y\}$. Let $d_c = 0$, $Q = \{0, 1\}$ and $r = 4$.

3) Choose $0 \in Q$ and set $Q = \{1\}$.

4) Let $d = 2$, $\mathbb{T}^2_{\leq d_c} = \{1\}$, and $\nu = 1$. Write down the equality $E_0 : x = \sum_{i=1}^{4} g_i h_i$ where $h_i = c_{i1}$.

5) The corresponding linear system is $L_0 = \{c_{11} + c_{21} = 0, c_{11} + c_{41} = 0, c_{11} = 0, c_{21} + c_{31} = 1, c_{31} = 0, c_{41} = 0\}$

6) Solve the linear system $L_0$.

7) The linear system $L_0$ has no solution therefore return to step 3).

3) Choose $1 \in Q$ and set $Q = \emptyset$.

4) Let $d = 2$, $\mathbb{T}^2_{\leq d_c} = \{1\}$, and $\nu = 1$. Write down the equality $E_1 : x - 1 = \sum_{i=1}^{4} g_i h_i$ where $h_i = c_{i1}$.

5) The corresponding linear system is $L_1 = \{c_{11} + c_{21} = 0, c_{11} + c_{41} = 0, c_{11} = 1, c_{21} + c_{31} = 1, c_{31} = 0, c_{41} = 0\}$.

6) The linear system $L_1$ has no solution.

7) The linear system $L_1$ has no solution therefore return to step 3).

3) Set $d_c = 1$ and $Q = \{0, 1\}$. Choose $0 \in Q$ and set $Q = \{1\}$.

4) Let $d = 3$, $\mathbb{T}^2_{\leq d_c} = \{x, y, 1\}$ and $\nu = 3$. Write down the equality $E_0 : x = \sum_{i=1}^{4} g_i h_i$ where $h_i = c_{i1} + c_{i2}x + c_{i3}y$.

5) The corresponding linear system is $L_0 = \{c_{12} + c_{22} + c_{33} = 0, c_{13} + c_{23} + c_{42} = 0, c_{11} + c_{12} + c_{21} + c_{23} + c_{33} + c_{42} = 0, c_{13} + c_{43} + c_{41} = 0, c_{22} + c_{31} + c_{32} = 0, c_{32} = 0, c_{43} = 0, c_{11} + c_{13} + c_{41} = 0, c_{12} + c_{21} + c_{31} = 1, c_{11} = 0\}$.

6) The linear system $L_0$ has a solution. Therefore we found the first solution coordinate which gives the value of $x$.

8) Let $S = \{0\}$. Substitute $x = 0$ in all the polynomials of $H$ which gives $H = \{y + 1, y^2 - y\}$. Then return to step 2).

2) Choose $y \in X$ and set $X = \emptyset$. Let $d_s = 0$ and $r = 2$.

3) Choose $1 \in Q$ and set $Q = \{0\}$.

4) Let $d = 2$, $\mathbb{T}^2_{\leq d_c} = \{1\}$, and $\nu = 1$. Write down the equality $E_1 : x = \sum_{i=1}^{2} h_i g_i$ where $h_i = c_{i1}$.

5) The corresponding linear system is $L_1 = \{c_{11} + c_{21} = 1, c_{11} = 1, c_{21} = 0\}$

6) The linear system $L_1$ has a solution.

8) Let $S = \{0, 1\}$, and let $H = \emptyset$. Then return to step 2).

2) Since $X = \emptyset$, we return the solution $\{0, 1\}$.

**Remark 3.2.8. (The Degree Bounds of the Nullstellensatz Certificates)**
We have the following observations on the degree bound of a Nullstellensatz certificate.

a) A natural question could be to ask about the degree of the Nullstellensatz certificates for finding an $i^{\text{th}}$ solution coordinate. A first answer to this question is straightforward. The well-known upper bounds on the degrees of Nullstellensatz certificates discussed in Section 3.1 (see Lemma 3.1.5) are also valid in our case. But these upper bounds for the degrees of the $g_i$ in the Nullstellensatz certificates for general systems of polynomials are doubly-exponential in the number of input polynomials and their degree. Lemma 3.1.8 provides a linear bound which is a considerable improvement but it is only valid for those polynomial systems which encode combinatorial problems. Since we are working over (small) finite fields, we can hope for less extreme (e.g., exponential or subexponential) bounds. The solution of linear systems of equations with polynomial coefficients is an important topic that has received attention by algebraic geometers, computer algebraists and cryptographers. The techniques we use are a specialization of prior techniques from computational algebra like Gröbner bases and border bases (see [145, 59, 111, 126, 120]). There have been several attempts on finding bounds for Gröbner bases algorithms. Recently, some bounds for the generic complexity of

Gröbner bases algorithms over the finite field $\mathbb{F}_2$ for *semi-regular* overdetermined systems were provided in [20]. Since the techniques we use are a specialization of Gröbner bases, the bounds for our techniques seem equal to the bounds in [20]. But it is still a challenge to settle specially for any concrete family of polynomial systems.

b) In combinatorial optimization, we know bounds on the degrees of the Nullstellensatz certificates for some concrete families of polynomial systems. For example, every Nullstellensatz certificate of a non-3-colorable graph has degree at least four and for every graph $\Gamma$ there exists a Nullstellensatz certificate of degree equal to the stability number of $\Gamma$, certifying that $\Gamma$ has no stable set of size greater than its stability number (see [63]). Recall that a *stable set* in a given graph $\Gamma$ is a subset of vertices such that no two vertices in the subset are adjacent and the size of the largest stable set in $\Gamma$ is called the stability number of $\Gamma$. But it is still a challenge to settle for other combinatorial problems. In a similar way we can also ask about upper bounds on the degrees of the Nullstellensatz certificates for finding an $i^{\text{th}}$ solution coordinate for some concrete families of systems of polynomial equations over finite fields. Again it is still a challenge to settle. Note that combinatorial problems have very special structures on them which support such kind of bounds. The polynomial systems coming from other areas may not have such special properties.

**Remark 3.2.9.** As described, the LA Algorithm reduces a polynomial system solving problem to a linear algebra problem. This enables us to use linear algebra techniques of last fifty years for polynomial system solving. `M4RI` is a library for fast arithmetic with dense matrices over $\mathbb{F}_2$ and it has good performances. For general finite fields the best linear algebra packages are `ATLAS`, `LinBox`, `FFLAS-FFPACK` and `Sage` which are very efficient for dense linear algebra, but not tuned for dealing special structures in matrices. In [75], J.-C. Faugère and S. Lachartre presented, a linear algebra package for computing Gaussian elimination of Gröbner bases matrices. The package works for any finite field and contains specific algorithms to compute Gaußian elimination as well as specific internal representation of matrices (sparse triangular blocks, sparse rectangular blocks and hybrid rectangular blocks). For matrices coming from Gröbner bases applications this package seems to be the fastest one. In the following we discuss some more methods available for linear system solving.

- **Sparse Linear Algebra:** The systems generated by the LA Algorithm are obviously sparse. A respected textbook on sparse matrices [70] remarks that in not using matrix algorithms more tailored for the situation "you would just be pushing milliards of zeros around". Moving around gigabytes full of zeros not only slows down the computation directly, but increases the amount of memory required. Instead of using naïve Gaußian elimination for sparse matrices, a better procedure is to find block structures with graph-coloring analysis (see [70]). The elimination cost is then dominated by the elimination cost for the largest block. Since we are working over finite fields, Lanczos, Conjugate Gradient (GC), or Block Wiedemann algorithms [55] could be a good choice for solving linear systems. The Wiedemann algorithm can be used to find vectors in the null space of a matrix but to solve a linear system one can simply make a "dummy variable", and replace the constant 1 with this dummy variable. Solutions with the dummy variable equal to one are valid solutions. Lanczos, Conjugate Gradient and Wiedmann methods all have comparable speeds. The Wiedemann algorithm looks slower but more reliable (see [38, 123, 172]). But there are also some reservations on these methods. Lanczos (or Conjugate Gradient) is known to terminate sometimes incorrectly over a finite field. The Wiedemann algorithm is not known to terminate always correctly for non-square matrices. Proper operating conditions are not fully understood. However, such methods are used in practice.

- **Structured Gaußian Elimination:** This algorithm is also called Pomerance-Smith Algorithm or the Created Catastrophes Algorithm but neither of these names is very descriptive (see [123, 153]). This method looks more convincing for sparse matrices. But to apply this method the matrix should also have the right structure. Mainly, this algorithm is used during factoring methods like the Quadratic Sieve and those matrices are over $\mathbb{F}_2$. But it can also work over arbitrary finite fields with small adjustments. The use of this method for linear systems other than those coming from integer factorization problems is not known. Therefore, it could be interesting to see whether this method is applicable for linear systems coming from the LA Algorithm. We present more details and our observations along with computational evidences in Section 3.3.

## 3.3 Experimental Results for the LA Algorithm

In this section we report on some experiments with the LA Algorithm. First we try to see what can be achieved using a straightforward, non-optimized implementation of the LA Algorithm. Then we present some ideas for exploiting possible sparsity and the structure of a polynomial system. Moreover, we compare some of the timings we obtained to the straightforward Gröbner basis approach. Since the LA Algorithm reduces a polynomial system solving problem to a linear algebra problem, it highly depends on methods from linear algebra used for rank computations. We also report on experimental results using different methods and implementations of linear algebra.

A linear system can be viewed as a matrix. Therefore, we frequently switch between matrices and linear systems to discuss experimental results. As described, the LA Algorithm reduces a polynomial system solving problem to a linear algebra problem. This brings to bear the full artillery of fifty years of linear algebra research on the difficulty of the problem. Since the calculations reduce mostly to rank computations, we focus on calculating echelon forms, instead of solving linear systems for values of variables. In particular, we use the following libraries and some self implemented codes for calculating echelon forms.

- **LinBox:** The C++ library `LinBox` available as ApCoCoA [12] package `linbox`. Mostly we use this library to calculate an echelon form of a relatively dense matrix.

- **Echelon Form (EF):** A self implemented code for calculating an echelon form of a dense matrix over $\mathbb{F}_2$ using "Naïve Gaußian Elimination". Consider a matrix $\mathcal{M}$. At iteration $i$, the $i-1$ columns at the left have been processed. Now in column $i$, one must find a 1 at position $\mathcal{M}_{ii}$ or swap one into place, using row swaps. Thus the pivoting strategy could be said to be to ensure a non-zero entry at $\mathcal{M}_{ii}$. This implementation can be found in the ApCoCoA [12] package `linalg`.

- **Sparse Echelon Form (SEF):** If the matrix is sparse then we encode the matrix by considering the positions of non-zero elements. Then we apply naïve Gaußian elimination. For sparse matrices over finite fields, at iteration $i$, the obvious approach is to take the lowest weight row that happens to have a non-zero element in column $i$. Recall that the weight of a row is the number of non-zero entries in it. We call this "Naïve Sparse Gaußian Elimination". At iteration $i$, after choosing a lowest weight row $r$ one can scan through the row to see which

column $c$ has lowest weight, but $\mathcal{M}_{rc}$ non-zero. Then pivot by swapping column $i$ and column $c$ as well as row $r$ and row $i$. Obviously, choosing the lowest weight row and then the lowest weight column can reduce the fill-in. We have observed that we do not need to choose the lowest weight column for matrices which come from the LA Algorithm. Thus we can avoid some unnecessary computations. Actually, the idea is to preprocess a matrix to arrange it in a particular form before computing SEF. The SEF algorithm is implemented in collaboration with X. Xiu and is available in the ApCoCoA [12] package `slinalg`.

- **Structured Gaußian Elimination (SGE):** An implementation of the method known as "structured Gaußian elimination" [123, 153] to calculate an echelon form of a matrix. The main idea of structured Gaußian elimination is to work on preserving sparsity of light part by declaring some columns (with largest weight) as heavy. The set of heavy columns is allowed to grow as the algorithm progresses. The variant of this algorithm that we have implemented is composed of the following four steps.

  1) Delete all columns that have a single non-zero entry and the rows in which those columns have non-zero coefficients.
  2) Declare some additional light columns to be heavy by choosing the heaviest ones.
  3) Delete some of the rows, selecting those which have the largest number of non-zero elements in the light columns.
  4) For any row which has only a single non-zero entry equal to 1 in the light column, subtract appropriate multiples of that row from all other rows that have non-zero coefficients on that column so as to make those coefficients zero.

  Finally, naïve Gaußian elimination is applied on the dense part of the matrix. For more details we refer to [123]. We focus on calculating an echelon form using structured Gaußian elimination. It is implemented in collaboration with X. Xiu and is available in the ApCoCoA [12] package `slinalg`.

The cryptosystems considered to construct algebraic systems of equations are HFE (Hidden Field Equations) and CTC (Courtois Toy Cipher). For more details about these cryptosystems and related algebraic systems of equations see Section 2.3. For the cryptosystems under consideration, we used the ApCoCoA implementations by

J. Limbeck (see [131]). The systems used for experiments are provided in the CD attached to the thesis. Finally, note that the only time consuming step in the LA Algorithm is solving a linear system. Throughout this section we consider the timings only for calculating echelon forms. The timings for forming certificates and extracting linear systems are ignored, since they were not implemented efficiently and should be seen as a preprocessing step. Throughout this section, time is given in seconds unless mentioned otherwise. All timings were obtained on a computer with a 2.1 GHz AMD Opteron 6172 processor and 64GB RAM. The implementation of the LA Algorithm is also available online as a part of the ApCoCoA [12] package `charP`. For more details about implementation see Appendix B.

### 3.3.1 Experimental Results for HFE

Consider algebraic systems of equations constructed from the HFE cryptosystem. Since these systems are determined, we represent the size of each system by using the number of variables in the system. For instance, HFE(6) means an instance of HFE with six equations and six variables. The systems were constructed to have a unique solution. In Table 3.1, we collect the sizes of the resulting polynomial systems from HFE cryptosystem over $\mathbb{F}_2$ and compare the timings for their solution with the straightforward computation of a Gröbner basis in CoCoA [51].

| System | Equations | Variables | $d$ | Matrix Size | `LinBox` | `EF` | `GBasis` |
|---|---|---|---|---|---|---|---|
| HFE(6) | 6 | 6 | 2 | 57×133 | 0.02 | 0 | 0.01 |
| HFE(7) | 7 | 7 | 2 | 99×204 | 0.06 | 0 | 0.02 |
| HFE(8) | 8 | 8 | 2 | 163×297 | 0.13 | 0 | 0.13 |
| HFE(9) | 9 | 9 | 2 | 256×441 | 0.19 | 0 | 0.26 |
| HFE(10) | 10 | 10 | 2 | 386×595 | 0.4 | 0.05 | 0.8 |
| HFE(11) | 11 | 11 | 2 | 562×781 | 0.6 | 0.3 | 3.15 |
| HFE(12) | 12 | 12 | 2 | 794×1002 | 1.9 | 0.6 | 31.24 |
| HFE(13) | 13 | 13 | 3 | 2380×4915 | 86 | 15 | 349 |

Table 3.1: HFE size and time comparison using the LA Algorithm

See the results in Table 3.1. Each timing represents the total time taken by `LinBox` or `EF` to calculate echelon forms of all the matrices during the process of solving a particular instance of HFE. The sixth and seventh columns give the time taken by `LinBox` and `EF` respectively. The last column shows the time taken by the computation of a `Lex` Gröbner basis in CoCoA [51]. The fifth column shows the size of biggest matrix

that was formed during the process by the LA Algorithm. Note that the biggest matrix is formed when we attempt to find the first solution coordinate. For instance, the total time taken by `LinBox` for calculating echelon forms of all the matrices to solve HFE(13) is 86 seconds and the size of biggest matrix is 2380×4915. In fourth column we have degrees $d$ of the certificate polynomials.

As one can see from Table 3.1, in practice, the degree $d$ of the certificate polynomials is much lower than the known upper bounds (see Theorem 3.1.5 and Corollary 3.1.8) and degree growth seem to be very slow. Even for very small instances of HFE the running time of the LA Algorithm compare favorably to the running times of a Gröbner basis computation. We can also see the dependence of the running time on the linear algebra technique used. A sophisticated use of linear algebra can improve the running time a lot.

### 3.3.2 Experimental Results for CTC

Given the CTC cryptosystem and a plaintext-ciphertext pair, we construct an overdetermined algebraic system of equations in terms of the indeterminates representing key bits and certain intermediate quantities. The task is to solve the system for the key bits. The size of the system depends mainly on two parameters: the number B of simultaneous S-boxes and the number N of encryption rounds used. For more details see Section 2.3. From now on we denote a particular instance of CTC by CTC(B,N). In [146], S. Murphey and M. Robshow remarked:

> *We can, of course, immediately reduce the sizes of these multivariate quadratic systems by using the linear equations to substitute for state and key variables, though the resulting system is slightly less sparse.*

As we know, for systems of polynomial equations like CTC, there are three possible levels of substitution. They are as follows.

1) **No Substitution:** The first possibility is to consider a system without any substitution. We denote such an instance by $\text{CTC(B,N)}_0$.

2) **Substitution Using Linear Equations:** The second possibility is to use linear equations for substituting state and key variables. For any linear polynomial in the system CTC(B,N) one can consider the term $x_i$ as:

$$x_i = x_1 + \cdots + x_{i-1} + x_{i+1}, \ldots, x_n + 1,$$

| System | Equations | Variables | System | Equations | Variables |
|---|---|---|---|---|---|
| $\mathrm{CTC}(1,3)_0$ | 72 | 39 | $\mathrm{CTC}(3,3)_0$ | 216 | 117 |
| $\mathrm{CTC}(2,2)_0$ | 98 | 54 | $\mathrm{CTC}(3,4)_0$ | 285 | 253 |
| $\mathrm{CTC}(2,3)_0$ | 144 | 78 | $\mathrm{CTC}(4,3)_0$ | 288 | 156 |
| $\mathrm{CTC}(3,2)_0$ | 147 | 81 | $\mathrm{CTC}(4,4)_0$ | 380 | 204 |
| $\mathrm{CTC}(1,3)_1$ | 42 | 9 | $\mathrm{CTC}(3,3)_1$ | 126 | 27 |
| $\mathrm{CTC}(2,2)_1$ | 56 | 12 | $\mathrm{CTC}(3,4)_1$ | 168 | 36 |
| $\mathrm{CTC}(2,3)_1$ | 84 | 18 | $\mathrm{CTC}(4,3)_1$ | 168 | 36 |
| $\mathrm{CTC}(3,2)_1$ | 84 | 18 | $\mathrm{CTC}(4,4)_1$ | 224 | 48 |
| $\mathrm{CTC}(1,3)_2$ | 0 | 0 | $\mathrm{CTC}(3,3)_2$ | 42 | 9 |
| $\mathrm{CTC}(2,2)_2$ | 13 | 3 | $\mathrm{CTC}(3,4)_2$ | 42 | 9 |
| $\mathrm{CTC}(2,3)_2$ | 28 | 6 | $\mathrm{CTC}(4,3)_2$ | 53 | 11 |
| $\mathrm{CTC}(3,2)_2$ | 44 | 7 | $\mathrm{CTC}(4,4)_2$ | 56 | 12 |

Table 3.2: CTC instances used

where the term 1 is optional and any number of $x_j$ on right hand side could be zero. This is, in a sense, a re-definition of $x_i$, and so we add this equation to every polynomial in the system where $x_i$ appears. Afterward, $x_i$ will appear nowhere in the system of equations, except in its definition. Note that after the substitution the maximum degree of polynomials remains quadratic. We denote such an instance by $\mathrm{CTC(B,N)}_1$.

3) **Substitution Using Linear and Quadratic Equations:** The third possibility is to use linear and quadratic equations for substituting key variables. For any specific polynomial one can reorder the terms as follows.

$$x_i = x_1 + \cdots + x_{i-1} + x_{i+1}, \ldots, x_s + (\text{quadratic terms not containing } x_i) + 1,$$

where the term 1 is optional and any number of $x_j$ on right hand side could be zero. This is, in a sense, a re-definition of $x_i$, and so we add this equation to every polynomial in the system where $x_i$ appears. Afterward, $x_i$ will appear nowhere in the system of equations, except in its definition. Note that after the substitution the maximum degree of polynomials is greater than two. We denote such an instance by $\mathrm{CTC(B,N)}_2$.

In Table 3.2, we collect the sizes of some resulting polynomial systems for three possible levels of substitution. These systems are such that each of them has a unique solution over $\mathbb{F}_2$. These systems will be used to report on experimental results in

this section.

### Substitution Using Linear and Quadratic Equations

If algorithms are employed which do not exploit possible sparseness of a system this substitution level may provide a slight improvement. This substitution results in an equation system in the key variables only. However, as S-box equations are substituted, it is not guaranteed that solutions found also solve the original system of equations. This is because there are fewer constrains on the variables than in the original system. So correctness is not guaranteed with this substitution. In this case we have a system of equations with much less equations and variables. Although this system is not sparse anymore, it gives us much better results as compared to other substitution levels as reported by Table 3.3.

| System | `SEF` | `LinBox` | `EF` | `GBasis` | $d$ | Matrix Size |
|---|---|---|---|---|---|---|
| $CTC(2,2)_2$ | 0 | 0 | 0 | 0 | 0 | 8×14 |
| $CTC(2,3)_2$ | 0.04 | 0.08 | 0 | 0.09 | 0 | 64×197 |
| $CTC(3,2)_2$ | 0.4 | 0.24 | 0 | 0.14 | 0 | 128×352 |
| $CTC(3,3)_2$ | 23 | 3.5 | 1 | 0.83 | 0 | 512×1933 |
| $CTC(3,4)_2$ | 45 | 5 | 0.7 | 2.99 | 0 | 512×1933 |
| $CTC(4,3)_2$ | 820 | 33 | 12 | 128 | 1 | 2048×3551 |

Table 3.3: $CTC(B,N)_2$ time comparison using the LA Algorithm

In Table 3.3, each timing represents the total time taken by `SEF`, `LinBox` or `EF` to calculate echelon forms of all the matrices during the process of solving a particular instance of CTC. The fifth column shows the time taken by the computation of a `Lex` Gröbner basis in CoCoA [51]. In sixth column we have degrees $d$ of the certificate polynomials. Note that, in practice, the degree growth of the certificate polynomials is very slow. The last column shows the size of biggest matrix that was formed during the process by the LA Algorithm.

### Substitution Using Linear Equations

After this substitution the system is less sparse. For instance, the order of biggest matrix to solve $CTC(2,2)_1$ is 299×729 and it has 21.8 non-zero elements per row. Whereas the size of biggest matrix to solve $CTC(2,2)_0$ is 96520×145605 and it has 5.7 non-zero elements per row. Further experimental results are reported in Table 3.4.

| System | SEF | $\text{SEF}_{\text{Lex}}$ | GBasis | $d$ | Matrix Size |
|---|---|---|---|---|---|
| $\text{CTC}(2,2)_1$ | 0.11 | 0.11 | 0.12 | 1 | 299×729 |
| $\text{CTC}(2,3)_1$ | 2 | 1.5 | 1.69 | 1 | 988×1597 |
| $\text{CTC}(3,2)_1$ | 6 | 4 | 1.8 | 1 | 988×1597 |
| $\text{CTC}(3,3)_1$ | 14506 | 7549 | 7315 | 2 | 20854×47753 |
| $\text{CTC}(3,4)_1$ | $>$ 100h | 81h | $>$ 100h | 2 | 66699×112723 |

Table 3.4: $\text{CTC(B,N)}_1$ time comparison using the LA Algorithm

Each timing in Table 3.4 represents the total time taken by SEF or EF to calculate echelon forms of all the matrices during the process of solving a particular instance of CTC except CTC(3,4), where we have considered timing in hours only for the biggest matrix. The fifth column shows the time taken by the computation of a Lex Gröbner basis in CoCoA [51]. In sixth column we have degrees $d$ of the certificate polynomials. Again note that, in practice, the degree growth of the certificate polynomials is very slow. The last column shows the size of the biggest matrix that was formed during the process by the LA Algorithm.

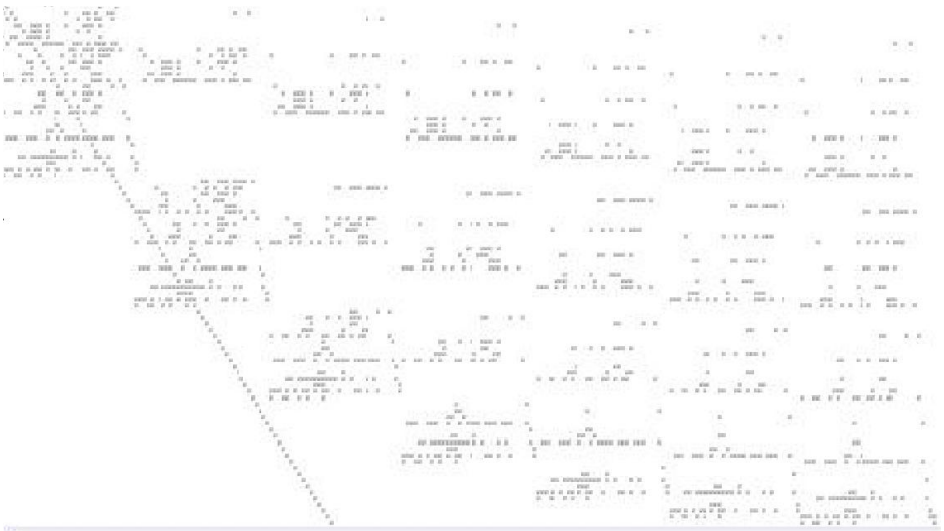


Figure 3.1: Structure of $\text{CTC}(1,2)_1$

Note that both second and third columns represent the time taken by SEF. The timings in column three represented by $\text{SEF}_{\text{Lex}}$ are taken after processing the system as follows. Order the polynomials in a system CTC(B,N) such that a polynomial with less number of terms comes first. In the algorithm of Theorem 3.2.4 linear equations correspond to terms. Arrange these linear equations using Lex order on terms. Fur-

thermore, consider the matrix of the linear system in step 6) of the LA Algorithm. A column of this matrix represents $tf_i$ for some term $t$. Arrange these columns as follows. The polynomial $t'f_i$ proceeds the polynomial $t'f_i$ if we have $\deg(t') < \deg(t)$, or if we have $\deg(t') = \deg(t)$ and $t' \geq_{\texttt{Lex}} t$. After preprocessing the matrix carries a particular structure on it which is very helpful to calculate an echelon form. Actually, in this matrix a big portion below the principal diagonal contains only zero entries. For instance, a degree one certificate matrix for $\mathrm{CTC}(1,2)_1$ looks like the one in Figure 3.1, where the diagonal straight line represents principal diagonal.

In column three of Table 3.4 the main idea is to exploit the portion under the principal diagonal. This technique worked effectively for us in solving matrices as large as $91219 \times 136069$ and with almost 23 elements per row in almost 492004 seconds on a small PC with 2.2 GHz processor and 20 GB of RAM. The actual memory used during the whole process was only about 1.8 GB. Note that `SEF` is implemented in a straightforward way to calculate an echelon form without using any sophisticated data structures. Also note that `LinBox` cannot solve linear systems of this size with time and space requirements as described above. The structural information described above is also valid for $\mathrm{CTC(B,N)}_0$.

**No Substitution**

If linear and quadratic equations are not substituted and algorithms are employed which do not exploit possible sparseness of given system then time and memory requirements are much higher. Since there are more variables in the system, the degree of a certificate could also be higher. A natural interpretation of this fact is that the degree of a certificate is exponential in the number of variables and the total degree of the system.

| System | Non-Zero | $\texttt{SEF}_{\texttt{Lex}}$ | $d$ | Matrix Size |
|---|---|---|---|---|
| $\mathrm{CTC}(1,2)_2$ | 5.0 | 0.3 | 1 | 1049×1373 |
| $\mathrm{CTC}(1,3)_2$ | 6.1 | 26 | 1 | 35030×56214 |
| $\mathrm{CTC}(2,2)_2$ | 5.7 | 337 | 2 | 96520×145605 |

Table 3.5: $\mathrm{CTC(B,N)}_0$ time comparison using the LA Algorithm

Since matrix is quite sparse another natural idea could be to apply the so called structured Gaussian elimination [123]. We have also implemented and tried structured Gaussian elimination. But it does not seem to perform in an expected way. The non-zero elements are scattered throughout the matrix so that it is difficult to distinguish between light and heavy parts. Furthermore, consider the matrix of the linear system

in step 6) of the LA Algorithm. A column of this matrix represents $tf_i$ for some term $t$. Thus the number of non-zero entries in a column is equal to number of terms in the polynomial $tf_i$. The reason for this is that there are almost no columns in the matrix with only one non-zero entry. So we conclude that this method can work only for matrices which have a specific structure such as matrices coming from integer factorization problems. The usefulness of its application to matrices which come from the LA Algorithm is not clear.

## 3.4 The Mutant LA Algorithm

In this section we highlight a hybrid method that combines ideas studied by De Loera et al. [61, 64] for transforming combinatorial infeasibility proofs to large systems of linear equations and ideas of J. Ding et al. [67, 140, 142, 143] involving the concept of mutants. The concept of mutants was first discovered by J. Ding [67] and has the potential to improve various algorithms for solving systems of polynomial equations which use linear algebra. We review the concept of mutants and explore the potential of their application to improve the LA Algorithm studied in Section 3.2.

One of the most useful applications of Gröbner bases is to compute the solution set of a system of polynomial equations. The Buchberger's algorithm [34] was the first algorithm for computing Gröbner bases. It is based on the computation of Gröbner bases using S-polynomials. Due to complexity issues of the standard Buchberger algorithm, several algorithms such as F4 [73], F5 [74] and XL (extended linearization) [59] have been proposed. The F4 [73] is an algorithm that uses linear algebra and Buchberger's S-polynomial techniques to compute Gröbner bases. It takes advantage of fast linear algebra techniques and possible sparseness of the linear system. The F5 algorithm was introduced in [74] to avoid reductions to zero. The F4 and F5 algorithms rose to popularity after they were successfully used to break the "HFE 80 Challenge". Recent improvements made them even more powerful.

The XL Algorithm was proposed in [59] as an efficient algorithm which in turn is based on a technique called relinearization. It was proposed for solving multivariate polynomial systems in case only a single solution exists. Contrary to some initial hopes, it does not solve the problem in subexponential time and, worse, it suffers from a rapid increase in memory consumption. To overcome this difficulty, several optimizations have been proposed. Actually, these algorithms find additional polynomials of not much larger degree in the ideal generated by the polynomials of the system $F$ by

multiplying them by terms. They apply Gaußian elimination after linearizing the system by replacing terms with variables.

In [67], J. Ding observed that during Gaußian elimination some special polynomials of degree lower than what they normally should be appear and he named them as mutants. The idea is that in the process of generating new equations, polynomials of small degree should be treated preferentially. Mutant strategy can be used to improve various algorithms for solving systems of polynomial equations which use linear algebra.

In [68], J. Ding et al. proposed the MutantXL algorithm as a variant of the XL Algorithm that is based on the mutant strategy. Note that the XL Algorithm is the first algorithm that is equipped with the mutant strategy until now. Therefore, there is a need to design mutant variants of other linear algebra algorithms such as the LA Algorithm and the Border basis algorithm for polynomial system solving.

In the following, we explore the potential of the mutant strategy to improve the LA Algorithm. This leads to a hybrid method that combines the ideas of De Loera et al. [61, 64] and J. Ding et al. [67, 140, 142, 143]. Before digging deep into further details, let us first define a bit of terminology to use in this section.

Let $p$ be a prime number, let $q = p^e$ for some $e > 0$, let $K = \mathbb{F}_q$ be the finite field with $q$ elements, and let $F = \{f_1, \ldots, f_m\}$ be a set of polynomials over $K$, i.e. $f_1, \ldots, f_m \in P = K[x_1, \ldots, x_n]$. Let $I = \langle F \rangle$ be the ideal generated by $F$. As we are interested in finding $K$-rational solutions of the system of polynomial equations defined by $F$, we will be working over the ring

$$R = K[x_1, \ldots, x_n]/\langle x_1^q - x_1, \ldots, x_n^q - x_n \rangle,$$

where we reduce everything modulo the field polynomials. In the process of solving the system, we are typically generating further elements of the ideal $I$ in $R$. Recall from Section 2.2 that $R$ is the residue class ring of $K[x_1, \ldots, x_n]$ whose every element (residue class) is represented as a polynomial. Since each element (residue class) in $R$ is represented by a polynomial, we can define the degree, leading term and leading variable of this polynomial in the natural way.

*In particular, we assume that the polynomial system defined by $F$ has a unique $K$-rational solution.* Furthermore, let $\sigma$ be a degree compatible term ordering on the terms of $P$. The monoid of terms is $\mathbb{T}^n$ and for $d \in \mathbb{N}$, we let $\mathbb{T}^n_{\leq d}$ denote the set of terms of $R$ with total degree at most $d$. In this setting we define mutants as follows.

**Definition 3.4.1.** Let $g \in I$. Let $g = h_1 f_1 + \cdots + h_m f_m$ be a representation of $g$,

where $h_i \in R$.

a) The number $l = \max\{\deg(h_i f_i) \mid i \in \{1, \ldots, m\}, h_i \neq 0\}$ is called the **level** of the representation $g = h_1 f_1 + \cdots + h_m f_m$.

b) The minimal level of any representation of $g$ is called the **level** of $g$ with respect to $F$.

c) We say that $g$ is a **mutant** with respect to $F$ if $\deg(g)$ is smaller than the level of $g$.

In Definition 3.4.1, the polynomial $g = h_1 f_1 + \cdots + h_m f_m \in I$ is called a mutant if one of the $h_i f_i, i = 1, \ldots, m$ has degree greater than the degree of $g$. Therefore, a mutant of degree $d$ is a polynomial which cannot be found by forming linear combinations of the products $tf_i$, where $t \in \mathbb{T}^n$ is a term such that $\deg(tf_i) \leq d$ and $i \in \{1, \ldots, m\}$. However, such mutants could help in solving the system $F$ if we can find them efficiently while implementing polynomial system solving algorithms. The mutants are identified and used during the process of applying Gaußian elimination. Before giving another important definition for mutants we need some ingredients which are given in the following.

**Definition 3.4.2.** Let $F$ be a vector subspace of $R$. We define

$$F^+ := F + \sum_{i=1}^{n} x_i F$$

where $x_i F := \{x_i f \mid f \in F\}$.

Note that $F^+$ is also a vector subspace of $R$. Then $F^+$ is precisely the linear span of $F$ and $x_i F$ for all $i \in \{1, \ldots, n\}$.

**Definition 3.4.3.** Let $F \subseteq L$ be vector subspaces of $R$. Define inductively the vector subspaces

$$F_0 = F \quad \text{and} \quad F_{k+1} = F_k^+ \cap L \text{ for } k = 0, 1, 2, \ldots$$

The union $F_L = \bigcup_{k \geq 0} F_k$ of ascending chain $F_0 \subseteq F_1 \subseteq F_2 \subseteq \ldots$ is called the **$L$-stable span of** $F$.

To keep things computable and to define mutants, we prefer finite-dimensional vector space $L$. In particular, we consider $L = \langle \mathbb{T}^n_{\leq d} \rangle_K$. Now we are ready to give an

alternate definition for mutants that explains how mutants can be distinguished during the computations from the rest of polynomials.

**Definition 3.4.4.** Let $F$ be the set of polynomials as above, let $L = \langle \mathbb{T}^n_{\leq d} \rangle_K$, and let $U = \langle \mathbb{T}^n_{\leq d-1} \rangle_K$. Let

$$F_d = \{tf_i \mid t \in \mathbb{T}^n, f_i \in F, \deg(tf_i) \leq d\}.$$

A polynomial $m$ is called a **mutant** if $\mathrm{LT}(m) \in \mathrm{LT}(F_L) \setminus \mathrm{LT}(F_U)$ and its degree is less than $d$.

The following example illustrates how we can generate mutants and how we can use them to solve systems.

**Example 3.4.5.** Let $F = \{f_1, \ldots, f_4\} \subseteq R = \mathbb{F}_2[x_1, \ldots, x_4]/\langle x_1^2 - x_1, \ldots, x_4^2 - x_4 \rangle$. Consider the system of equations $f_1 = f_2 = f_3 = f_4 = 0$, where

$$\begin{aligned} f_1 = x_1x_2 + x_2x_3 + x_2x_4 + x_3x_4 + x_1 + x_3 + 1 &= 0, \\ f_2 = x_1x_2 + x_1x_3 + x_1x_4 + x_3x_4 + x_2 + x_3 + 1 &= 0, \\ f_3 = x_1x_2 + x_1x_3 + x_2x_3 + x_3x_4 + x_1 + x_4 + 1 &= 0, \\ f_4 = x_1x_3 + x_1x_4 + x_2x_3 + x_2x_4 + 1 &= 0. \end{aligned}$$

Considering terms as indeterminates, Gaussian elimination yields the following system.

$$\begin{aligned} \widetilde{f_1} = x_1x_2 + x_2x_3 + x_2x_4 + x_3x_4 + x_1 + x_3 + 1 &= 0 \\ \widetilde{f_2} = x_1x_3 + x_1x_4 + x_2x_3 + x_2x_4 + x_1 + x_2 &= 0 \\ \widetilde{f_3} = x_1x_4 + x_2x_3 + x_1 + x_2 + x_3 + x_4 &= 0 \\ \widetilde{f_4} = x_1 + x_2 + 1 &= 0 \end{aligned}$$

The polynomial $\widetilde{f_4}$ is mutant with respect to $F$.

**Definition 3.4.6.** Let $F = \{f_1, \ldots, f_m\} \subseteq K[x_1, \ldots, x_n]$ be a set of polynomials. Let $d = \max\{\deg(f_i) \mid i \in \{1, \ldots, m\}\}$ and let $\mathbb{T}^n_{\leq d} = \{t_1, \ldots, t_\mu\}$. Let $\sigma$ be a term ordering. Then the **term vector** of $F$ with respect to $\sigma$ is denoted by $\mathcal{L}$ and is defined as $\mathcal{L} = (t_1, \ldots, t_\mu)$, where $t_1 >_\sigma t_2 >_\sigma \cdots >_\sigma t_\mu$.

Note that $\mathcal{L}$ is nothing but the set $\mathbb{T}^n_{\leq d}$ ordered by $\sigma$.

**Definition 3.4.7.** Let $f \in P = K[x_1, \ldots, x_n]$ be a polynomial and let $\sigma$ be a term ordering. Let $\mathcal{L} = (t_1, \ldots, t_\mu)$ be the term vector of $f$. Write $f$ as

$$f = \sum_{i=1}^{\mu} c_i t_i \text{ with } c_i \in K.$$

Then the **vector representation** $\psi_{\mathcal{L}} : P \longrightarrow K^\mu$ of $f$ with respect to $\mathcal{L}$ is defined as follows

$$\psi_{\mathcal{L}}(f) = (c_1, \ldots, c_\mu).$$

**Definition 3.4.8.** Let $F = \{f_1, \ldots, f_m\} \subseteq P = K[x_1, \ldots, x_n]$ be a set of polynomials and let $\sigma$ be a term ordering. Let $\mathcal{L} = (t_1, \ldots, t_\mu)$ be the term vector of $F$. Then the **matrix representation** (also called as **coefficient matrix**) of $F$ with respect to $\mathcal{L}$ is defined by the map

$$\psi_{\mathcal{L}}(F) : P^m \longrightarrow \mathrm{Mat}_{m,\mu}(K)$$

such that

$$(f_1, \ldots, f_m) \longmapsto \begin{pmatrix} \psi_{\mathcal{L}}(f_1) \\ \vdots \\ \psi_{\mathcal{L}}(f_m) \end{pmatrix} = (c_{ij}).$$

We denote it by $\mathcal{F} = (c_{ij}) \in \mathrm{Mat}_{m,\mu}(K)$.

Note that the vector representation of a polynomial $f \in K[x_1, \ldots, x_n]$ is the matrix representation of the set $\{f\}$.

**Definition 3.4.9.** Let $F = \{f_1, \ldots, f_m\} \subseteq K[x_1, \ldots, x_n]$ be a set of polynomials and let $\sigma$ be a term ordering. Let $\mathcal{L} = (t_1, \ldots, t_\mu)$ be the term vector of $F$. Let $x_i \in \{x_1, \ldots, x_n\}$ and let $k \in K$. Then the **constant vector** of $F$ is given by $\mathcal{C} = (c_1, \ldots, c_\mu)$ such that

$$c_j = \begin{cases} 1 & \text{if } t_j = x_i, \\ -k & \text{if } t_j = 1, \\ 0 & \text{otherwise.} \end{cases}$$

**Example 3.4.10.** Let $F = \{f_1, f_2\} \subseteq \mathbb{F}_2[x, y]$, be a set of polynomials, where $f_1 = xy + y + 1$ and $f_2 = xy + x$. Let $\sigma = \texttt{DegLex}$.

1) The term vector is $\mathcal{L} = (xy, x, y, 1)$.

2) The coefficient matrix is given as follows

$$\mathcal{F} = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

such that

$$\mathcal{FL} = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} xy \\ x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} xy + y + 1 \\ xy + x \end{pmatrix}.$$

In this way if we consider terms as indeterminates then we can view $F$ represented by a linear system of equations.

3) For $k = 1$ and $x_i = x \in \{x, y\}$, the constant vector is $\mathcal{C} = (0, 1, 0, 1)$.

**Definition 3.4.11.** Let $F = \{f_1, \ldots, f_m\} \subseteq K[x_1, \ldots, x_n]$ be a set of polynomials. Let $\mathcal{F}$ be the coefficient matrix of $F$.

a) The row echelon form of $\mathcal{F}$ will be denoted by $\widetilde{\mathcal{F}}$.

b) The set of polynomials corresponding to the row echelon form $\widetilde{\mathcal{F}}$ will be denoted by $\widetilde{F}$.

**Remark 3.4.12.** Let $F = \{f_1, \ldots, f_m\} \subseteq R$ be a set of polynomials and let $d_m = \max\{\deg(f_j) \mid j \in \{1, \ldots, m\}\}$. Let $d_c$ be the degree of a certificate for an $i^{\text{th}}$ solution coordinate $k \in K$. Let $\mathbb{T}^n_{\leq d_c} = \{t_1, \ldots, t_\nu\}$ and let $\mu = |\mathbb{T}^n_{\leq d_m + d_c}|$. Thus, the certificate is

$$x_i - k = \sum_{j=1}^{m} g_j f_j \tag{3.6}$$

where $g_j = \sum_{k=1}^{\mu} c_{jk} t_k \in R$ with indeterminate coefficients $c_{jk} \in K$. Let

$$(\mathcal{F} \mid \mathcal{B}) \tag{3.7}$$

be the augmented matrix of the linear system obtained by comparing the coefficients of each term $t \in \mathbb{T}^n_{\leq d_m + d_c}$ on both sides of the certificate 3.6. The matrix $\mathcal{F}$ has one column corresponding to each indeterminate coefficient $c_{jk}$ and one row corresponding to each term in $\mathbb{T}^n_{\leq d_m + d_c}$. The vector $\mathcal{B} = (b_1, \ldots, b_\mu)$ has one entry for each term

$t \in \mathbb{T}^n_{\leq d_m+d_c}$ such that

$$b_j = \begin{cases} 1 & \text{if } t_j = x_i, \\ -k & \text{if } t_j = 1, \\ 0 & \text{otherwise.} \end{cases}$$

Now consider $H = \{tf \mid f \in F \text{ and } t \in \mathbb{T}^n_{\leq d_s}\}$. Let $\mathcal{H}$ be the coefficient matrix of $H$ and let $\mathcal{C}$ be the constant vector of $H$. Form the following augmented matrix

$$(\mathcal{H}^{tr} \mid \mathcal{C}) \tag{3.8}$$

A closer look at the augmented matrices 3.7 and 3.8 shows that they are the same. Note that this is a very important observation to formulate the Mutant LA Algorithm. This will help us to extract linear systems without writing certificates.

The main idea that we discuss is to generate the ideal $I$ gradually until we have a linear system which has a solution. In view of Remark 3.4.12, we slightly change the way of extracting a linear system. We extract the associated linear system by generating new polynomials $tf_j \in I$, where $t \in \mathbb{T}^n_{\leq d_c}$, instead of writing down a certificate. After fixing a tentative degree equal to $\deg(F)$, we will be working in the vector space bounded by this degree. The basic idea behind the Mutant LA Algorithm is that, if we have not found a linear system which has a solution, then instead of replacing $F$ with $F^+$ and thereby increasing the degree bound, we check whether there are any mutants in $F$. If there are some mutants found then we use them and try to find a linear system which has a solution.

Due to ideas discussed above, the LA Algorithm can be extended to the Mutant LA Algorithm (MLA), a modification of the LA Algorithm that uses mutants. These ideas provide us with the following algorithm.

**Theorem 3.4.13. (The Mutant LA Algorithm)**
*Let $F = \{f_1, \ldots, f_m\} \subseteq R$ be a set of non-zero polynomials such that the system of polynomial equations $f_1 = 0, \ldots, f_m = 0$ has a unique solution $(a_1, \ldots, a_n) \in K^n$. Let $\sigma$ be a degree compatible term ordering on $\mathbb{T}^n$. Consider the following sequence of instructions.*

*1) Interreduce $F$, let $S := \emptyset$, $X := \{x_1, \ldots, x_n\}$, and let $H := F$.*

*2) If $X := \emptyset$, return $S$. Otherwise, choose an indeterminate $x_s \in X$ and delete it from $X$. Let $P_{mutant} := \emptyset$, $Q := K$ and $d_{min} := d_{elim} := \min\{\deg(h) \mid h \in H\}$.*

*3) If $Q \neq \emptyset$, then choose an element $k \in Q$, delete it from $Q$ and continue with step 8). Otherwise, let $Q := K$.*

*4) Form the coefficient matrix $\mathcal{H}$ of $H$. Use Gaußian elimination to bring $\mathcal{H}$ into row echelon form $\widetilde{\mathcal{H}}$ and let $H := \widetilde{H}$.*

*5) Form the set $M$ of all polynomials of degree less than $d_{elim}$ in $H$. (The polynomials in the set $M \setminus P_{mutant}$ are mutants with respect to $F$.)*

*6) If $M \setminus P_{mutant} \neq \emptyset$, then for each $m \in M \setminus P_{mutant}$ add $m$ to $P_{mutant}$ and all possible products $x_\alpha m$ with $\alpha \in \{1, \ldots, n\}$ to $H$. Let $d_{elim} := \min\{\deg(m) \mid m \in M \setminus P_{mutant}\} + 1$. Then continue with step 3).*

*7) For each $h \in H \setminus P_{mutant}$ of degree $d_{min}$ add $h$ to $P_{mutant}$ and all possible products $x_\alpha h$ with $\alpha \in \{1, \ldots, n\}$ to $H$. Increase $d_{min}$ by one, and let $d_{elim} := d_{min}$. Then continue with step 3).*

*8) Form the coefficient matrix $\mathcal{H}$ of $H$. Form the constant vector $\mathcal{C}$ of $H$.*

*9) Solve the linear system given by the augmented matrix $(\mathcal{H}^{tr} \mid \mathcal{C})$.*

*10) If the linear system in step 9) has no solution then continue with step 3).*

*11) Append the corresponding $k$ to $S$, substitute $x_s = k$ in $H$. Then continue with step 2), applied to polynomials in a smaller ring.*

*This is an algorithm which returns the solution $(a_1, \ldots, a_n)$ of the system of polynomial equations $f_1 = 0, \ldots, f_m = 0$.*

*Proof.* If we exclude the steps 4)−6) then we obtain the algorithm in Theorem 3.2.4. In these steps mutants are found and used to extend the set $H$. This implies that the linear systems in Theorem 3.2.4 are subsystems of the linear systems in step 9). So the algorithm terminates since the algorithm in Theorem 3.2.4 terminates.

If we exclude steps 4)−6), the correctness follows from Theorem 3.2.4. Therefore, to see the correctness it suffices to show that the polynomials found in steps 4)−6) are, in fact, mutants with respect to the current set of polynomials $F$. For this consider the elimination degree $d_{elim}$ in step 5) of the algorithm. Let $L := \langle \mathbb{T}^n_{\leq d_{elim}} \rangle_K$ and $U := \langle \mathbb{T}^n_{\leq d_{elim}-1} \rangle_K$. Let $m \in M \setminus P_{mutant}$ as in step 6) of the algorithm. Then from steps 5)−6) it is easy to see that $\mathrm{LT}(m) \in \mathrm{LT}(F_L) \setminus \mathrm{LT}(F_U)$ and $\deg(m) < d_{min}$. Therefore, $m$ is a mutant. □

**Remark 3.4.14.** We have the following remarks for the MLA Algorithm.

a) The Mutant LA Algorithm inherits all useful properties of the LA Algorithm like parallelization, taking advantage of fast linear algebra and exploiting the possible sparseness and the structure of the system. In addition it uses mutant strategy to guide and improve the performance of the LA Algorithm. The experimental results reported in Section 3.6 clearly show that running time of the Mutant LA Algorithm compares favorably to the running times of the LA Algorithm. This is possible due to smaller sized linear systems in step 9).

b) Another issue in steps 6)−7) is to control the redundancy produced while multiplying polynomials with indeterminates. This can be achieved by using some "selection strategy" while multiplying polynomials with indeterminates. For instance, if we are working over the field $\mathbb{F}_2$, redundancy can be avoided as follows. During the multiplication process we keep the multiplier indeterminate that gave rise to every newly produced polynomial and we keep one for the original polynomials. When we extend the system, we multiply the polynomial $h$ by all indeterminates smaller than its previous multiplier indeterminate. In case of the previous multiplier of $h$ is one, we multiply by all indeterminates. The aim of this selection method is to speed up the extension process of the system. We only multiply by terms of degree one (indeterminates) without any trivial redundancy.

## 3.5 The Improved Mutant LA Algorithm

In [142], M.S.E. Mohamed et al. introduced an improvement to the mutant strategy called the *improved mutant strategy*. In this section, we review the improved mutant strategy and employ this strategy to improve the LA Algorithm. This strategy allows us to solve systems of polynomial equations with small sized linear systems.

In Section 3.4 we saw how mutant strategy achieves to enlarge the system $F$ without increasing the degree of the coefficient polynomials in a certificate. The experiments with mutant strategy [142] show that there are two issues. The first arises when the number of lower degree mutants is very large, this produces many reductions to zero. The second arises when an iteration does not produce mutants at all or produces only an insufficient number of mutants at a lower degree. In this case, the mutant strategy makes no improvement. To handle both problems an improved mutant strategy was introduced in [142]. This strategy is based on the mutant concept. It introduces a

heuristic strategy of only choosing the minimum number of mutants, which is called *necessary mutants*.

Although the LA Algorithm can be used for solving big system of polynomial equations but we can not ignore the fact that it can be infeasible for many realistic applications. This is due to the fact that, in many practical cases, the computations made by these algorithms lead to constructing a huge system of polynomial equations, and consequently a huge matrix, which requires a lot of time and memory resources. Therefore, a big challenge is to improve this algorithms in a way allowing it to use only the limited available memory and time resources for solving a multivariate polynomial system with as large number of equations and variables as possible.

One of the strategies to improve the efficiency of this algorithms is to find better linear algebra techniques. This mainly reduces the time consumption. On the other hand, strategies improving the enlargement step of the polynomial system, by reducing the matrix size, will affect both time and memory consumption. This section proposes to use improved mutant strategy along with the LA Algorithm to improve the enlargement step of the LA algorithm. In Section 3.6 we show with the help of experiments that combining the improved mutant strategy with LA Algorithm gives both time and memory efficiency. We use the notation and terminology from Section 3.4. Let $X = \{x_1, \ldots, x_n\}$ be the set of indeterminates, upon which we impose the following order:

$$x_1 > x_2 > \cdots > x_n$$

Our first goal is to recall a few definitions to be able to formulate the algorithm.

**Definition 3.5.1.** Let $p \in R$. Then the **leading variable** of $p$ is the largest indeterminate, according to the order defined on the indeterminates, in the leading term of $p$ and denoted by $\mathrm{LV}(p)$.

**Definition 3.5.2.** Let $F_d = \{f \in F \mid \deg(f) = d\}$ and let $x_i \in X$. We define $F_d^{x_i}$ as follows

$$F_d^{x_i} = \{f \in F_d \mid \mathrm{LV}(f) = x_i\}.$$

**Remark 3.5.3.** As in Section 3.4, let $F = \{f_1, \ldots, f_m\} \subseteq R$ be a system of polynomials. For $d \in \mathbb{N}$, we let $\mathbb{T}^n_{\leq d}$ denote the set of terms of $R$ with total degree at most $d$. The two issues described above can be handled as follows.

a) **Partitioned Enlargement Strategy:** In the process of enlargement step, the improved mutant strategy deals with the polynomials of $F_d$ in a different way.

We divide the set $F_d$ into a set of subsets depending on the leading variable of each polynomial. In other words, we view $F_d$ as $F_d = \bigcup_{i=1}^{n} F_d^{x_i}$. The improved mutant strategy enlarges $F$ by incrementing $d$ and multiplying the elements of $F_d$ as follows. Let $x$ be the smallest variable, according to the order defined on the variables, that has $F_d^x \neq \emptyset$. Improved mutant strategy successively multiplies each polynomial of $F_d^x$ by indeterminates such that each variable is multiplied only once. This process is repeated for the next larger indeterminate $x$ with $F_d^x \neq \emptyset$ until the solution is obtained, otherwise the system enlarges to the next $d$. Therefore the improved mutant strategy may solve the system by enlarging only subsets of $F_d$, while the mutant strategy solves the system by enlarging all the elements of $F_d$. This technique is helpful if we have a system with not enough mutants at a certain degree. In this case the Mutant LA Algorithm increases the degree of a certificate. In most cases only a small number of the extended polynomials produced are needed to solve the system. By using this strategy, we can solve the system $F$ with small size linear systems.

b) **Necessary Mutants:** If the number of lower degree mutants is very large then we can select mutants which are necessary to solve the system using some selection criteria. For instance, if we are working over the field $\mathbb{F}_2$, the following notation helps. Using combinatorics we can compute the number of elements of the set $\mathbb{T}^n_{\leq d}$ as

$$|\mathbb{T}^n_{\leq d}| = \sum_{i=1}^{d} \binom{n}{i}, 1 \leq d \leq n.$$

Let $k$ be the degree of the lowest degree mutant occurring and let $N$ be the number of the linearly independent polynomials of total degree at most $k+1$. Then the smallest number of mutants that are needed to generate $|\mathbb{T}^n_{\leq k+1}|$ linearly independent equations of total degree at most $k+1$ is

$$\lceil (|\mathbb{T}^n_{\leq k+1}| - N)/n \rceil.$$

By using only the necessary number of mutants, the system can be solved by a smaller number of polynomials and a minimum number of multiplication.

The following theorem turns above ideas into an effective algorithm.

**Theorem 3.5.4. (The Improved Mutant LA (MLA2) Algorithm)**
*Let $F = \{f_1, \ldots, f_m\} \subseteq R$ be a set of non-zero polynomials such that the system of*

*polynomial equations $f_1 = 0, \ldots, f_m = 0$ has a unique solution $(a_1, \ldots, a_n) \in K^n$. Let $\sigma$ be a degree compatible term ordering on $\mathbb{T}^n$. Consider the following sequence of instructions.*

*1) Interreduce $F$, let $S := \emptyset$, $X := \{x_1, \ldots, x_n\}$, and let $H := F$.*

*2) If $X := \emptyset$, return $S$. Otherwise, choose an indeterminate $x_s \in X$ and delete it from $X$. Let $P_{mutant} := \emptyset$, $Q := K$, $d_{min} := d_{elim} := \min\{\deg(h) \mid h \in H\}$ and let $X' := \{\mathrm{LV}(h) \mid h \in H\}$ be the set of leading variables.*

*3) If $Q \neq \emptyset$, then choose an element $k \in Q$, delete it from $Q$ and continue with step 9). Otherwise, let $Q := K$.*

*4) Form the coefficient matrix $\mathcal{H}$ of $H$. Use Gaußian elimination to bring $\mathcal{H}$ into row echelon form $\widetilde{\mathcal{H}}$ and let $H := \widetilde{H}$.*

*5) Form the set $M$ of all polynomials of degree less than $d_{elim}$ in $H$. (The polynomials in the set $M \setminus P_{mutant}$ are mutants with respect to $F$.)*

*6) If $M \setminus P_{mutant} \neq \emptyset$, then let $\nu := \min\{\deg(m) \mid m \in M \setminus P_{mutant}\}$ and let $M_{=\nu} := \{m \in M \setminus P_{mutant} \mid \deg(m) = \nu\}$ be the set of necessary mutants. For each $m \in M_{=\nu}$ add $m$ to $P_{mutant}$ and all possible products $x_\alpha m$ with $\alpha \in \{1, \ldots, n\}$ to $G$. Let $d_{elim} := \nu + 1$. Then continue with step 3).*

*7) If $X' := \emptyset$ then let $X' := \{\mathrm{LV}(h) \mid h \in H\}$ be the set of leading variables, increase $d_{min}$ by one, and let $d_{elim} := d_{min}$.*

*8) Choose the smallest leading variable $x_l \in X'$ and delete it from $X'$. Form the set $PG$ of polynomials which belong to $H \setminus P_{mutant}$ and $H^{x_l}_{d_{min}}$. For each $h \in PG$ add $h$ to $P_{mutant}$ and all possible products $x_\alpha h$ with $\alpha \in \{1, \ldots, n\}$ to $H$. Then continue with step 3).*

*9) Form the coefficient matrix $\mathcal{H}$ of $H$. Form the constant vector $\mathcal{C}$ of $H$.*

*10) Solve the linear system given by the augmented matrix $(\mathcal{H}^{tr} \mid \mathcal{C})$.*

*11) If the linear system in step 10) has no solution then continue with step 3).*

*12) Append the corresponding $k$ to $S$, and substitute $x_s = k$ in $H$. Then continue with step 2), applied to polynomials in a smaller ring.*

*This is an algorithm which returns the unique solution* $(a_1, \ldots, a_n)$ *of the system of polynomial equations* $f_1 = 0, \ldots, f_m = 0$.

*Proof.* To show that the procedure terminates and that the algorithm is correct, we consider its steps which differ from the algorithm in Theorem 3.4.13. The steps 6)−8) compute the necessary mutants. The additional mutants, if any, produce reductions to zero. They do not play any role in finding the respective solution coordinate. If the number of mutants produced in step 6) is less than the number of necessary mutants then the mutants are used to extend the system $H$ and we move on to the next steps, otherwise we found the respective solution coordinate. Step 8) extends the system $H$ using a partitioned enlargement strategy. This strategy does not increase the total degree of the system $H$. This shows that the associated linear systems are subsystems of the linear systems in Theorem 3.4.13. This covers all changes in the algorithm and concludes the proof. □

**Remark 3.5.5.** Future work on this topic could involve to see how to build these algorithms using a sparse matrix representation instead of the dense one and the use of techniques from linear algebra such as multiple right hand side [155]. Furthermore, the very recent adjustments in partitioned enlargement strategy [36] can make mutant variants of the LA Algorithm faster.

## 3.6 Experimental Results for Mutant Variants of the LA Algorithm

In this section, we report on some experiments with the mutant variants of the LA Algorithm. This section is a continuation of Section 3.3. We follow the notation and terminology defined in Section 3.3 to report on experimental results. We try to see what can be improved using mutant strategy without exploiting possible sparseness of a system. In particular, we restrict ourself to dense matrix computations. Moreover, we compare some of the timings we obtained to the straightforward Gröbner basis approach. We report experimental results using `EF` (see Section 3.3) for rank computations.

As in Section 3.3, the cryptosystems considered to construct algebraic systems of equations are HFE (Hidden Field Equations) and CTC (Courtois Toy Cipher). For more details about these cryptosystems and related algebraic systems of equations see

Section 2.3. Finally, note that the only time consuming step in the LA Algorithm is to solve linear systems. Throughout this section, we consider the timings only for calculating echelon forms. The timings for extracting linear systems are ignored, since they were not implemented efficiently and should be seen as a preprocessing step. Throughout this section we measure time in seconds unless otherwise mentioned. All timings were obtained on a computer with a 2.1 GHz AMD Opteron 6172 processor and 64GB RAM. The implementations of the algorithms of Propositions 3.4.13 and 3.5.4 are available online as a part of the ApCoCoA [12] package `charP`. For more details about implementations see Appendix B.

### 3.6.1 Experimental Results for HFE

Consider algebraic systems of equations constructed from the HFE cryptosystem. Since these systems are determined, we represent the size of each system by using the number of variables in the system. For instance, HFE(6) means an instance of HFE with 6 equations and six variables. The systems were constructed to have a unique solution. In Table 3.6 we compare the sizes of the resulting linear systems from three variations of the LA Algorithm. Note that Table 3.6 shows the size of the biggest linear system that was formed to solve a particular instance of HFE. The "$*$" in the first column for a system means that there are some mutants in this system.

| System | Equations | Variables | LA | MLA | MLA2 |
|---|---|---|---|---|---|
| HFE(6)* | 6 | 6 | 57×169 | 42×93 | 42×48 |
| HFE(7)* | 7 | 7 | 99×253 | 64×98 | 64×69 |
| HFE(8)* | 8 | 8 | 163×361 | 93×128 | 93×97 |
| HFE(9) | 9 | 9 | 256×496 | 256×441 | 130×333 |
| HFE(10) | 10 | 10 | 386×661 | 386×595 | 325×387 |
| HFE(11) | 11 | 11 | 562×859 | 562×781 | 562×756 |
| HFE(12) | 12 | 12 | 794×1093 | 794×1002 | 794×989 |
| HFE(13)* | 13 | 13 | 2380×4915 | 1093×2886 | 1093×1247 |

Table 3.6: HFE size comparison using LA, MLA and MLA2 algorithms

In Table 3.7, we collect the sizes of the resulting polynomial systems from the HFE cryptosystem over $\mathbb{F}_2$ and compare the timings for their solution with different approaches. See the results in Table 3.7. Each timing represents the total time taken by `EF` to calculate echelon forms of all the matrices during the process of solving a particular instance of HFE. The fifth column shows the time taken by the computation

| System | LA | MLA | MLA2 | GBasis |
|---|---|---|---|---|
| HFE(6)* | 0 | 0 | 0 | 0.01 |
| HFE(7)* | 0 | 0 | 0 | 0.02 |
| HFE(8)* | 0 | 0 | 0 | 0.13 |
| HFE(9) | 0 | 0 | 0 | 0.26 |
| HFE(10) | 0.05 | 0.06 | 0.04 | 0.8 |
| HFE(11) | 0.3 | 0.35 | 0.33 | 3.15 |
| HFE(12) | 0.6 | 0.6 | 0.7 | 31.24 |
| HFE(13)* | 15 | 10 | 2 | 349 |

Table 3.7: HFE time comparison using LA, MLA and MLA2 algorithms

of a `Lex` Gröbner basis in CoCoA [51].

### 3.6.2 Experimental Results for CTC

Given the CTC cryptosystem and a plaintext-ciphertex pair, we construct an overdetermined algebraic system of equations in terms of the indeterminates representing key bits and certain intermediate quantities. The task is to solve the system for the key bits. As we saw in Section 3.3, substitution of linear and quadratic equations results in an equation system in the key variables only. We present our experimental results with this level of substitution as it is more suitable for computation with dense matrices.

| System | Equations | Variables | LA | MLA | MLA2 |
|---|---|---|---|---|---|
| $CTC(2,2)_2$ | 13 | 3 | 8×14 | 8×8 | 8×8 |
| $CTC(2,3)_2^*$ | 28 | 6 | 64×197 | 64×91 | 42×61 |
| $CTC(3,2)_2^*$ | 44 | 7 | 128×352 | 128×133 | 100×128 |
| $CTC(3,3)_2^*$ | 42 | 9 | 512×1933 | 485×903 | 485×384 |
| $CTC(3,4)_2^*$ | 42 | 9 | 512×1933 | 512×2663 | 256×309 |

Table 3.8: $CTC(B,N)_2$ size comparison using LA, MLA and MLA2 algorithms

In Table 3.8, we compare the sizes of the resulting linear systems from three variations of the LA Algorithm. As usual Table 3.8 shows the size of the biggest linear system that was formed to solve $CTC(B,N)_2$. The "$*$" in the first column for a system means that there are some mutants in this system. The order of the biggest matrix to solve $CTC(3,4)_2^*$ by MLA is 512×2663. Whereas the order of the biggest matrix to solve $CTC(3,4)_2^*$ by LA is 512×1933. This is due to the reason that there are lots of mutants in this system. This issue is adjusted by MLA2.

| System | LA | MLA | MLA2 | GBasis |
|---|---|---|---|---|
| $CTC(2,2)_2$ | 0 | 0 | 0 | 0 |
| $CTC(2,3)_2^*$ | 0 | 0 | 0 | 0.09 |
| $CTC(3,2)_2^*$ | 0 | 0 | 0 | 0.14 |
| $CTC(3,3)_2^*$ | 1 | 1 | 0.11 | 0.83 |
| $CTC(3,4)_2^*$ | 0.7 | 1.5 | 0.3 | 2.99 |

Table 3.9: $CTC(B,N)_2$ time comparison using LA, MLA and MLA2 algorithms

Each timing in Table 3.9 represents the total time taken by `EF` to calculate echelon forms of all the matrices during the process of solving $CTC(B,N)_2$. The fifth column shows the time taken by the computation of a `Lex` Gröbner basis in CoCoA [51]. We see that in practice the improved mutant LA is an improvement for memory efficiency over the original mutant LA. For systems for which mutants are produced during the computation, the mutant LA is better than the LA. If no mutants occur, the mutant LA behaves identically to the LA. The improved mutant LA Algorithm is the most efficient even if there are no mutants.

Experimentally, we can conclude that the MLA2 algorithm is an improvement over the MLA algorithm. Not only can MLA2 solve multivariate systems at a lower degree than the usual LA but also can solve these systems using a smaller number of polynomials than the MLA algorithm, since we produce all possible new equations without enlarging the number of the terms. Therefore, the size of the matrix constructed by MLA2 is much smaller than the matrix constructed by MLA. This demonstrates the great potential of the mutant strategy to improve the LA Algorithm.

Chapter 4

# Techniques From the Theory of Border Bases

One of the most useful applications of border bases is to solve zero-dimensional systems of polynomial equations (see, e.g., [13, 139, 145, 117]), thus giving rise to applications in cryptography and coding theory in a natural way. Some attempts illustrating the use of border bases in cryptography and coding theory can be found in [30, 117]. Another application of border bases is the modeling of dynamic systems from measured data (see [1, 95, 119]) where the better numerical stability can be advantageous.

In this chapter we study techniques from the theory of border bases in order to solve systems of polynomial equations over finite fields. In Chapter 3, we saw some techniques to solve systems of polynomial equations over finite fields. Such techniques are used for solving systems of polynomial equations in case only a single solution exists. Although the cryptographic systems of polynomial equations have a unique solution most of the time, uniqueness is a very strong condition for general systems of polynomial equations. In the traditional literature, for systems which do not have a unique solution we are left with the theory of Gröbner bases and the theory of border bases. One advantage of border bases over Gröbner bases is that the algorithm to compute them is a linear algebra algorithm with a tiny exception, the final step, which does not contribute to the inherent complexity though as its running time is polynomial in the input size. This enables us to use linear algebra techniques of last fifty years for computing border bases. This brings to bear the full artillery of fifty years of linear algebra research on the difficulty of the problem. We consider the possibility of accelerating the Border Basis Algorithm by using techniques from linear algebra.

The theory of border bases is not as well-known as that of Gröbner bases. Border bases provide a more flexible concept than Gröbner bases and deserve further efficient implementation and experimentation for solving cryptographic systems of polynomial equations. A very recent attempt to use border bases for this purpose is done by M. Kreuzer in [117]. We build this chapter on this foundation.

In this chapter we first review the idea of computing border bases. In particular, we review the Border Basis Algorithm which is actually called the *Improved Border Basis Algorithm* in [111]. Then we propose some hybrid algorithms that combine the theory of border bases and the concept of mutants to accelerate the computation of border bases over finite fields. This enables us to exploit the use of linear algebra to accelerate the computation of border bases. On one side it enables us to study F4 [73], F5 [74] and MXL3 [143] type algorithms in the theory of border bases and on other side it provides us an accelerated way of finding solutions for those systems of polynomial equations which do not have a unique solution. Moreover, by using mutant strategies the algorithms to compute border bases can compete with the linear algebra algorithms like F4 [73] and MXL3 [143]. Finally, the efficiency of the developed techniques is examined using standard cryptographic examples.

## 4.1 Computing Border Bases

In this section we give a brief introduction to the theory of border bases and algorithms to compute them. For details and proofs we refer to the book (see [121], Section 6.4) by M. Kreuzer and L. Robbiano. For an extensive coverage of border bases we refer to [110, 111, 112]. Our exposition here follows that of [111, 117]. We are mainly interested in the case when a polynomial system is defined over a finite field $K$ and contains the field polynomials. Hence the ideal $I$ generated by the polynomial system is a zero-dimensional radical ideal.

Border bases represent a convenient way to characterize the solutions of a system of polynomial equations and can be considered as a generalization of the well-known Gröbner bases. In fact, every Gröbner basis (with respect to a degree compatible term ordering) can be extended to a border basis (see [111]) but not every border basis is an extension of a Gröbner basis. Moreover, not every order ideal supports a border basis even if it has the right cardinality. An example illustrating these two cases is presented in [111], Example 6.

While the Border Basis Algorithm in [111], which is a specification of Mourrain's

generic algorithm [145], enables us to compute border bases of zero-dimensional ideals for order ideals supported by a degree-compatible term ordering, it falls short to provide a border basis for more general order ideals: The computed border basis is supported by a reduced Gröbner basis but in certain cases it may perform better than Buchberger's algorithm (see [111], Examples 19 and 20). The alternative algorithm presented in [111], Proposition 5, which can potentially compute arbitrary border bases requires a prior knowledge of the order ideal that might support a border basis so that the order ideal has to be guessed in advance. As we cannot expect this prior knowledge, the algorithm does not solve the problem of characterizing those order ideals for which a border basis does exist. Further, as pointed out in [111], the basis transformation approach of this algorithm is unsatisfactory as it significantly relies on Gröbner basis computations.

Let $p$ be a prime number, let $q = p^e$ for some $e > 0$, and let $K = \mathbb{F}_q$ be the finite field with $q$ elements and let $f_1, \ldots, f_\ell \in P = K[x_1, \ldots, x_n]$ be a set of non-zero polynomials. We are interested in finding $K$-rational solutions of a system of polynomial equations.

$$\begin{aligned} f_1(x_1, \ldots, x_n) &= 0 \\ &\vdots \\ f_\ell(x_1, \ldots, x_n) &= 0 \end{aligned}$$

Let $F = \{f_1, \ldots, f_\ell, f_{\ell+1}, \ldots, f_m\}$, where $f_1, \ldots, f_\ell, f_{\ell+1} = x_1^q - x_1, \ldots, f_m = x_n^q - x_n \in P$. Then the ideal $I = \langle F \rangle$ is a zero-dimensional radical ideal (see Section 3.2).

**Definition 4.1.1.** Let $\mathcal{O} = \{t_1, \ldots, t_\mu\}$ be a finite set of terms in $\mathbb{T}^n$.

1) The set $\mathcal{O}$ is called an **order ideal** if $t \in \mathcal{O}$ and $t'|t$ imply $t' \in \mathcal{O}$, i.e. if $\mathcal{O}$ is closed under forming divisors.

2) The set $\partial\mathcal{O} = (x_1\mathcal{O} \cup \cdots \cup x_n\mathcal{O}) \setminus \mathcal{O}$ is called the **border** of $\mathcal{O}$.

3) Let $\partial\mathcal{O} = \{b_1, \ldots, b_\nu\}$. A set of polynomials $G = \{g_1, \ldots, g_\nu\} \subset P$ is called an **$\mathcal{O}$-border prebasis** if its elements are of the form $g_j = b_j - \sum_{i=1}^{\mu} c_{ij} t_j$ with $c_{ij} \in K$.

4) An $\mathcal{O}$-border prebasis is called an **$\mathcal{O}$-border basis** if the residue classes of the elements of $\mathcal{O}$ form a $K$-vector space basis of the ring $P/\langle g_1, \ldots, g_\nu \rangle$.

In the following we let $\mathcal{O} = \{t_1, \ldots, t_\mu\}$ be an order ideal and $\partial\mathcal{O} = \{b_1, \ldots, b_\nu\}$. Of course, we shall say that $G$ is an $\mathcal{O}$-border (pre-) basis of $I$ if $G$ is an $\mathcal{O}$-border

(pre-) basis and $G$ generates $I$. Let us collect some basic properties of $\mathcal{O}$-border (pre-) bases.

**Proposition 4.1.2.** *Let $G = \{g_1, \ldots, g_\nu\}$ be an $\mathcal{O}$-border prebasis of $I$.*

1) *Using the Border Division Algorithm (see [121], Proposition 6.4.11), we can represent every $f \in P$ in the form $f = h_1 g_1 + \cdots + h_\nu g_\nu + c_1 t_1 + \cdots + c_\mu t_\mu$ with $h_i \in P$ and $c_j \in K$.*

2) *The residue classes of the elements of $\mathcal{O}$ generate the $K$-vector space $P/I$.*

3) *If $\mathcal{O} = \mathbb{T}^n \setminus \mathrm{LT}_\sigma(I)$ for some term ordering $\sigma$, then $I$ has an $\mathcal{O}$-border basis $\widetilde{G}$. The elements of $\widetilde{G}$ corresponding to the corners of $\mathrm{LT}_\sigma(I)$ (i.e. the minimal generators of this monomial ideal) form the reduced $\sigma$-Gröbner basis of $I$ .*

4) *If $I$ has an $\mathcal{O}$-border basis, it is uniquely determined.*

*Proof.* See [121], Propositions 6.4.11 and 6.4.17. □

In general, the ideal $I$ has many more border bases than reduced Gröbner bases (see [117], Example 7.3). We will now formulate the Border Basis Algorithm (BBA), which enables us for computing border bases of zero-dimensional ideals, i.e, $P/I$ is finite dimensional with respect to an order ideal $\mathcal{O}$ induced by a degree compatible term ordering $\sigma$.

Given a system of generators $\{f_1, \ldots, f_m\}$ of $I$, the Border Basis Algorithm computes an order ideal $\mathcal{O}$ and an $\mathcal{O}$-border basis of $I$. All the computations performed by the BBA take place in a finite dimensional $K$-vector subspace $U$ of $P$ called the *computational universe*. At certain points of the algorithm the space $U$ has to be enlarged, and exactly these enlargements enable us to control the "direction" and the "speed" of the spacial growth of the computation. The next important ingredient is the following method to approximate the intersection $I \cap U$.

**Definition 4.1.3.** Let $F \subseteq U$ be two finite dimensional $K$-vector subspaces of $P$. Inductively, we define the vector subspaces $F_0 := F$ and $F_{i+1} = F^+ \cap U$ for $i = 0, 1, \ldots$, where $F_i^+ = F_i + x_1 F_i + \cdots + x_n F_i$. Then the union $F_U = \bigcup_{i \geq 0} F_i$ is called the **$U$-stable span** of $F$.

From now on we denote the set of polynomials $f_1, \ldots, f_m$ by $F$ and the $K$-vector space spanned by the polynomials $f_1, \ldots, f_m$, which should be viewed as the part of $I \cap U$, by $\langle F \rangle_K$. In the following, we explain how the $U$-stable span can be computed

explicitly for $U$. For this computation we use Gaußian elimination in the following form.

**Lemma 4.1.4. (Gaußian Elimination for Polynomials)**
*Let $\sigma$ be a degree compatible term ordering and $V = \{f_1, \ldots, f_r\} \subset P \setminus \{0\}$ be a finite set of polynomials with pairwise distinct leading terms and leading coefficients equal to 1. Let $G = \{g_1, \ldots, g_s\} \subset P$ be a finite set of polynomials. The following algorithm computes a finite set $W \subset P$ with leading coefficients equal to 1 and such that $V \cup W$ has pairwise distinct leading terms and $\langle V \cup W \rangle_K = \langle V \cup G \rangle_K$ (The set $V$ or $W$ may be empty.)*

*1) Let $H := G$ and $\varrho := 0$.*

*2) If $H = \emptyset$ then return $W := \{v_{r+1}, \ldots, v_{r+\varrho}\}$ and stop.*

*3) Choose $f \in H$ and remove it from $H$. Let $i := 1$.*

*4) If $f = 0$ or $i > r + \varrho$ then go to step 7).*

*5) If $\mathrm{LT}_\sigma(f) = \mathrm{LT}_\sigma(v_i)$ then replace $f$ with $f - \mathrm{LC}_\sigma(f) \cdot v_i$ , reset $i := 1$ and go to step 4).*

*6) Increase $i$ by 1, and go to step 4).*

*7) If $f \neq 0$ then increase $\varrho$ by 1, and put $v_{r+\varrho} := f/\mathrm{LC}_\sigma(f)$. Continue with step 2).*

*Proof.* See [111], Lemma 12. □

We can now compute the $U$-stable span using the following lemma.

**Lemma 4.1.5.** *Let $F := \{f_1, \ldots, f_r\} \in P$ and $U = \bigcup_{i=1}^{r} \mathrm{Supp}(f_i)$. Let $\sigma$ be a degree compatible term ordering. The following algorithm computes a vector basis $V$ of the stable span $F_U$, together with an updated $K$-vector subspace $U$ of $P$ called the computational universe. Moreover, the basis vectors have pairwise distinct leading terms.*

*1) Compute a vector basis $V$ of $\langle F \rangle_K$ with pairwise distinct leading terms. (Apply Lemma 4.1.4 to $V = \emptyset$ and $G := F$.)*

*2) Compute a basis extension $V \cup W'$ of $V$ such that the elements of $V \cup W'$ have pairwise different leading terms. (Apply Lemma 4.1.4 to $V$ and $G := V^+ \setminus V$.)*

*3) Let $W = \{w \in W' \mid \mathrm{LT}_\sigma(w) \in U\}$.*

*4) If* $\bigcup_{w \in W} \mathrm{Supp}(w) \not\subseteq U$*, enlarge* $U$ *by the terms in the order ideal generated by this set and continue with step 3).*

*5) If* $W \neq \emptyset$*, append* $W$ *to* $V$*, replace* $F$ *by* $F^+$*, and continue with step 2).*

*6) Return (V,U).*

*Proof.* See [111], Proposition 21. □

In our case the vector space $F$ in Definition 4.1.3 is actually $\langle F \rangle_K$ which should be viewed as the part of $I \cap U$ that we know already. By computing $F_U$, we enlarge it to produce a kind of "approximation" of $I \cap U$ . The following criterion is then the key point of the BBA.

**Proposition 4.1.6.** *Let* $U$ *be a vector subspace of* $P$*, let* $\langle F \rangle_K$ *be a vector subspace of* $I$ *which generates* $I$ *and satisfies* $F^+ \cap U = \langle F \rangle_K$*, and let* $\mathcal{O}$ *be an order ideal such that* $U = \langle F \rangle_K \oplus \langle \mathcal{O} \rangle_K$ *and* $\partial\mathcal{O} \subseteq U$*. Then* $I$ *has an* $\mathcal{O}$*-border basis which is contained in* $U$*.*

*Proof.* See [111], Proposition 16. □

The last ingredient that we need in order to formulate the border basis algorithm is the *Final Reduction Algorithm*. This algorithm applies linear algebra to interreduce the elements so that they only have support in the leading term and $\mathcal{O}$, as required by Definition 4.1.1.

**Proposition 4.1.7.** *Let* $F = \{f_1, \ldots, f_m\}$ *be a system of generators of a zero-dimensional ideal* $I$*. Let* $\sigma$ *be a degree compatible term ordering. Let* $U$ *be an order ideal. Let* $V$ *be a vector space basis of the span* $F_U$ *with pairwise different leading terms, and let* $\mathcal{O} := U \setminus \mathrm{LT}_\sigma(V)$ *such that*

$$U = F_U \oplus \langle \mathcal{O} \rangle_K \quad \text{and} \quad \partial\mathcal{O} \subseteq U.$$

*Then the following algorithm computes the* $\mathcal{O}$*-border basis* $g_1, \ldots, g_\nu$ *of* $I$*.*

*1) Let* $V_R := \emptyset$*.*

*2) If* $V = \emptyset$ *then go to step 8).*

*3) Determine in* $V$ *the element* $v$ *with minimal leading term. Remove it from* $V$*.*

*4) Let* $H := \mathrm{Supp}(v) \setminus (\{\mathrm{LT}_\sigma(v)\} \cup \mathcal{O})$.

*5) If* $H = \emptyset$ *then append* $v/\mathrm{LC}_\sigma(v)$ *to* $V_R$ *and go to step 2).*

*6) For each* $h \in H$ *determine* $w_h \in V_R$ *and* $c_h \in K$ *such that* $\mathrm{LT}_\sigma(w) = h$ *and* $h \notin \mathrm{Supp}(v - c_h.w_h)$.

*7) Replace* $v$ *with* $v - \sum_h c_h.w_h$, *append* $v/\mathrm{LC}_\sigma(v)$ *to* $V_R$ *and go step 2).*

*8) Let* $\partial\mathcal{O} = \{b_1, \ldots, b_\nu\}$. *Determine for each* $b_j \in \partial\mathcal{O}$ *the polynomial* $g_j \in V_R$ *with* $b_j = \mathrm{LT}_\sigma(g_j)$. *Return* $g_1, \ldots, g_\nu$.

*Proof.* See [111], Proposition 17. □

Finally, we are ready to formulate the Border Basis Algorithm which is actually called the *Improved Border Basis Algorithm* in [111], Proposition 21.

**Proposition 4.1.8. (The Border Basis Algorithm (BBA))**
*Let* $I = \langle f_1, \ldots, f_m \rangle$ *be a zero-dimensional ideal in* $P$ *and let* $\sigma$ *be a degree compatible term ordering. Consider the following sequence of instructions.*

*1) Let* $F := \{f_1, \ldots, f_m\}$ *and let* $U$ *be the order ideal generated by* $\bigcup_{i=1}^m \mathrm{Supp}(f_i)$.

*2) By using Lemma 4.1.5, compute a* $K$*-basis* $V$ *of the stable span* $F_U$, *together with the updated order ideal* $U$.

*3) Let* $\mathcal{O} = U \setminus \mathrm{LT}_\sigma(V)$. *If* $\partial\mathcal{O} \nsubseteq U$, *replace* $U$ *by* $U^+$ *and continue with step 2).*

*4) Apply the Final Reduction Algorithm 4.1.7 and return the set* $G = \{g_1, \ldots, g_\nu\}$ *it computes.*

*This is an algorithm which computes an order ideal* $\mathcal{O}$ *and the* $\mathcal{O}$*-border basis of* $I$.

*Proof.* See [111], Proposition 21. □

**Remark 4.1.9.** In our specified setting, where the border bases are derived from a degree-compatible term ordering $\sigma$, the border basis of a finite set of polynomials $F$ that generates a zero-dimensional ideal $I = \langle F \rangle$ contains a reduced Gröbner basis of the ideal $I$. If $G$ is the $\mathcal{O}$-border basis of $I$, then $\widetilde{G} := \{g \in G \mid \text{for all } t \in \mathbb{T}^n \text{ with } t|\mathrm{LT}_\sigma(g) \text{ we have } t \in \mathcal{O}\}$ is a reduced $\sigma$-Gröbner basis [111]. The term ordering $\sigma$ in this algorithm can be replaced by some other rules. It is merely used to guide the computation and to make sure that step 3) yields an order ideal. It is

possible to replace it by other rules guiding the computation. However, we note that in this case one has to either prove that the new rule produces an order ideal $\mathcal{O}$ or modify the computation so that it backtracks some steps if necessary.

## 4.2 Computing Border Bases With Mutant Strategies

The Border Basis Algorithm is based on the construction of further polynomials in the ideal $I$ generated by the original system. In [67, 140, 142, 143], J. Ding et al. suggested a new strategy, which is based on the concept of mutants, to speed up the construction of polynomials in the ideal $I$. The idea is that in the process of generating new polynomials, polynomials of small degree should be treated preferentially. The concept of mutants has the potential to improve various algorithms. In this section we explore the potential of their application to improve the computation of border bases. In particular, we use the concept of mutants to speed up the computation of a vector space basis which is the most time consuming part of a border basis computation. Furthermore, we see how to avoid redundancy during the computation of a vector space basis. This leads us to some hybrid techniques that combine the computation of border bases and the concept of mutants.

The Border Basis Algorithm reviewed in Section 4.1 enables us to compute border bases of zero-dimensional ideals for order ideals supported by a degree compatible term ordering. The restriction to degree compatible order ideals is due to the design of the algorithm to proceed degree-wise. The preference of border bases over Gröbner bases partly arises from the iterative generation of linear syzygies, inherent in the border basis algorithm, which enables us for successively approximating the basis degree by degree. We exploit this fact to generate some new polynomials (mutants) of lower degree in the ideal $I$ generated by the original system while avoiding a quick exhaustion of the available memory and removing redundancy. We follow the notation and terminology as standard in Section 4.1 unless mentioned otherwise.

Let $p$ be a prime number, let $q = p^e$ for some $e > 0$. Let $K = \mathbb{F}_q$ be the finite field and let $F = \{f_1, \ldots, f_m\}$ be a set of polynomials over $K$, i.e. $f_1, \ldots, f_m \in K[x_1, \ldots, x_n]$. Let $I = \langle F \rangle$ be the ideal generated by $F$. In the following we will be working over the ring

$$R = K[x_1, \ldots, x_n]/\langle x_1^q - x_1, \ldots, x_n^q - x_n \rangle,$$

where we reduce everything modulo the field polynomials. We typically generate further elements of the ideal $I$ in $R$. Recall from Section 2.2 that $R$ is the residue class ring of $K[x_1, \ldots, x_n]$ whose every element (residue class) is represented as a polynomial. Since each element (residue class) in $R$ is represented by a polynomial, we can define the degree, leading term and leading variable of this polynomial in the natural way. From now on we let $\mathbb{T}^n$ denote the set of terms of $K[x_1, \ldots, x_n]$. For $d \in \mathbb{N}$, we let $\mathbb{T}^n_{\leq d}$ denote the set of terms of $R$ with total degree at most $d$. Let $\sigma$ be a degree compatible term ordering on the terms of $R$. In this setting, mutants are defined as follows.

**Definition 4.2.1.** Let $g \in I$, i.e. $g = h_1 f_1 + \cdots + h_m f_m$, where $h_i \in R$. The level of this representation is the number $l = \max\{\deg(h_i f_i) \mid i \in \{1, \ldots, m\}, h_i \neq 0\}$. Then the **level** of $g$ with respect to $F$ is defined to be the minimal level of any representation of $g$ and we say that $g$ is a **mutant** with respect to $F$ if $\deg(g)$ is smaller than the level of $g$.

For details about the concept of mutants we refer to Section 3.4.

### 4.2.1 The Mutant Border Basis Algorithm

In [67, 140], J. Ding et al. suggested a new strategy to speed up the XL Algorithm which is based on the concept of mutants. We are going to employ the mutant strategy to speed up the Border Basis Algorithm. Consider the algorithm to compute border bases in Proposition 4.1.8. Step 2) of this algorithm computes a $U$-stable span which is the most time consuming part of the algorithm. Moreover, the complexity of the Border Basis Algorithm relies on this step. In the following proposition we explicitly explain a way to compute a $U$-stable span using the mutant strategy while avoiding a quick exhaustion of the available memory and removing redundancy.

**Proposition 4.2.2.** *Let $F := \{f_1, \ldots, f_r\} \subseteq R$ and $U = \bigcup_{i=1}^{r} \operatorname{Supp}(f_i)$. Let $V$ be a $K$-basis of $\langle F \rangle_K$ with pairwise distinct leading terms. Let $\sigma$ be a degree compatible term ordering. Consider the following sequence of instructions.*

*S1) Let $d_{max} := \max\{\deg(v) \mid v \in V\}$, $d_{min} := \min\{\deg(v) \mid v \in V\}$, $P_{mutant} := \emptyset$, and let $G := V$.*

*S2) Let $H := \emptyset$, and for each $g \in G \setminus P_{mutant}$ of degree $d_{min}$ append $g$ to $P_{mutant}$ and all possible products $x_\alpha g$ with $\alpha \in \{1, \ldots, n\}$ to $H$. Let $d_{elim} := d_{min} + 1$.*

*S3) Apply Lemma 4.1.4 to $G$ and $H$ to compute a basis extension $H'$ for $\langle G_{\leq d_{min}}\rangle_K \subseteq \langle G^+_{\leq d_{min}}\rangle_K$ so that the elements of $H' \cup G$ have pairwise distinct leading terms. Then replace $G$ with $G \cup H'$.*

*S4) Form the set $M$ of all polynomials of degree less than $d_{elim}$ in $G$. (The polynomials in the set $M \setminus P_{mutant}$ are mutants with respect to $V$.)*

*S5) Let $H := \emptyset$. If $M \setminus P_{mutant} \neq \emptyset$, then for each $m \in M \setminus P_{mutant}$ append $m$ to $P_{mutant}$ and all possible products $x_\alpha m$ with $\alpha \in \{1, \ldots, n\}$ to $H$. Let $d_{elim} := \min\{\deg(m) \mid m \in M \setminus P_{mutant}\} + 1$. Then continue with step S3).*

*S6) If $d_{elim} \leq d_{min}$, then increase $d_{elim}$ by one and continue with step S3).*

*S7) Let $W' := G \setminus V$. ($W'$ is a basis extension of $V$ such that the elements of $V \cup W'$ have pairwise distinct leading terms.)*

*S8) Let $W := \{w \in W' \mid \mathrm{LT}_\sigma(w) \in U\}$.*

*S9) If $\bigcup_{w \in W} \mathrm{Supp}(w) \not\subseteq U$, enlarge $U$ by the terms in the order ideal generated by this set and continue with step 7).*

*S10) If $W \neq \emptyset$, append $W$ to $V$.*

*S11) If $d_{min} < d_{max}$ then increase $d_{min}$ by one and continue with step S2).*

*S12) Return the result $(G, V, U, P_{mutant})$.*

*This is an algorithm which returns a vector basis $V$ of the stable span $F_U$, together with the order ideal $U$, a $K$-basis $G$ that contains $V$ of $\langle F\rangle_K$ and the set $P_{mutant}$ to keep record of mutants. Furthermore, the basis vectors of $V$ have pairwise distinct leading terms.*

*Proof.* Let $L' = \mathbb{T}^n_{\leq d_{min}}$ and $L = \mathbb{T}^n_{\leq d_{max}}$. Starting from the minimum possible $d_{min}$, each iteration of the loop of steps S2)−S11) computes a $L'$-stable span of $G_{\leq d_{min}}$ in the following way.

Step S2) starts with a set $G$ with pairwise different leading terms. So the loop is correctly initialized. Then step S2) computes a set $H$ by multiplying all polynomials of degree $d_{min}$ in $G$ with indeterminates. Note that $H = G^+_{\leq d_{min}} \setminus G_{\leq d_{min}}$ and it helps to get rid of redundant polynomials. Now consider the loop of steps S3)−S5). By Lemma

4.1.4, step S3) computes a finite set $H'$ such that $G \cup H'$ has pairwise different leading terms and

$$\langle G_{\leq d_{min}} \rangle_K \subseteq \langle G_{\leq d_{min}} \cup H'_{\leq d_{min}} \rangle_K = \langle G^+_{\leq d_{min}} \rangle_K \cap L' \subseteq L'.$$

In particular, $G_{\leq d_{min}} \cup H'_{\leq d_{min}}$ is a vector basis of $\langle G^+_{\leq d_{min}} \rangle_K \cap L'$. Then steps S4) and S5) compute a set $H$ using mutants (new polynomials of degree less than or equal to $d_{elim} = d_{min} + 1$). Again note that $H = G^+_{\leq d_{min}} \setminus G_{\leq d_{min}}$ and it helps to get rid of redundant polynomials. In this way the loop of steps S3)−S5) is repeated until there are no more mutants. The loop of steps S3)−S6) serves as a safeguard to ensure that the elimination degree $d_{elim}$ is equal to $d_{min} + 1$ when the process of finding mutants terminates.

Another iteration is called in step S11) if and only if $d_{min} < d_{max}$. If $d_{min} \geq d_{max}$, the loop of steps S2)−S11) terminates. After termination we have $\langle G_{\leq d_{max}} \rangle_K = \langle G_{\leq d_{max}} \rangle^+_K \cap L$ which is exactly the $L$-stable span of $G$. This proves the correctness of computing a $L$-stable span. The loop of steps S7)−S9) along with step S10) computes a $U$-stable span of $V$ from the $L$-stable span of $G$. The correctness of this part follows from [111], Proposition 21. □

**Remark 4.2.3.** Assume that we are in the setting of the algorithm of Proposition 4.2.2. If we skip the loop of steps S7)−S9) and step S10), the algorithm computes the $L$-stable span of $G$. The $L$-stable span of $G$ can be used to accelerate the initial version of the Border Basis Algorithm given in [111], Proposition 18. The algorithm in its original form computes a $U$-stable span of $V$ from the $L$-stable span of $G$. This leads us to the following question. Can there be an algorithm that computes a $U$-stable span of $V$ without considering the $L$-stable span of $G$?

The answer is positive but it will require computing the same polynomials in the ideal $I$ again and again. This restriction is due to the design of the Border Basis Algorithm to proceed degree-wise. Furthermore, mutants can be found effectively if we generate the ideal $I$ degree-wise. Actually, using $G$ we reduce the work for the next iterations by saving the polynomials in $H'$ because it is quite likely that they can be reused in the next iterations. If the polynomials in $G$ are wasted, we may need to do the same work in the next iterations again.

**Remark 4.2.4.** During the computation of a $U$-stable span we create a multiplication history to get rid of redundancy. For instance, if we are working over the field $\mathbb{F}_2$,

redundancy can be avoided as follows. During the multiplication process we keep the multiplier variable that gives rise to every new produced polynomial and we keep one for the original polynomials. While computing $F_i^+$, we multiply the polynomial $f \in F_i$ by all variables smaller than its previous multiplier variable. In case of the previous multiplier of $f$ is one, we multiply by all variables. The target of this selection method is to speed up the extension process of a $U$-stable span. We multiply by terms of degree one (variables or indeterminates) only without any trivial redundancy.

Now we are ready to assemble a variant of the Border Basis Algorithm that uses mutant strategy to accelerate the computation of a $U$-stable span. In the setting of the following proposition we apply Proposition 4.2.2 without step S1) because this step is settled by the proposition itself.

**Proposition 4.2.5. (The Mutant Border Basis Algorithm (MBBA))**
*Let $F = \{f_1, \ldots, f_m\} \subset R$ be a set of polynomials that generates a zero-dimensional ideal $I = \langle F \rangle$ in $R$. Let $\sigma$ be a degree compatible term ordering. Consider the following sequence of instructions.*

*1) Let $U$ be the order ideal generated by $\bigcup_{i=1}^{m} \mathrm{Supp}(f_i)$.*

*2) Interreduce $F$ to get a $K$-basis $V$ of $\langle F \rangle_K$ with pairwise distinct leading terms.*

*3) Let $d_{max} := \max\{\deg(v) \mid v \in V\}$, $d_{min} := \min\{\deg(v) \mid v \in V\}$, $G := V$ and let $P_{mutant} := \emptyset$.*

*4) Apply Proposition 4.2.2 to compute a $K$-basis $V$ of the stable span $F_U$, the updated $K$-basis $G$, the updated order ideal $U$ and the updated set $P_{mutant}$.*

*5) Let $\mathcal{O} = U \setminus \mathrm{LT}_\sigma(V)$.*

*6) If $\partial\mathcal{O} \not\subseteq U$, replace $U$ by $U^+$ and let $d_{min} := d_{max} := \max\{\deg(u) \mid u \in U\}$. Then continue with step 4).*

*7) Apply the Final Reduction Algorithm 4.1.7.*

*This is an algorithm which computes an order ideal $\mathcal{O}$ and the $\mathcal{O}$-border basis of $I$.*

*Proof.* To show that the procedure terminates and that the algorithm is correct, we consider its steps which differ from the algorithm in [111], Proposition 21. Step 1) initializes $U$ so that $F \subseteq U$. Step 2) computes a vector basis $V$ of $\langle F \rangle_K$ with pairwise

different leading terms. Since we plan to proceed degree-wise, step 3) initializes the degree bound $d_{max}$ on the degrees of the polynomials in $V$, and the minimum degree $d_{min}$ of a polynomial (not processed) in $G$. To find mutants and reduce redundancy $d_{max}$ is initialized to be empty.

Now consider step 4). Each new iteration in step 4) starts with a universe $U$, a vector basis $V$ with pairwise different leading terms of the stable span $F_U$, a vector basis $G$ of $F$ bounded by the degree $d_{max}$ and the set $P_{mutant}$. Note that $G$ serves to reduce the work for the next iterations by saving all the polynomials produced during the computation of the universe in step 4) because it is quite likely that they can be reused in the next iterations. Applying Proposition 4.2.2, we see that step 4) computes an updated vector basis $V$ of $F_U$ and an updated universe $U$. This covers all changes in the algorithm and concludes the proof. □

The set of polynomials $P_{mutant}$ is used to recognize mutants. It can be replaced by some other method that can recognize mutants. An alternate way could be to use a flag with each polynomial. After checking a polynomial to be a mutant this flag will be turned off. To understand Proposition 4.2.5 better, we now apply it in a concrete case.

**Example 4.2.6.** Over the field $K = \mathbb{F}_2$, consider the ideal $I = \langle f_1, f_2, f_3 \rangle$ in $K[x, y, z]$, where $f_1 = xy + xz + 1$, $f_2 = xz + yz + z$, and $f_3 = xy + xz + y + 1$. Let $\sigma = \texttt{DegLex}$. To get rid of redundancy we store each polynomial $f_i$ as a tuple $(m, f_i)$, where $m$ is a variable multiplier. We will we working over the ring $K[x, y, z]/\langle x^2 - x, y^2 - y, z^2 - z \rangle$. Let us follow the steps of the MBBA.

1) We have $U = \{xy, xz, yz, y, z, 1\}$.

2) Interreduction yields the basis $V = \{(1, xy + xz + 1), (1, xz + yz + z), (1, y)\}$.

3) Let $d_{max} := 2$, $d_{min} := 1$, $G := V$, and $P_{mutant} := \emptyset$.

4) Consider the following steps to compute a $U$-stable span due to Proposition 4.2.2.

   S2) Let $P_{mutant} := \{(1, y)\}$, $d_{elim} := 2$, and $H := \{(x, xy), (y, y), (z, yz)\}$.

   S3) We have $H' = \{(x, yz + z + 1), (z, z + 1)\}$ and $G = \{(1, xy + xz + 1), (1, xz + yz + z), (1, y), (x, yz + z + 1), (z, z + 1)\}$.

   S4) $M = \{y, z + 1\}$.

S5) We have $M \setminus P_{mutant} = \{z+1\}$. Let $P_{mutant} := \{y, z+1\}$, $H := \{(x, xz+x), (y, yz+y)\}$, and $d_{elim} := 2$.

S3) We have $H' = \{(x, x+1)\}$ and $G = \{(1, xy+xz+1), (1, xz+yz+z), (1, y), (x, yz+z+1), (z, z+1), (x, x+1)\}$.

S4) $M = \{y, z+1, x+1\}$.

S5) We have $M \setminus P_{mutant} = \{x+1\}$. Let $P_{mutant} := \{y, z+1, x+1\}$, $H := \{(y, xy+y), (z, xz+z)\}$, and $d_{elim} := 2$.

S3) $H' = \emptyset$.

S4) $M = \{y, z+1, x+1\}$.

S5) $M \setminus P_{mutant} = \emptyset$

S6) $d_{elim} = 2 = d_{min} + 1$.

S7) $W' = \{(x, yz+z+1), (z, z+1), (x, x+1)\}$.

S8) $W = \{(x, yz+z+1), (z, z+1)\}$.

S9) We have $\bigcup_{w \in W} \mathrm{Supp}(w) \subseteq U$.

S10) $V = \{(1, xy+xz+1), (1, xz+yz+z), (1, y), (x, yz+z+1), (z, z+1)\}$.

S11) Since $d_{min} = 1 < d_{max} = 2$, increase $d_{min}$ by one and continue with step 2).

S2) Let $P_{mutant} := \{(1, xy+xz+1), (1, xz+yz+z), (1, y), (x, yz+z+1), (z, z+1), (x, x+1)\}$, $d_{elim} := 3$, and $H := \{(x, xy+xz+x), (y, xyz+xy+y), (z, xyz+xz+z), (x, xyz), (y, xyz), (z, xz+yz+z), (y, y), (z, yz)\}$.

S3) We have $H' = \{(y, xyz+xy+y)\}$ and $G = \{(1, xy+xz+1), (1, xz+yz+z), (1, y), (x, yz+z+1), (z, z+1), (x, x+1), (y, xyz+xy+y)\}$.

S4) $M = \{xy+xz+1, xz+yz+z, y, yz+z+1, z+1, x+1\}$.

S5) $M \setminus P_{mutant} = \emptyset$

S6) $d_{elim} = 3 = d_{min} + 1$.

S7) $W' = \{(y, xyz+xy+y)\}$.

S8) $W = \emptyset$.

S9) We have $\bigcup_{w \in W} \mathrm{Supp}(w) \subseteq U$.

S10) $V = \{(1, xy+xz+1), (1, xz+yz+z), (1, y), (x, yz+z+1), (z, z+1)\}$.

S11) $d_{min} = 2 = d_{max}$.

S12) Return $(G, V, U, d_{max})$.

5) $\mathcal{O} = \{1\}$.

6) Since $\partial\mathcal{O} = \{x, y, z\} \not\subseteq U$, let $U = \{xyz, xy, xz, yz, x, y, z, 1\}$ and continue with step 4).

4) Consider the following steps to compute a $U$-stable span due to Proposition 4.2.2.

S2) Let $P_{mutant} := \{(1, xy+xz+1), (1, xz+yz+z), (1, y), (x, yz+z+1), (z, z+1), (x, x+1), (y, xyz+xy+y)\}$, $d_{elim} := 4$, and $H := \{(z, yz)\}$.

S3) $H' = \emptyset$.

S4) $M = \{(1, xy+xz+1), (1, xz+yz+z), (1, y), (x, yz+z+1), (z, z+1), (x, x+1), (y, xyz+xy+y)\}$.

S5) $M \setminus P_{mutant} = \emptyset$.

S6) $d_{elim} = 4 = d_{min} + 1$.

S7) $W' = \{(y, xyz+xy+y)\}$.

S8) $W = \{(y, xyz+xy+y)\}$.

S9) We have $\bigcup_{w\in W} \mathrm{Supp}(w) \subseteq U$.

S10) $V = \{(1, xy+xz+1), (1, xz+yz+z), (1, y), (x, yz+z+1), (z, z+1), (y, xyz+xy+y)\}$.

S11) $d_{min} = 3 = d_{max}$.

S12) Return $(G, V, U, d_{max})$.

5) $\mathcal{O} = \{1\}$.

6) $\partial\mathcal{O} = \{x, y, z\} \subseteq U$.

7) The Final Reduction Algorithm 4.1.7 returns $\{x+1, y, z+1\}$.

Thus we see that $(1, 0, 1)$ is the only zero of $I$.

**Remark 4.2.7.** Note that in Example 4.2.6 the maximum degree of a term in the universe $U$ is 3 and we need to compute with polynomials of maximum degree 4. In this example we have extended $U$ by considering the support of polynomials in $W$ as given in Proposition 4.2.5. But if we choose $U = \mathbb{T}^2_{\leq d_{max}}$ as given by the basic version of the Border Basis Algorithm (see [111], Proposition 18), we can end with a universe

in which the maximum degree of a term is 2 and we need to compute with polynomials of maximum degree 3.

### 4.2.2 The Improved Mutant Border Basis Algorithm

The experiments with the mutant strategy [142] show that there are two issues. The first arises when the number of lower degree mutants is very large, as this produces many reductions to zero. The second arises when an iteration does not produce mutants at all or produces only an insufficient number of mutants at a lower degree. In later case the mutant strategy makes no improvement. To handle both problems an improved mutant strategy was introduced in [142]. This strategy is based on the mutant concept, however it introduces a heuristic strategy of only choosing the minimum number of mutants, which is called *necessary mutants*. For details about the improved mutant strategy we refer to Section 3.5. In the following we employ the improved mutant strategy to speed up the Border Basis Algorithm.

Let $X = \{x_1, \ldots, x_n\}$ be the set of variables, upon which we impose the following order:

$$x_1 > x_2 > \cdots > x_n$$

Our first goal is to recall a few definitions and concepts to be used later. Recall from Section 2.2, the ring $R$ is the residue class ring of $K[x_1, \ldots, x_n]$. The elements of $R$ are residue classes where each residue class has a unique polynomial representation. Every element $f \in R \setminus \{0\}$ has a unique representation as a linear combination of terms

$$f = \sum_{i=0}^{s} c_i t_i$$

where $c_i \in K$, $t_i \in \mathbb{T}^n$ is a term in $R$, i.e. $t_i = x_1^{\alpha_1} \ldots x_n^{\alpha_n}$, such that $\alpha_i < q$, $1 \leq i \leq n$, and where $t_1 >_\sigma t_2 >_\sigma \cdots >_\sigma t_s$. Considering this representation we can define the degree, leading term and leading coefficient in the natural way.

**Definition 4.2.8.** Let $p \in R$. Then the **leading variable** of $p$ is the largest variable, according to the order defined on the variables, in the leading term of $p$ and denoted by $\mathrm{LV}(p)$.

**Definition 4.2.9.** Let $F_d = \{f \in F \mid \deg(f) = d\}$ and let $x_i \in X$. We define $F_d^{x_i}$ as follows.

$$F_d^{x_i} = \{f \in F_d \mid \mathrm{LV}(f) = x_i\}$$

As described above the complexity of the Border Basis Algorithm relies on the computation of a $U$-stable span. We employ the improved mutant strategy to improve its computation.

**Remark 4.2.10.** The computation of the $U$-stable span in Proposition 4.2.2 using the improved mutant strategy can be achieved as follows. Replace step 1) in the algorithm of Proposition 4.2.2 by the following instruction.

1') Let $d_{max} := \max\{\deg(v) \mid v \in V\}$, $d_{min} := \min\{\deg(v) \mid v \in V\}$, $P_{mutant} := \emptyset$, $G := V$, and let $X' := \emptyset$.

Replace step 2) in the algorithm of Proposition 4.2.2 by the following two instructions.

2a) If $X' \neq \emptyset$ then choose the smallest variable $x_l \in X'$ and delete it from $X'$. Otherwise, let $X' := \{\mathrm{LV}(g) \mid g \in G \text{ and } \deg(g) = d_{min}\}$, and let $d_{elim} := d_{min}+1$. Choose the smallest variable $x_l \in X'$ and delete it from $X'$.

2b) Let $H := \emptyset$. For each $g \in \{g \mid g \in G \setminus P_{mutant} \text{ and } g \in G_{d_{min}}^{x_l}\}$ append $g$ to $P_{mutant}$ and all possible products $x_\alpha g$ with $\alpha \in \{1, \ldots, n\}$ to $H$.

Replace step 5) in the algorithm of Proposition 4.2.2 by the following instruction.

5') Let $H := \emptyset$. If $M \setminus P_{mutant} \neq \emptyset$, then for each *necessary mutant* $m \in M \setminus P_{mutant}$ append $m$ to $P_{mutant}$ and all possible products $x_\alpha m$ with $\alpha \in \{1, \ldots, n\}$ to $G$. Let $d_{elim} := \min\{\deg(m) \mid m \in M \setminus P_{mutant}\} + 1$. Then continue with step 3).

The resulting algorithm still computes a vector basis $V$ of the stable span $F_U$, together with the order ideal $U$ and a $K$-basis $G$ that contains $V$ of $\langle F \rangle_K$. In general, it keeps the size of the computational universe $U$ even smaller than the algorithm in Proposition 4.2.2, but it may require more iterations. In other words, we are sacrificing time efficiency for gaining space efficiency. But experiments with the improved mutant strategy show that it provides space efficiency as well as time efficiency (see [142] and Section 4.3).

**Proposition 4.2.11. (The Improved Mutant BBA (MBBA2))**
*Let $F = \{f_1, \ldots, f_m\} \subset R$ be a set of polynomials that generates a zero-dimensional ideal $I = \langle F \rangle$ in $R$. Let $\sigma$ be a degree-compatible term ordering. Consider the following sequence of instructions.*

*1) Let $U$ be the order ideal generated by $\bigcup_{i=1}^{m} \mathrm{Supp}(f_i)$.*

*2) Interreduce $F$ to get a $K$-basis $V$ of $\langle F\rangle_K$ with pairwise distinct leading terms.*

*3) Let $d_{max} := \max\{\deg(v) \mid v \in V\}$, $d_{min} := \min\{\deg(v) \mid v \in V\}$, $X' := \emptyset$, $G := V$ and let $P_{mutant} := \emptyset$.*

*4) Apply Proposition 4.2.2 along with the modifications given by Remark 4.2.10 to compute a $K$-basis $V$ of the stable span $F_U$, the updated $K$-basis $G$, the updated order ideal $U$ and the updated sets $P_{mutant}$ and $X'$.*

*5) Let $\mathcal{O} = U \setminus \mathrm{LT}_\sigma(V)$.*

*6) If $X' \neq \emptyset$ and $\partial\mathcal{O} \not\subseteq U$, then continue with step 4).*

*7) If $\partial\mathcal{O} \not\subseteq U$, replace $U$ by $U^+$, and let $d_{min} := d_{max} := \max\{\deg(u) \mid u \in U\}$. Then continue with step 4).*

*8) Apply the Final Reduction Algorithm 4.1.7.*

*This is an algorithm which computes an order ideal $\mathcal{O}$ and the $\mathcal{O}$-border basis of $I$.*

*Proof.* The proof follows from Proposition 4.2.5. The only exception is that in step 4) we compute a $K$-basis $V$ of $F_U$ in parts. But this does not affect the correctness of the algorithm. □

**Remark 4.2.12.** Assume that we are in the setting of the algorithm in Proposition 4.2.11. Since $G$, $P_{mutant}$, $d_{min}$, $X'$ and $d_{max}$ are initialized by the algorithm itself, we skip step 1') of the algorithm of Remark 4.2.10 each time it is called to compute a $U$-stable span. The set of polynomials $P_{mutant}$ is used to recognize mutants.

Since we are mostly looking for $K$-rational solutions and our ideal $I$ is a zero-dimensional radical ideal, we can take advantage of the following proposition and theorem to choose necessary mutants and to reduce some iterations that produce reductions to zero.

**Proposition 4.2.13.** *Let $f_1, \ldots, f_m \in P$ be polynomials which generate a zero-dimensional ideal $I = \langle f_1, \ldots, f_m\rangle$. Then the system of equations*

$$f_1(x_1, \ldots, x_n) = 0, \ldots, f_m(x_1, \ldots, x_n) = 0$$

*has at most $\dim_K(P/I)$ solutions in $\overline{K}^n$.*

*Proof.* See [120], Proposition 3.7.5. □

The following theorem tells us the exact number of solutions in case the ideal $I$ in Proposition 4.2.13 is a zero-dimensional radical ideal.

**Theorem 4.2.14.** *Let $f_1, \ldots, f_m \in P$ be polynomials which generate a zero-dimensional radical ideal $I = \langle f_1, \ldots, f_m \rangle$. Let $\overline{K}$ be the algebraic closure of $K$, and let $\overline{P} = \overline{K}[x_1, \ldots, x_n]$. If $K$ is a perfect field, the number of solutions of the system of equations*

$$f_1(x_1, \ldots, x_n) = 0, \ldots, f_m(x_1, \ldots, x_n) = 0$$

*is equal to the number of maximal ideals of $\overline{P}$ containing $I\overline{P}$, and this number is precisely $\dim_K(P/I)$.*

*Proof.* See [120], Theorem 3.7.19. □

In most of the cases we may know the exact number of $K$-rational solutions, thus we know $\dim_K(P/I)$.

**Remark 4.2.15.** Assume that we are in the setting of Proposition 4.2.11. Since $U$ is a vector space and $V \subseteq U$ is a vector subspace we can form the vector space quotient $U/V$ and we know that

$$\dim_K(U/V) = \dim_K U - \dim_K V.$$

In our case the ideal is a zero-dimensional radical ideal and we may know the exact number of solutions, especially when the system defined by $F$ has a unique solution. This means that we know $\dim_K(U/V)$. Thus in practice, we can check $\dim_K U - \dim_K V$ during the course of the algorithm. If at a certain stage the difference $\dim_K U - \dim_K V$ equals $\dim_K(U/V)$, the $U$-stable span has been reached and further iterations are producing reductions to zero. This also helps to choose necessary mutants.

### Effectiveness of Mutant Strategies for Computing a $U$-stable span

The complexity of the Border Basis Algorithm relies on the computation of a vector space basis. In Proposition 4.2.2 and Remark 4.2.10 we employed the mutant strategy and the improved mutant strategy respectively to speed up its computation. A natural question could be to ask about the effectiveness of the mutant strategies to compute a vector space basis.

Surprisingly, it turns out that there are deep connections to other mathematical disciplines, and border bases that represent the combinatorial structure of the ideal under consideration in a canonical way. In fact the XL Algorithm (see [59, 117]), which is based on relinearization methods, in its classical form is actually identical to a $L$-stable span and therefore it is a border basis computation at its core. Recently developed MutantXL [140], MXL2 [142] and MXL3 [143] algorithms are improved versions of the XL Algorithm. This leads us to the following observation.

Assume that we are working over the finite field $\mathbb{F}_2$ and using an adapted version of M4RI [4], a library for dense matrix linear algebra over $\mathbb{F}_2$. An algorithm to compute a vector space basis using mutant strategies can perform substantially better than the F4 algorithm [73] implemented in Magma [32], currently the best publicly available implementation of F4. This can be observed from experimental results for the MutantXL, the MXL2 and the MXL3 algorithms presented in [37, 142, 143]. Furthermore, a complexity analysis of these algorithms is given in [141], which suggests a new upper bound for the complexity of computing Gröbner bases. This complexity analysis also supports our arguments.

**Future Work**

In the spirit of the algorithms developed in this section, it is obviously possible to generate a number of further variations of the Border Basis Algorithm. Such variations may have the potential to speed up the computation of border bases considerably while at the same time avoiding a quick exhaustion of the available memory. In this sense, the flexible partial enlargement strategy introduced in [36] can definitely improve the computation of a $U$-stable span. The flexible partial enlargement strategy is a very recent idea and surpasses old boundaries set by mutant strategies.

As a final remark, we point out that by using mutant strategies the algorithms to compute a border basis can compete with the linear algebra algorithms such as F4 [73] and MXL3 [143] which are used for the computation of Gröbner bases. Actually, the idea of mutants is more suitable for border bases as compared to Gröbner bases. The preference arises from the iterative generation of linear syzygies, inherent in the border basis algorithm, which allows for successively approximating the basis degree by degree. We point out that the computation of a $U$-stable span can be found with the help of all the sparse linear algebra techniques which lie at the heart of linear algebra. Therefore, the Border Basis Algorithm is able to absorb several techniques for speed up and deserves further efficient implementation and experimentation.

## 4.3 Experimental Results

In this section we report on some experiments with the mutant variants of the Improved Border Basis Algorithm. These experimental results show that even for small instances of systems of polynomial equations we can sufficiently reduce the size of the problem using mutant strategies. The mutant strategies provide us space efficiency as well as time efficiency. We restrict ourself to dense matrix computations. We compare some of the timings we obtained to the straightforward Gröbner basis approach. A preliminary implementation of the Border Basis Algorithm is available in the ApCoCoA [12] library for general zero-dimensional ideals. We also consider this implementation to compare some of the timings we obtained.

To fulfil the needs of Lemma 4.1.4 we use a self implemented code to perform Gaußian elimination over $\mathbb{F}_2$. This implementation is denoted by `EF` in Section 3.3. It is also available in the ApCoCoA [12] package `linalg`. Using this implementation we compare the timings and sizes we obtained with the BBA, the MBBA and the MBBA2.

We consider the HFE (Hidden Field Equations) and the CTC (Courtois Toy Cipher) to construct algebraic systems of equations. For more details about these cryptosystems and related algebraic systems of equations see Section 2.3. Throughout this section we consider the timings only for Gaußian elimination. The timings for preparing matrices, computing order ideals and their borders and the Final Reduction Algorithm are ignored, since they were not implemented efficiently and should be seen as a preprocessing step. Throughout this section we measure time in seconds unless otherwise mentioned. All timings were obtained on a computer with a 2.1 GHz AMD Opteron 6172 processor and 64GB RAM.

### 4.3.1 Experimental Results for CTC

Given the CTC cryptosystem and a plaintext-ciphertex pair, we construct an overdetermined algebraic system of equations in terms of the indeterminates representing key bits and certain intermediate quantities. The task is to solve the system for the key bits. As we saw in Section 3.3, substitution of linear and quadratic equations results in an equation system in the key variables only. We present our experimental results with this level of substitution as it is more suitable for computations with dense matrices. These systems are such that each of them has a unique solution over $\mathbb{F}_2$.

| | Universe | | BBA | MBBA | | MBBA2 | |
|---|---|---|---|---|---|---|---|
| System | $\mathcal{O}$ | V | Matrix | G | Matrix | G | Matrix |
| CTC(2,2)$^*_2$ | 8 | 7 | 25×8 | 7 | 15×8 | 7 | 13×8 |
| CTC(2,3)$^*_2$ | 64 | 63 | 436×64 | 63 | 230×64 | 63 | 79×64 |
| CTC(3,2)$^*_2$ | 128 | 127 | 1013×128 | 127 | 433×128 | 127 | 242×128 |
| CTC(3,3)$^*_2$ | 492 | 491 | 4908×511 | 501 | 1757×485 | 510 | 1428×488 |
| CTC(3,4)$^*_2$ | 512 | 511 | 5107×512 | 511 | 2662×512 | 511 | 564×512 |
| CTC(4,3)$^*_2$ | 512 | 511 | - | 2046 | 3603×2036 | 1984 | 1984×2046 |

Table 4.1: CTC size comparison using BBA, MBBA and MBBA2

| System | Equations | Variables | BBA | MBBA | MBBA2 | BBasis | GBasis |
|---|---|---|---|---|---|---|---|
| CTC(2,2)$^*_2$ | 13 | 3 | 0 | 0 | 0 | 0 | 0 |
| CTC(2,3)$^*_2$ | 28 | 6 | 0 | 0 | 0 | 0.29 | 0.09 |
| CTC(3,2)$^*_2$ | 44 | 7 | 0.02 | 0 | 0 | 3.14 | 0.14 |
| CTC(3,3)$^*_2$ | 42 | 9 | 1.7 | 0.75 | 0.5 | - | 0.83 |
| CTC(3,4)$^*_2$ | 42 | 9 | 1.8 | 0.9 | 0.1 | - | 2.99 |
| CTC(4,3)$^*_2$ | 53 | 11 | - | 20 | 7 | - | 128 |

Table 4.2: CTC time comparison using BBA, MBBA and MBBA2

The "$*$" in the first column for a system means that there are some mutants in this system. In Table 4.1 we collect the sizes of universes and the biggest sizes of matrices formed by Lemma 4.1.4. We can see the results in Table 4.3 agree with the results in Table 4.1. In Table 4.2 we compare the timings for the solutions of CTC instances. Each timing represents the total time taken by Lemma 4.1.4. The last two columns show the time taken by the computation of a `Lex` Gröbner basis and a `DegLex` border basis in ApCoCoA [12] respectively.

### 4.3.2 Experimental Results for HFE

Consider algebraic systems of equations constructed from the HFE cryptosystem. Since these systems are determined, we represent the size of each system by using the number of variables in the system. The "$*$" in the first column for a system means that there are some mutants in this system.

| | Universe | | BBA | MBBA | | MBBA2 | | |
|---|---|---|---|---|---|---|---|---|
| System | $\mathcal{O}$ | V | Matrix | G | Matrix | G | Matrix | Sol. |
| HFE(6)* | 22 | 21 | 143×42 | 41 | 92×42 | 41 | 47×42 | 1 |
| | 22 | 20 | 140×42 | 40 | 95×42 | 40 | 51×42 | 2 |
| HFE(7)* | 29 | 28 | 223×64 | 63 | 97×64 | 63 | 68×64 | 1 |
| | 29 | 25 | 200×64 | 60 | 116×64 | 59 | 71×64 | 4 |
| HFE(8)* | 36 | 35 | 314×93 | 92 | 259×93 | 92 | 95×93 | 1 |
| | 37 | 34 | 305×93 | 90 | 191×93 | 90 | 95×93 | 3 |
| HFE(9) | 130 | 129 | 1290×256 | 255 | 695×256 | 199 | 402×200 | 1 |
| | 130 | 128 | 1277×254 | 254 | 685×256 | 184 | 185×241 | 2 |
| HFE(10) | 176 | 175 | 1925×386 | 385 | 1144×386 | 315 | 386×325 | 1 |
| | 176 | 174 | 1913×386 | 384 | 1133×386 | 314 | 386×325 | 2 |
| HFE(11) | 232 | 231 | 2771×562 | 561 | 1781×562 | 561 | 677×562 | 1 |
| | 232 | 229 | 2746×562 | 559 | 1408×562 | 559 | 935×562 | 3 |
| HFE(12) | 299 | 298 | 3873×794 | 793 | 2652×794 | 793 | 879×794 | 1 |
| | 299 | 297 | 3861×794 | 792 | 2640×794 | 792 | 879×794 | 2 |
| HFE(13)* | 378 | 377 | 5276×1093 | 1092 | 2885×1093 | 1092 | 1195×1093 | 1 |
| | 378 | 372 | 5276×1093 | 1087 | 2885×1093 | 1087 | 1195×1093 | 6 |
| HFE(14)* | 470 | 469 | 7035×1471 | 1470 | 4032×1471 | 1471 | 1483×1471 | 1 |

Table 4.3: HFE size comparison using BBA, MBBA and MBBA2

In Table 4.3 we collect the sizes of universes and the biggest sizes of matrices formed by Lemma 4.1.4. In all the three cases the size of a universe remains the same. The MBBA is an improvement of the BBA if there are mutants in the system HFE(N). Since we are also getting rid of redundancy in the MBBA, we see some improvement in sizes even for those systems which do not have mutants. If no mutants occur and we do not get rid of redundancy, the MBBA behaves identically to the BBA.

| System | Equations | Variables | BBA | MBBA | MBBA2 | BBasis | GBasis |
|---|---|---|---|---|---|---|---|
| HFE(6)* | 6 | 6 | 0 | 0 | 0 | 0.04 | 0.01 |
| | 6 | 6 | 0 | 0 | 0 | 0.04 | 0 |
| HFE(7)* | 7 | 7 | 0 | 0 | 0 | 0.09 | 0.02 |
| | 7 | 7 | 0 | 0 | 0 | 0.1 | 0.01 |
| HFE(8)* | 8 | 8 | 0 | 0 | 0 | 0.26 | 0.13 |
| | 8 | 8 | 0 | 0 | 0 | 0.28 | 0.04 |
| HFE(9) | 9 | 9 | 0.1 | 0.03 | 0 | 5.11 | 0.26 |
| | 9 | 9 | 0.1 | 0.03 | 0 | 4.74 | 0.12 |
| HFE(10) | 10 | 10 | 0.28 | 0.11 | 0.02 | 13.45 | 0.8 |
| | 10 | 10 | 0.28 | 0.12 | 0.02 | 13.54 | 0.81 |
| HFE(11) | 11 | 11 | 0.77 | 0.35 | 0.02 | 36 | 3.15 |
| | 11 | 11 | 1.1 | 0.33 | 0.06 | 59.15 | 5.3 |
| HFE(12) | 12 | 12 | 1.4 | 0.8 | 0.26 | 96 | 31 |
| | 12 | 12 | 1.4 | 0.8 | 0.26 | 98 | 28 |
| HFE(13)* | 13 | 13 | 4.9 | 2.5 | 0.6 | 449 | 349 |
| | 13 | 13 | 5 | 2 | 0.6 | 441 | 287 |
| HFE(14)* | 14 | 14 | 10 | 4.2 | 1.2 | 885 | 2313 |

Table 4.4: HFE time comparison using BBA, MBBA and MBBA2

In Table 4.4 we compare the timings for the solution of HFE(N). Each timing represents the total time taken by Lemma 4.1.4 during the process of solving HFE(N). The last two columns show the time taken by the computation of a `Lex` Gröbner basis and a `DegLex` border basis in ApCoCoA [12] respectively. Finally, we conclude that the MBBA2 is the most efficient version even if there are no mutants.

Chapter 5

# Techniques Using Mixed Integer Linear Programming

In this chapter we will restrict our attention to solving polynomial systems defined over $\mathbb{F}_2$. Although the generalization to other finite base fields is straightforward, we want to concentrate on the fundamental principles in the most important case. We study recent suggestions of transferring the problem of solving a system of polynomial equations over $\mathbb{F}_2$ into a mixed integer linear programming problem. In particular, we develop several strategies for converting the polynomial system over $\mathbb{F}_2$ to a polynomial system over $\mathbb{R}$ (respectively over $\mathbb{Z}$). Furthermore, we present a new conversion method based on propositional logic and pseudo-boolean optimization. Towards the end of this chapter we develop new hybrid conversion methods to accelerate the performance of an IP solver. This enables us to make use of several algorithms in the field of discrete optimization. Several algorithms have been developed (or optimized) and implemented. The experimental results are presented and compared which show that our newly developed techniques result in substantial speed up of IP solvers. Furthermore, note that some IP solvers like CPLEX can be parallelized. Thus we can benefit from parallelization capabilities of IP solvers to solve systems of polynomial equations. To conclude this chapter we present a comparison of all techniques in plots and tables.

## 5.1 Mixed Integer Linear Programming (MILP)

This section serves as a foundation for coming sections. We discuss our approach to use techniques from integer linear programming for finding a 0-1 valued solution

of a system of polynomial equations over $\mathbb{F}_2$. Furthermore, we review some necessary concepts from the theory of integer linear programming. In particular, we discuss some standard techniques used for modeling and solving mixed integer linear problems.

### 5.1.1 Mixed Integer Linear Programming Problems

*Integer optimization* deals with the problem of minimizing (or maximizing) a function of several variables subject to equality and inequality constraints and integrality restrictions on some or all of the variables. A *Mixed Integer Linear Programming (MILP) problem* is a problem of the form

$$\begin{aligned} \text{Minimize} \quad & z = cx + dy \\ \text{Subject to} \quad & \mathcal{A}x + \mathcal{B}y \leq b \\ & x \geq 0 \text{ integral} \\ & y \geq 0 \end{aligned} \tag{5.1}$$

where the data are the row vectors $c \in \mathbb{Q}^n$, $d \in \mathbb{Q}^p$, the matrices $\mathcal{A} \in \mathbb{Q}^{m \times n}$, $\mathcal{B} \in \mathbb{Q}^{m \times p}$ and the column vector $b \in \mathbb{Q}^m$; and the variables are the column vectors $x \in \mathbb{Z}_+^n$ and $y \in \mathbb{R}_+^p$. The set $S$ of all $(x, y) \in \mathbb{Z}_+^n \times \mathbb{R}_+^p$ which satisfies the linear constraints $\mathcal{A}x + \mathcal{B}y \leq b$, i.e.

$$S = \{(x, y) \in \mathbb{Z}_+^n \times \mathbb{R}_+^p \mid \mathcal{A}x + \mathcal{B}y \leq b\}$$

is called a *feasible set*. An element $(x, y) \in S$ is called a *feasible point*. A MILP problem is called feasible if $S \neq \emptyset$ and infeasible if $S = \emptyset$. The function $z = cx + dy$ is the *Objective Function (OF)* that we want to minimize (or maximize).

A MILP problem has either an optimal solution, is unbounded, or is infeasible. If there exists for any $w \in \mathbb{R}$ an $(x', y') \in S$ such that $cx' + dy' < w$, the MILP problem is unbounded. A solution for a MILP problem is *optimal* if a feasible point $(x'', y'') \in S$ exists with $cx'' + dy'' \leq cx + dy$ for all $(x, y) \in S$.

Special cases of MILP problems are the *Linear Programming (LP)* problems, where all variables are continuous, i.e. when $c = 0$, and the *pure Integer Programming (IP)* problems, where all variables are integer valued, i.e. when $d = 0$. In first case the set $S$ of feasible solutions to 5.1 is called a *mixed integer linear set* and in second case it is called a *pure integer linear set*. We are mainly interested in two other special cases of MILP problems. The first one is called 0-1 MILP problem where the integer variables are replaced by binary variables. The second one is the case where the majority of the variables (but not all) are replaced by binary variables.

### Cutting Plane Methods

Solving a MILP problem such as 5.1 is NP-hard (see Cook [54]). The polyhedral approach is a powerful tool for solving MILP problems 5.1. One approach that has been quite successful in practice is based on an idea that is commonly used in computational mathematics: Find a relaxation that is easier to compute and gives a tight approximation. We focus on linear programming relaxations. Given a mixed integer linear set

$$S = \{(x, y) \in \mathbb{Z}_+^n \times \mathbb{R}_+^p \mid \mathcal{A}x + \mathcal{B}y \leq b\},$$

a *linear programming relaxation* of $S$ is a set

$$\{(x, y) \in \mathbb{R}_+^n \times \mathbb{R}_+^p \mid \mathcal{A}'x + \mathcal{B}'y \leq b'\}$$

that contains $S$. Why LP relaxations? Mainly for two reasons: there are efficient practical algorithms for solving linear programs. Second, one can generate a sequence of linear programming relaxations that provides increasingly tight approximations of the set $S$. For a mixed integer set $S$, there is a natural linear programming relaxation:

$$\{(x, y) \in \mathbb{R}_+^n \times \mathbb{R}_+^p \mid \mathcal{A}x + \mathcal{B}y \leq b\}$$

which is obtained from the system that defines $S$ by discarding the integrality requirement on the vector $x$. Starting from a linear programming relaxation cutting plane methods aim at finding an optimal solution of the MILP problem 5.1. For details and an extensive coverage of the subject we refer to [107], Chapter 11. In this chapter we focus on conversion methods of the following type.

## 5.1.2 Conversion Methods

The main idea that we address is to transform the problem of solving a system of polynomial equations over $\mathbb{F}_2$ (boolean system) into a MILP problem. We convert the boolean equation system into an equation system over $\mathbb{R}$ (resp. $\mathbb{Z}$) and model the problem of finding a 0-1 valued solution for the boolean system as a MILP problem. In this sense the process of solving consists of the following four steps.

S1) **Transformation to $\mathbb{R}$ or $\mathbb{Z}$:** Transform a system of polynomial equations over $\mathbb{F}_2$ into a system of polynomial equations over $\mathbb{R}$ (resp. $\mathbb{Z}$).

S2) **Modeling a MILP Problem:** Model the transformed system as a MILP problem.

S3) **IP Solver:** Use an IP solver to solve the modeled MILP problem.

S4) **Inverse Transformation:** Use inverse transformation to obtain a solution of the original system from the solution of the MILP problem.

Note that, depending on the chosen conversion method, the first two steps can also be performed as a single step. For instance, this is the case for the method developed in Section 5.4. Figure 5.1 gives a visible explanation of these steps.

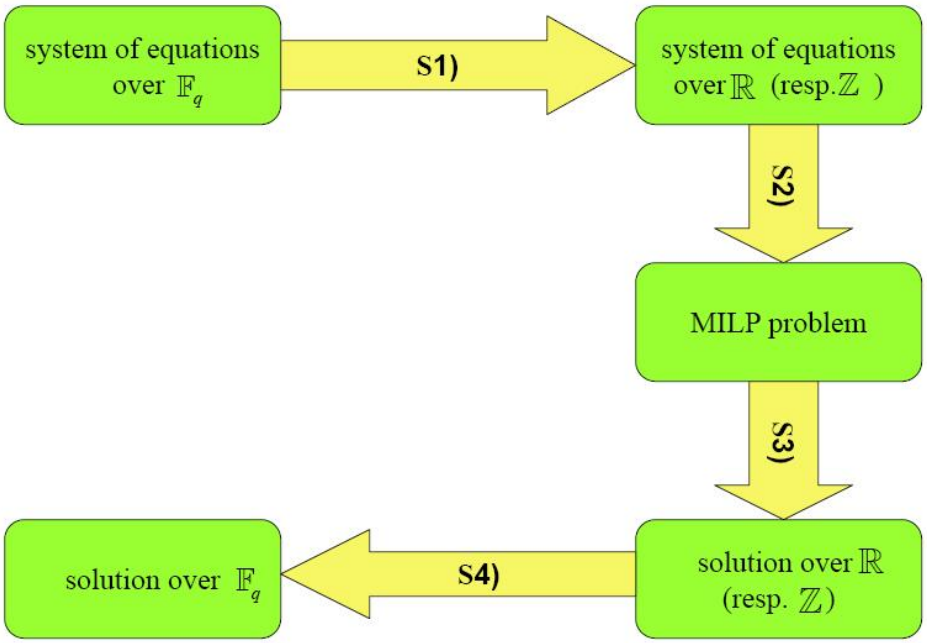



Figure 5.1: Solving Process

In the following we discuss these four steps one by one.

### 5.1.3 Transformation to $\mathbb{R}$ or $\mathbb{Z}$

Let $f_1 = \ldots = f_m = 0$, where $f_1, \ldots, f_m \in \mathbb{F}_2[x_1, \ldots, x_n]$, be a polynomial equation system. We consider the problem of solving the system $f_1 = \cdots = f_m = 0$ for $\mathbb{F}_2$-rational solutions. Let $X = \{X_1, \ldots, X_n\}$ be a set of real variables (respectively integer variables), i.e. variables over $\mathbb{R}$ (respectively over $\mathbb{Z}$). The task of solving

the polynomial equation system $f_1 = \cdots = f_m = 0$ can be rephrased as follows: Find a tuple $(a_1, \ldots, a_n) \in \{0,1\}^n$ such that

$$\begin{aligned} F_1(a_1, \ldots, a_n) &\equiv 0 \pmod 2 \\ &\vdots \\ F_m(a_1, \ldots, a_n) &\equiv 0 \pmod 2 \end{aligned} \tag{5.2}$$

where $F_i \in \mathbb{R}[X_1, \ldots, X_n]$ (respectively $F_i \in \mathbb{Z}[X_1, \ldots, X_n]$) is the standard (respectively a lifting) representative (see Definition 5.2.5) of $f_i$. Thus we are looking for an integer solution $(a_1, \ldots, a_n)$ of the system 5.2 which satisfies $0 \leq a_i \leq 1$. Note that at this stage the polynomials $F_1, \ldots, F_m$ are nonlinear.

There could be different methods to perform step S1). Most probably such methods exist in the literature but they have not been used for our purpose. For instance, the method used by J. Borghoff et al. [31] and M. Lamberger et al. [124], is frequently used in operations research (see [24]). Also the transformation method used in [117] is based on a basic understanding of operations in polynomial rings and their quotients. Last but not least, there are methods developed by us in Sections 5.4 and 5.5, which perform substantially better than the methods in [117, 31]. It is possible that further investigations in this direction could lead towards even more efficient transformation methods.

### 5.1.4 Modeling a MILP Problem

Usually there is more than one way to model a MILP problem or an IP problem. In integer programming, formulating a "good" model is of crucial importance for solving the MILP problem (see [173], Chapter 1). A first question in formulating a model is usually defining variables but in the case of a system of polynomial equations which results from step S1) the obvious choice is to take the variables involved (initial variables). We will also introduce some additional variables depending on the transformation model we use.

The second question is finding a good objective function. As we will see later we are interested in a feasible point and not in an optimal solution for our problem. Hence, we have a lot of freedom to choose the objective function. The main problem is to find a good formulation for

$$S = \{(x, y) \in \mathbb{Z}_+^n \times \mathbb{R}_+^p \mid \mathcal{A}x + \mathcal{B}y \leq b\}.$$

Here it is often easy to find $\mathcal{A}$, $\mathcal{B}$, and $b$ which yield a valid formulation for $S$ but this description of the feasible set might not be the best one for actually solving the problem. The reason is that integer optimization algorithms such as branch-and-cut need a lower bound on the objective function and they determine this bound often by using relaxations. One possible relaxation of the problem is to solve the corresponding LP problem, i.e. by assuming $c = 0$ (see Problem 5.1). The feasible set $S$ of the MILP problem is a subset of the feasible set of the LP problem. The smaller the feasible set of the LP problem is, the more precise are the bounds for the MILP problem and the efficiency of most algorithms depends on these bounds. To formulate $S$, we consider the available literature on MILP which is mentioned in the following.

One branch of integer programming deals with optimizing a nonlinear objective function subject to nonlinear constraints. In particular, the most interesting is the case of the 0-1 *quadratic programming problem*. The 0-1 quadratic programming problem seeks to optimize a quadratic objective function subject to several quadratic constraints, along with the condition that each variable is restricted to take on a value of either 0 or 1. Such problems arise in a host of contemporary application areas such as telecommunications [15, 97, 113], manufacturing and scheduling [5, 170], epileptic seizure warning [45, 99], subsume unconstrained 0-1 quadratic programs, 0-1 quadratic knapsack problems, and quadratic assignment problems.

Due to such huge applications of the 0-1 quadratic programming problems, several methods and algorithms have been proposed for solving them. Because of its NP-hardness, many of these are heuristic in nature (see [39, 115]). But we will consider exact optimization approaches only. Therefore, we are in luck: we can profit from some of the approaches to solve 0-1 quadratic programming problems which are given in the following.

### Convex Reformulations

The various approaches to solve 0-1 quadratic programming problems also include the 0-1 convex quadratic reformulations (see [14, 26, 44, 91, 136, 152]). Although 0-1 linear reformulations of 0-1 quadratic programming problems are the most common approaches, other methods have been proposed. Let us cite, for example, algebraic and dynamic programming methods ([60, 92]), reformulation to a continuous concave minimization problem ([108]) and enumerative methods based on different types of relaxations such as Lagrangian relaxation, semidefinite relaxation or convex quadratic relaxation ([11, 29, 43, 47, 77, 96]). To perform step S2) we can benefit from such

relaxation techniques. But it is still a challenge to find the best ones. In this chapter, we do not consider relaxation techniques. Instead we focus on reformulation-linearization techniques given in the following.

#### Reformulation-Linearization Techniques

Several linearization strategies have been proposed in the literature for reformulating 0-1 quadratic programs as equivalent MILP problems, starting with Fortet [78], and evolving into more concise formulations that either require additional binary variables as in Glover and Woolsey [86], or that introduce only additional continuous variables and constraints as in Glover [85] and Glover and Woolsey [87]. A significantly tighter reformulation was proposed by Adams and Sherali [2], which was demonstrated to dramatically improve the computational performance in comparison with the preceding strategies. This approach was subsequently generalized and enhanced to design the Reformulation-Linearization Technique (RLT) in Sherali and Adams [161].

Further improvement is due to the work of Chaovalitwongse *et al.* [46], who provide a novel transformation of the 0-1 quadratic program into a linear 0-1 mixed integer program, which requires the introduction of only a linear number of additional variables and constraints. Later it was demonstrated by Sherali and Smith in [162], that by re-stating the problem as an equivalent mixed integer bilinear program and using a series of variable transformations, a similarly structured but tighter equivalent linearized 0-1 mixed integer programming problem can be derived. They provide both theoretical as well as computational comparisons between this resulting problem and alternative traditional linearizations based on substituting a new (continuous) variable for each distinct quadratic term in the problem. The latest development in this direction is due to H. Xiaozheng et al. [174] who propose a computational enhancement for a linearization technique to make the linearized model much faster to solve. We consider all these developments to perform step S2). This enables us to benefit from the research on reformulation-linearization to perform step S2). Note that in most of the cases we need to make adjustments to the linearization techniques described above to make them applicable for our needs.

### 5.1.5 IP Solver and Inverse Transformation

After step S2) we need an IP solver to solve the MILP problem. We use the following solvers to solve MILP problems.

1) **CPLEX:** The commercial linear optimization tool CPLEX by ILOG [102]. CPLEX parameters reference manual provides several useful features for solving MILP problems. Parameters, accessible and manageable by users, control the behavior of CPLEX. Using these parameters we can choose different optimal solution search strategies. For instance, some very useful parameters for our purpose are given in the following table. For each MILP problem first we need to learn from

| Parameter | Functionality |
|---|---|
| MIP emphasis switch | Controls trade-offs between speed, feasibility, optimality, and moving bounds in MIP. |
| MIP variable selection strategy | Sets the rule for selecting the branching variable at the node which has been selected for branching. |
| Feasibility pump switch | Turns on or off the feasibility pump heuristic for mixed integer programming (MIP) models. |
| MIP integer solution limit | Sets the number of MIP solutions to be found before stopping. |
| Parallel mode switch | Sets the parallel optimization mode. Possible modes are automatic, deterministic, and opportunistic. |
| Global default thread count | Sets the default maximal number of parallel threads that will be invoked by any CPLEX parallel optimizer. |

Table 5.1: Some important CPLEX parameters

our experiments that under which parameter settings it can be solved efficiently, i.e. by using less time and memory. Different MILP problems behave differently under same parameter settings. There are MILP problems whose timings are affected by the change of parameters dramatically. So we need to be careful while choosing CPLEX parameters for our problem. As given by Table 5.1, CPLEX has a users choice for emphasis on feasibility or optimality. We choose emphasis on feasibility for our experimental results because we are not interested in optimality and stop after we found the first solution because we assume that there is only one solution (in most of the cases). We use CPLEX version 12.2. Moreover, CPLEX can be parallelized. We run CPLEX in parallel on a laptop with a 2.13 GHz Intel Pentium P6200 Dual Core processor and 4GB RAM.

2) **GLPK:** The GLPK (GNU Linear Programming Kit) package is intended for solving large-scale linear programming (LP), mixed integer programming (MIP),

and other related problems. It is a set of routines written in ANSI C and organized in the form of a callable library. GLPK does not provide the facility of parallel computing. We run GLPK on a laptop with a 2.13 GHz Intel Pentium P6200 Dual Core processor and 4GB RAM, but we are using only a single thread out of two.

After solving the MILP problem, the last step S4) is to obtain the 0-1 valued solution of the original system of polynomial equations from the solution of the MILP problem. This step returns the values of the initial variables, i.e. the variables in the original system by ignoring the values for the newly introduced variables.

Experimental results are presented on cryptographic examples coming from the CTC cipher and the small scale AES cipher. For details about the CTC cipher, we refer to Section 2.3. For an extensive coverage of the subject, we refer to [57]. Furthermore, we model S-boxes of the CTC cipher in the following three ways.

1) **Full-Sbox (F-Sbox):** Using all 14 equations.

2) **Half-Sbox (H-Sbox):** Using first 7 equations out of 14.

3) **Min-Sbox (M-Sbox):** Using minimum number of equations out of 14. To this end we can choose last three equations of the S-Box.

For details about the small scale AES cipher, we refer to Section 2.3. The conversion algorithms are implemented in C++. In all experimental results presented, the timings for the conversion algorithms were ignored, since they do not contribute to the complexity of solving and take very little time. For instance, all conversion algorithms, for the conversion of CTC(7,7), take less than 10 seconds. Finally, we note that all timings can be reproduced, if we do not permute the set of input linear equalities and inequalities. The reason for this is that IP solvers also implement randomized algorithms which rely heavily on heuristical methods. The implementations of conversion techniques developed in this chapter are available online in the package `glpk` (see Appendix C) of the computer algebra system ApCoCoA [12].

## 5.2 Techniques for Polynomial Conversion

In this section we review very recent suggestions (see [31] and [117]) of transferring the problem of solving a system of polynomial equations over $\mathbb{F}_2$ into a mixed integer

linear programming problem and generalize the approach. Furthermore, we investigate restrictions on variables introduced by a conversion method and the choice of a suitable objective function. The experimental results are presented and compared. Based on our experimental results, we conclude which conversion method provides better results.

In [24], an overview of possible representations of polynomials over $\mathbb{F}_2$ as polynomials over $\mathbb{R}$ is listed. Later this study was extended slightly in [124] but the main idea behind the representation methods was basically unaltered. Recently, some methods have been proposed for transferring the problem of solving a system of polynomial equations over $\mathbb{F}_2$ into a MILP problem. In [117], M. Kreuzer provided an algorithm based on converting polynomial equations over $\mathbb{F}_2$ into polynomial equations over $\mathbb{Z}$. We call this method *Integer Polynomial Conversion (IPC)*. In [31], J. Borghoff et al. provided another method based on converting polynomial equations over $\mathbb{F}_2$ into polynomial equations over $\mathbb{R}$. We reformulate the algorithm for IPC and provide an algorithm based on the ideas of J. Borghoff et al. to solve general systems of polynomial equations.

Let $\mathbb{F}_2$ be the finite field with two elements and let $f_1, \ldots, f_m \in P = \mathbb{F}_2[x_1, \ldots, x_n]$ be non-zero polynomials. We are interested in finding $\mathbb{F}_2$-rational solutions of the following system of polynomial equations.

$$\begin{aligned} f_1(x_1, \ldots, x_n) &= 0 \\ &\vdots \\ f_m(x_1, \ldots, x_n) &= 0 \end{aligned}$$

We convert the polynomial equations defined over $\mathbb{F}_2$ into polynomial equations which hold over the reals (respectively over the integers). We denote by $\mathbb{T}^n$ the monoid of terms for $\mathbb{F}_2[x_1, \ldots, x_n]$. An element of the monoid of terms $\mathbb{T}^n$ will be denoted by $t$.

**Definition 5.2.1.** Let $n \geq 1$.

a) A polynomial $f \in \mathbb{F}_2[x_1, \ldots, x_n]$ of the form $f = x_{i_1} \cdots x_{i_s}$, where $1 \leq i_1 < i_2 < \cdots < i_s \leq n$ is called a **squarefree term**.

b) For a squarefree term $t_j = x_{i_1} \cdots x_{i_s} \in \mathbb{T}^n$, where $1 \leq i_1 < i_2 < \cdots < i_s \leq n$, we let $N_j = \{i_1, \ldots, i_s\}$.

Note that in Definition 5.2.1 the set $N_j$ is a kind of indexing set. For a squarefree term $t_j$, the number of elements in the set $N_j$ is the degree of $t_j$. From now on $f_i$ will be a squarefree polynomial (boolean polynomial), i.e. all terms in the support of $f_i$ will

be squarefree. Let $X = \{X_1, \ldots, X_n\}$ be a set of real variables (respectively integer variables), i.e. variables over $\mathbb{R}$ (respectively over $\mathbb{Z}$). A conversion method should at least guarantee that a solution for $f_i$ results in a solution for the associated polynomial (or polynomials) over $\mathbb{R}$ (respectively over $\mathbb{Z}$).

We are looking for an integer solution $(a_1, \ldots, a_n)$ of the system 5.2 which satisfies $0 \leq a_i \leq 1$. This formulation suggests to linearize the system and to apply a mixed integer linear programming algorithm for finding a solution satisfying the stated bounds. In the following we turn these ideas into effective algorithms.

### 5.2.1 Integer Polynomial Conversion (IPC)

The first conversion method we discuss was proposed by M. Kreuzer in [117]. It is based on converting polynomial equations over $\mathbb{F}_2$ into polynomial equalities and inequalities over $\mathbb{Z}$. The idea behind it is to convert each equation as a whole to preserve the structure of equation. Since we want to use the resulting system of equalities and inequalities in an integer programming problem, this means that we can restrict some (or all) variables to be integers. The task of solving the polynomial equation system $f_1 = \cdots = f_m = 0$ can be rephrased as follows: Find a tuple $(a_1, \ldots, a_n) \in \{0,1\}^n$ such that

$$\begin{aligned} F_1(a_1, \ldots, a_n) &\equiv 0 \pmod 2 \\ &\vdots \\ F_m(a_1, \ldots, a_n) &\equiv 0 \pmod 2 \end{aligned} \tag{5.3}$$

where $F_i \in \mathbb{Z}[X_1, \ldots, X_n]$ are lifting's of the polynomials $f_i$. Thus we are looking for an integer solution $(a_1, \ldots, a_n)$ of the system 5.3 which satisfies $0 \leq a_i \leq 1$. So the idea is to formulate these congruences 5.3 as a system of linear equalities and inequalities over $\mathbb{Z}$ and solve it using an IP-solver.

**Proposition 5.2.2. (Integer Polynomial Conversion (IPC))**
*Let $f_1, \ldots, f_m \in P = \mathbb{F}_2[x_1, \ldots, x_n]$. Then the following instructions define an algorithm which computes a tuple $(a_1, \ldots, a_n) \in \{0,1\}^n$ whose residue class in $\mathbb{F}_2^n$ represent a zero of the 0-dimensional radical ideal $I = \langle f_1, \ldots, f_m, x_1^2 + x_1, \ldots, x_n^2 + x_n \rangle$.*

*1) Reduce $f_1, \ldots, f_m$ modulo the field equations, i.e. make their supports squarefree. For $i = 1, \ldots, m$ let $S_i$ be the set of terms of degree $\geq 2$ in $f_i$, $s_i = \#\mathrm{Supp}(f_i)$ and $S = \bigcup_{i=1}^m S_i$.*

*2) For $i = 1, \ldots, m$, introduce a new integer indeterminate $K_i$ and write down the*

*linear inequality* $I_i: \ K_i \leq \lfloor s_i/2 \rfloor$.

3) *For every* $t_j \in S$, *introduce a new integer indeterminate* $X_{n+j}$. *For* $i = 1, \ldots, m$, *write* $f_i = \sum_j t_j + \ell_i$ *where the sum extends over all* $j$ *such that* $t_j \in S_i$ *and where* $\ell_i \in P_{\leq 1}$. *Form the equation* $F_i: \ \sum_j X_{n+j} + L_i - 2K_i = 0$, *where* $L_i \in \mathbb{Z}[X_1, \ldots, X_n]_{\leq 1}$ *is a lifting of* $\ell_i$ *over* $\mathbb{Z}$.

4) *For* $t_j \in S$, *write* $t_j = \prod_{\alpha \in N_j} x_\alpha$. *Form the linear inequalities*

$$I_{n+j}: \ \sum_{\alpha \in N_j} X_\alpha - X_{n+j} \leq |N_j| - 1,$$

*and*

$$I_{j\alpha}: \ X_\alpha \geq X_{n+j} \text{ for all } \alpha \in N_j.$$

5) *For all* $\alpha \in \{1, \ldots, n\}$, *let* $I'_\alpha: \ X_\alpha \leq 1$.

6) *Choose a linear polynomial* $C \in \mathbb{Z}[X_\alpha, X_{n+j}, K_i]$ *and use an IP solver to find the tuple of natural numbers* $(a_\alpha, a_{n+j}, c_i)$ *which solves the system of equations and inequalities* $\{I_i, F_i, I_{n+j}, I_{j\alpha}, I'_\alpha\}$ *and minimizes (or maximizes)* $C$.

7) *Return* $(a_1, \ldots, a_n)$ *and stop.*

*Proof.* For $\alpha = 1, \ldots, n$, we are looking for natural numbers $a_\alpha$ for which $I'_\alpha$ holds, so we have $a_\alpha \in \{0, 1\}$. Similarly, we have $a_{n+j} \in \{0, 1\}$ by $I'_\alpha$ and $I_{j\alpha}$. Moreover, if $t_j = \prod_{\alpha \in N_j} x_\alpha \in S$ and if one of the numbers $a_\alpha$ for $\alpha \in N_j$ is zero then $I_{j\alpha}$ implies $a_{n+j} = 0$. On the other hand, if $a_\alpha = 1$ for all $\alpha \in N_j$ then $I_{n+j}$ implies $a_{n+j} \geq 1$. Altogether, this means that $a_{n+j}$ equals $\prod_{\alpha \in N_j} a_\alpha$, the value of $t_j$ at $(a_1, \ldots, a_n)$.

Next it follows from $F_i$ that $F_i(a_1, \ldots, a_n) = 2K_i$ is an even number, and $I_i$ is nothing but the trivial bound for $K_i$ implied by the size of the support of $f_i$. In this way the solutions of the IP problem correspond uniquely to the tuples $(a_1, \ldots, a_n) \in \{0, 1\}^n$ which satisfy the above reformulation of the given polynomial system. □

**Remark 5.2.3.** Assume that we are in the setting of the algorithm in Proposition 5.2.2. If we can find a feasible binary/integer-valued solution for the MILP for an arbitrary objective function, this solution can be converted into a solution for the original system. Hence it is not important to find a minimal (or maximum) solution but a feasible point. But we have three natural questions. Which linear function might be a good objective function? Which variables should be restricted to be binary

or integers? Which optimization direction (maximize or minimize) could be a better choice?

We do not have a final answer for the first question at this stage but we remark that a good choice of an objective function can affect the running time of an IP solver strongly. We try to study it with the help of computation experiences in Section 5.2.2. A partial answer to the second question could be the following. The difficulty of solving a mixed integer program depends more on the number of integer variables than on the number of continuous variables (see [87]). Therefore our intuition tells us to keep as many variables continuous as we can. As proposed by F. Glover and E. Woolsey in [87], the linear inequalities in step 4) of the algorithm keep the variables $X_{n+j}$ continuous. It is however necessary to keep upper bounds of 1 on these variables, as noted by A.J. Goldman [88].

In view of these remarks we fix variables as follows. The initial state variables $X_1, \ldots, X_n$ will be forced to take on binary values. The variables $K_1, \ldots, K_m$ will be forced to take on integer values in the interval $[0, \lfloor s_i/2 \rfloor]$. The variables $X_{n+j}$ will be kept continuous in the interval $[0, 1]$. The variables $X_{n+j}$ depend on the initial state variables. This means that we do not have to restrict them to be integer or binary. In Section 5.2.2 we confirm our intuition by experiments.

Again we do not have an answer for the third question at this stage but we remark that it can affect the running time of an IP solver in certain cases. We try to study it with the help of computation experiences in Section 5.2.2.

To understand Proposition 5.2.2 better, we now apply it in a concrete case.

**Example 5.2.4.** Over the field $K = \mathbb{F}_2$, consider $f_1, f_2, f_3 \in K[x_1, x_2, x_3]$, where $f_1 = x_1x_2 + x_1x_3 + 1$, $f_2 = x_1x_3 + x_2x_3 + x_1 + x_3 + 1$, and $f_3 = x_1x_2 + x_1x_3 + x_2 + 1$. Let us follow the steps of the algorithm in Proposition 5.2.2.

1) Let $S_1 = \{x_1x_2, x_1x_3\}$, $S_2 = \{x_1x_3, x_2x_3\}$ and $S_3 = \{x_1x_2, x_1x_3\}$. Let $s_1 = 3$, $s_2 = 5$, $s_3 = 4$ and $S = \{x_1x_2, x_1x_3, x_2x_3\}$.

2) Introduce new integer indeterminates $K_1, K_2, K_3$ and write down the linear inequalities $I_1 : K_1 \leq 1$, $I_2 : K_2 \leq 2$ and $I_3 : K_3 \leq 2$.

3) Introduce new integer indeterminates $X_4, X_5, X_6$ and form the following equa-

tions.

$$\begin{aligned} F_1 &: X_4 + X_5 + 1 - 2K_1 = 0 \\ F_2 &: X_5 + X_6 + X_1 + X_3 + 1 - 2K_2 = 0 \\ F_3 &: X_4 + X_5 + X_2 + 1 - 2K_3 = 0 \end{aligned}$$

4) Form the linear inequalities

$$\begin{aligned} I_4 &: X_1 + X_2 - X_4 \leq 1,\ I_{11} : X_1 \geq X_4,\ I_{12} : X_2 \geq X_4, \\ I_5 &: X_1 + X_3 - X_5 \leq 1,\ I_{21} : X_1 \geq X_5,\ I_{23} : X_3 \geq X_5, \\ I_6 &: X_2 + X_3 - X_6 \leq 1,\ I_{32} : X_2 \geq X_6,\ I_{33} : X_3 \geq X_6. \end{aligned}$$

5) Let $I_1' : X_1 \leq 1$, $I_2' : X_2 \leq 1$ and $I_3' : X_3 \leq 1$.

6) Let $C = X_1 + X_2 + X_3$. Now use an IP solver to minimize $C$ subject to $\{I_1, \ldots, I_6, F_1, F_2, F_3, I_{11}, I_{12}, I_{21}, I_{23}, I_{32}, I_{33}, I_1', I_2', I_3'\}$.

7) Choose values for $X_1$, $X_2$ and $X_3$ from the solution provided by an IP solver. This will return $(1, 0, 1)$.

### 5.2.2 Experimental Results

Now we present our observations and results from experiments with the algorithm in Proposition 5.2.2. We try to find a good objective function and variables which should be restricted to be binary. Furthermore, we try to see which optimization direction (minimize or maximize) is a better choice. From now on we assume that we are in the setting of the conversion algorithm in Proposition 5.2.2.

#### Optimization Direction

Assume that we choose the objective function as the sum over all the initial variables $X_1, \ldots, X_n$ and assume that we impose restrictions on variables as given in Remark 5.2.3. Then Tables 5.2 and 5.3 show the experimental results for the two optimization directions. We run a number of experiments which show that more or less the same observations can be obtained for different choices of objective functions. After observing the timings in Tables 5.2 and 5.3, we are still not able to give a concrete answer to this question but we are on the safe side if we choose *maximization* as optimization direction. Furthermore, if we model S-boxes using first 7 equations out of 14, we can obtain better timings.

| | F-SBox | | H-SBox | | M-SBox | |
|---|---|---|---|---|---|---|
| System | Max | Min | Max | Min | Max | Min |
| CTC(3,3) | 16.8 | 14.7 | 9.2 | 6.6 | 10.8 | 14 |
| CTC(3,4) | 121 | 236 | 29 | 129 | 20 | >1000 |
| CTC(4,3) | 73 | 174 | 48 | 72 | 86 | 540 |
| CTC(4,4) | 4539 | 4831 | 1225 | 2177 | >14000 | >16000 |

Table 5.2: GLPK time comparison for optimization direction using IPC

| | F-SBox | | H-SBox | | M-SBox | |
|---|---|---|---|---|---|---|
| System | Max | Min | Max | Min | Max | Min |
| CTC(3,3) | 3 | 3.9 | 4 | 3.6 | 4.5 | 2.4 |
| CTC(3,4) | 6.5 | 44 | 2.6 | 35 | 3.9 | 31 |
| CTC(4,3) | 3.4 | 45 | 20 | 21 | 4 | 31 |
| CTC(4,4) | 90 | 58 | 107 | 152 | 85 | 73 |
| CTC(4,5) | 847 | 205 | 218 | 1018 | 484 | 272 |
| CTC(5,4) | 408 | 1224 | 742 | 650 | 1264 | 1295 |

Table 5.3: CPLEX time comparison for optimization direction using IPC

### Restrictions on Variables

Assume that we choose the objective function as the sum over all the initial variables $X_1, \ldots, X_n$ and *maximization* as optimization direction. Since we are looking for a 0-1 solution, the initial variables $X_1, \ldots, X_n$ must be forced to take on binary values. The integer variables $K_1, \ldots, K_m$ must take on integer values in the interval $[0, \lfloor s_i/2 \rfloor]$. The remaining variables $X_{n+j}$ can be restricted in the following ways.

| | F-SBox | | | H-SBox | | | M-SBox | | |
|---|---|---|---|---|---|---|---|---|---|
| System | R1 | R2 | R3 | R1 | R2 | R3 | R1 | R2 | R3 |
| CTC(3,3) | 18 | 16.8 | 16.4 | 10.1 | 9.2 | 9 | 10.7 | 10.8 | 10.2 |
| CTC(3,4) | 214 | 121 | 117 | 31 | 29 | 28 | 23 | 20 | 20 |
| CTC(4,3) | 89 | 73 | 71 | 61 | 48 | 47 | 99 | 86 | 84 |
| CTC(4,4) | 6750 | 4539 | 4351 | 2623 | 1225 | 5520 | >10000 | >14000 | >5000 |

Table 5.4: GLPK time comparison for restrictions on variables using IPC

**R1:** Force the variables $X_{n+j}$ to take on binary values.

**R2:** Keep the variables $X_{n+j}$ continuous in the interval $[0, 1]$.

**R3:** Keep the variables $X_{n+j}$ continuous without any bounds.

| | F-SBox | | | H-SBox | | | M-SBox | | |
|---|---|---|---|---|---|---|---|---|---|
| System | R1 | R2 | R3 | R1 | R2 | R3 | R1 | R2 | R3 |
| CTC(3,3) | 3.68 | 3.04 | 3.03 | 1.6 | 4 | 4 | 4.5 | 4.48 | 4.6 |
| CTC(3,4) | 2.7 | 6.5 | 6.3 | 1.6 | 2.65 | 2.6 | 3.8 | 3.9 | 3.9 |
| CTC(4,3) | 3.8 | 3.4 | 3.6 | 8.3 | 20 | 20 | 3.9 | 4 | 3.9 |
| CTC(4,4) | 127 | 90 | 89 | 53 | 107 | 106 | 85 | 85 | 85 |
| CTC(4,5) | 172 | 847 | 841 | 85 | 218 | 218 | 483 | 484 | 483 |
| CTC(5,4) | 274 | 408 | 406 | 690 | 742 | 742 | 1250 | 1264 | 1250 |

Table 5.5: CPLEX time comparison for restrictions on variables using IPC

Tables 5.4 and 5.5 show that **R1** is a better choice for CPLEX and **R2** is a better choice for GLPK.

### Objective Function

The objective function is not important to get the correct solution of the problem, it is important for the performance of many mixed integer programming algorithms. In branch and bound algorithms the boundary function estimates the best value of the objective function obtainable by growing the search tree one node further. This value is an important factor in the process of choosing the next node in the search tree. The closer the value of the bounding function to the objective function the better. Assume that we choose *maximization* as optimization direction and the variables $X_{n+j}$ are restricted according to **R1**. The only restriction for the objective function we have is that it must be linear. Natural choices are as follows.

| | F-SBox | | | H-SBox | | | M-SBox | | |
|---|---|---|---|---|---|---|---|---|---|
| System | O1 | O2 | O3 | O1 | O2 | O3 | O1 | O2 | O3 |
| CTC(3,3) | 18 | 20 | 22 | 10 | 5 | 5.7 | 10 | 13 | 20 |
| CTC(3,4) | 214 | 305 | 281 | 21 | 38 | 32 | 23 | 132 | 93 |
| CTC(4,3) | 89 | 221 | 170 | 61 | 30 | 26 | 99 | 80 | 127 |
| CTC(4,4) | 6750 | 1853 | 3168 | 2536 | 480 | 480 | >10000 | >10000 | >10000 |

Table 5.6: GLPK time comparison for objective function using IPC

**O1:** The sum over all the initial variables $X_1, \ldots, X_n$.

**O2:** The sum over all variables.

**O3:** The sum over all new variables $X_{n+j}$ and $K_1, \ldots, K_n$ introduced by the conversion algorithm.

| | F-SBox | | | H-SBox | | | M-SBox | | |
|---|---|---|---|---|---|---|---|---|---|
| System | O1 | O2 | O3 | O1 | O2 | O3 | O1 | O2 | O3 |
| CTC(3,3) | 3.68 | 5.3 | 4.9 | 1.6 | 2.2 | 1.8 | 4.5 | 0.3 | 3.4 |
| CTC(3,4) | 2.7 | 23 | 11 | 1.6 | 10 | 20 | 3.8 | 4.8 | 0.9 |
| CTC(4,3) | 3.8 | 20 | 20 | 8.3 | 0.8 | 15 | 3.9 | 8 | 14 |
| CTC(4,4) | 127 | 105 | 433 | 53 | 78 | 120 | 85 | 82 | 19 |
| CTC(4,5) | 172 | 301 | 8.5 | 85 | 971 | 307 | 483 | 94 | 129 |
| CTC(5,4) | 274 | 1472 | 1915 | 690 | 1320 | 818 | 1250 | 501 | 1134 |

Table 5.7: CPLEX time comparison for objective function using IPC

Tables 5.6 and 5.7 show that **O1** is a better choice if we use full S-Box or half S-Box. Furthermore, **O3** can provide surprising results as in case of CTC(3,4) and CTC(4,5).

### 5.2.3 Real Polynomial Conversion (RPC)

In [31], J. Borghoff et al. provided a method based on converting polynomial equations over $\mathbb{F}_2$ into polynomial equations over $\mathbb{R}$. We call this method *Real Polynomial Conversion (RPC)*. They studied their method for systems of polynomial equations due to the *Bivium Cipher* but an algorithm for general systems of polynomial equations is still missing. We provide an algorithm for RPC to solve general systems of polynomial equations. The first ingredient that we need is the following definition.

**Definition 5.2.5.** The **standard conversion** is given by the map $\phi : \mathbb{F}_2 = \{\overline{0}, \overline{1}\} \to \{0, 1\} \subset \mathbb{R}$ defined by $\phi(\overline{0}) = 0$ and $\phi(\overline{1}) = 1$. The map $\phi$ can be extended to a map $\Phi : \mathbb{F}_2[x_1, \ldots, x_n] \longrightarrow \mathbb{R}[X_1, \ldots, X_n]$ defined by

$$\begin{aligned} c &\mapsto \phi(c) \\ x_i &\mapsto X_i \end{aligned}$$

where $c \in \mathbb{F}_2$. Then the **standard representation** of a polynomial $f \in \mathbb{F}_2[x_1, \ldots, x_n]$ is $\Phi(f)$.

So the task of solving the polynomial equation system $f_1 = \cdots = f_m = 0$ can be rephrased as follows: Find a tuple $(a_1, \ldots, a_n) \in \{0, 1\}^n$ such that

$$\begin{aligned} F_1(a_1, \ldots, a_n) &\equiv 0 \pmod 2 \\ &\vdots \\ F_m(a_1, \ldots, a_n) &\equiv 0 \pmod 2 \end{aligned} \tag{5.4}$$

where $F_i \in \mathbb{R}[X_1, \ldots, X_n]$ are standard representations of the polynomials $f_i$. Thus we are looking for an integer solution $(a_1, \ldots, a_n)$ of the system 5.4 which satisfies $0 \leq a_i \leq 1$. So the idea is to formulate this system as a system of linear equalities and inequalities over $\mathbb{R}$ and solve it using an IP-solver.

**Example 5.2.6.** Consider the polynomial $f = x_1x_2+x_3x_4+x_5+x_6+1 \in \mathbb{F}_2[x_1, \ldots, x_6]$. In the following we explicitly explain how to lift this polynomial over $\mathbb{R}$ using standard representation in such a way that the residue class of a zero of $\Phi(f)$ in $\mathbb{F}_2$ represent a zero of $f$. We use the following conversion rules for addition and multiplication.

$$\begin{aligned} \Phi(x_ix_j) &= X_iX_j \\ \Phi(x_i + x_j) &= X_i + X_j - 2X_iX_j \end{aligned}$$

Considering each term as a node we apply the map $\Phi$ once for each pair of nodes. This results in the following conversion steps.

1) $f = (x_1x_2 + x_3x_4) + (x_5 + x_6) + 1$.

2) Taking standard representation we have

$$(X_1X_2 + X_3X_4 - 2X_1X_2X_3X_4) + (X_5 + X_6 - 2X_5X_6) + 1.$$

3) Let $f' = X_1X_2 + X_3X_4 - 2X_1X_2X_3X_4$, and $f'' = X_5 + X_6 - 2X_5X_6$. Now the polynomial in step 2) becomes $(f') + (f'') + 1$.

4) Taking standard representation we have $f' + f'' - 2f'f'' + 1$.

5) Let $f''' = f' + f'' - 2f'f''$. Now the polynomial in step 4) becomes $f''' + 1$

6) Finally, taking standard representation we have $f''' + 1 - 2f''' = 1 - f'''$.

By substituting the values of $f'$, $f''$ and $f'''$ we have the polynomial

$$\begin{aligned} F = {} & 8X_1X_2X_3X_4X_5X_6 - 4X_1X_2X_3X_4X_5 - 4X_1X_2X_3X_4X_6 + 2X_1X_2X_3X_4 \\ & -4X_1X_2X_5X_6 - 4X_3X_4X_5X_6 + 2X_1X_2X_5 + 2X_3X_4X_5 + 2X_1X_2X_6 + 2X_3X_4X_6 \\ & -X_1X_2 - X_3X_4 + 2X_5X_6 - X_5 - X_6 + 1 \in \mathbb{R}[X_1, \ldots, X_6] \end{aligned} \tag{5.5}$$

The polynomial $F$ has 16 terms in its support and degree 6.

The effect of standard representation is that every tuple $(a_1, \ldots, a_n) \in \{0,1\}^n$ at which $F$ is satisfied corresponds uniquely to a zero of $f$ in $\mathbb{F}_2^n$, that is, the residue class of $(a_1, \ldots, a_n)$ in $\mathbb{F}_2^n$ represent a zero of $f$. To see this it suffices to observe the standard conversion rule for addition which is given by the following table.

| $x_1$ | $x_2$ | $x_1 + x_2$ | $X_1 + X_2 - 2 \cdot X_1 \cdot X_2$ |
|---|---|---|---|
| $\overline{0}$ | $\overline{0}$ | $\overline{0}$ | 0 |
| $\overline{0}$ | $\overline{1}$ | $\overline{1}$ | 1 |
| $\overline{1}$ | $\overline{0}$ | $\overline{1}$ | 1 |
| $\overline{1}$ | $\overline{1}$ | $\overline{0}$ | 0 |

The standard representation results in increasing degree and increasing number of terms over the real domain.

**Remark 5.2.7. (Splitting)**
To keep the degrees of converted polynomials low, we introduce some new auxiliary variables. This will split a long polynomial into smaller polynomials, then we take its standard representations. The maximum number of terms in a polynomial over $\mathbb{F}_2$ could be four to keep the real polynomial quadratic. For instance, the equation $x_1x_2 + x_3x_4 + x_5 + x_6 + 1 = 0$ can be split up into two equations $y_1 + x_1x_2 = x_3x_4 + x_5$ and $y_1 = x_6 + 1$ having at most four terms. The variable $y_1$ is the splitting variable. To keep the degree of real polynomial two we introduce two more variables $y_2$ and $y_3$ as follows:

$$\begin{aligned} y_1 + y_2 &= y_3 + x_5 \\ y_1 &= x_6 + 1 \\ y_2 &= x_1x_2 \\ y_3 &= x_3x_4 \end{aligned}$$

Now the standard representation results in the following four quadratic equations which hold over reals.

$$\begin{aligned} Y_1 + Y_2 - 2Y_1Y_2 &= Y_3 + X_5 - 2Y_3X_5 \\ Y_1 &= 1 - X_6 \\ Y_2 - X_1X_2 &= 0 \\ Y_3 - X_3X_4 &= 0 \end{aligned}$$

While converting a boolean equation, we ensure that the new equations are defined over $\mathbb{R}$. The only requirement we have is that the solution of the system over $\mathbb{F}_2$ is also a solution of the real system. The additional non-binary solutions of the real system can be ignored.

In the following we abuse the notation $\mathbb{T}^n$. Since the monoid of terms $\mathbb{T}^n$ does not depend on the ring of coefficients, we consider $\mathbb{T}^n$ as monoid of terms of $\mathbb{F}_2[x_1, \ldots, x_n]$ and $\mathbb{R}[X_1, \ldots, X_n]$. The only distinction we make is the following. An element of the monoid of terms for $\mathbb{F}_2[x_1, \ldots, x_n]$ will be denoted by $t$ and an element of the monoid of terms for $\mathbb{R}[X_1, \ldots, X_n]$ will be denoted by $T$. The following proposition turns above ideas into an effective algorithm.

**Proposition 5.2.8. (Real Polynomial Conversion (RPC) )**
*Let $f_1, \ldots, f_m \in P = \mathbb{F}_2[x_1, \ldots, x_n]$. Then the following instructions define an algorithm which computes a tuple $(a_1, \ldots, a_n) \in \{0,1\}^n$ whose residue class in $\mathbb{F}_2^n$ represent a zero of the 0-dimensional radical ideal $I = \langle f_1, \ldots, f_m, x_1^2 + x_1, \ldots, x_n^2 + x_n \rangle$.*

1) *Reduce $f_1, \ldots, f_m$ modulo the field equations, i.e. make their support squarefree. For $i = 1, \ldots, m$, let $S_i$ be the set of terms of degree $\geq 2$ in $f_i$. Let $S = \bigcup_{i=1}^m S_i$ and $s = |S|$.*

2) *For every $t_j \in S$, introduce a new indeterminate $x_{n+j}$ and form the equation $f'_{m+j} : x_{n+j} = t_j$. For $i = 1, \ldots, m$, write $f_i = \sum_j t_j + \ell_i$ where the sum extends over all $j$ such that $t_j \in S_i$ and where $\ell_i \in P_{\leq 1}$. Form the equation $f'_i : \sum_j x_{n+j} + \ell_i = 0$.*

3) *For $i = 1, \ldots, m+s$, let $F_i$ be the equation which is the standard representation of $f'_i$. Let $S'_i$ be the set of terms of degree $\geq 2$ in $F_i$ and let $S' = \bigcup_{i=1}^{m+s} S'_i$.*

4) *For every $T_k \in S'$, introduce a new real indeterminate $X_{n+s+k}$. For $i = 1, \ldots, m+s$, replace $T_k \in S'_i$ by $X_{n+s+k}$ in the support of $F_i$. This makes $F_i$ linear.*

5) *For $T_k \in S'$, write $T_k = \prod_{\alpha \in N_k} X_\alpha$. Form the linear inequalities*

$$I_{n+s+k} : \sum_{\alpha \in N_k} X_\alpha - X_{n+s+k} \leq |N_k| - 1,$$

*and*

$$I_{k\alpha} : X_\alpha \geq X_{n+s+k} \text{ for all } \alpha \in N_k.$$

.

*6) For all $\alpha \in \{1, \ldots, n\}$, let $I_\alpha : \ X_\alpha \leq 1$.*

*7) Choose a linear polynomial $C \in \mathbb{Q}[X_\alpha, X_{n+j}, X_{n+s+k}]$ and use an IP solver to find the tuple of natural numbers $(a_\alpha, a_{n+j}, a_{n+s+k})$ which solves the system of equations and inequalities $\{F_i, I_{n+s+k}, I_{k\alpha}, I_\alpha\}$ and minimizes (or maximizes) $C$.*

*8) Return $(a_1, \ldots, a_n)$ and stop.*

*Proof.* For $\alpha = 1, \ldots, n$, we are looking for natural numbers $a_\alpha$ for which $I_\alpha$ holds, therefore we have $a_\alpha \in \{0, 1\}$. Similarly, we have $a_{n+j} \in \{0, 1\}$ by $I_\alpha$ and $F_{m+j}$ where $j = 1, \ldots, s$. Also we have $a_{n+s+k} \in \{0, 1\}$ by $I_\alpha$, $F_{m+j}$ and $I_{k\alpha}$. Moreover, if $T_k = \prod_{\alpha \in N_k} X_\alpha \in S'$ and if one of the numbers $a_\alpha$ for $\alpha \in N_k$ is zero then $I_{k\alpha}$ implies $a_{n+s+k} = 0$. On the other hand, if $a_\alpha = 1$ for all $\alpha \in N_k$ then $I_{n+s+k}$ implies $a_{n+s+k} \geq 1$. Altogether, this means that $a_{n+s+k}$ equals $\prod_{\alpha \in N_k} a_\alpha$, the value of $T_k$ at $(a_1, \ldots, a_n, a_{n+1}, \ldots, a_{n+s})$.

Next it follows from standard representation 5.2.5 that $F_i \in \{0, 1\}$. In this way the solutions of the IP problem correspond uniquely to the tuples $(a_1, \ldots, a_n) \in \{0, 1\}^n$ which satisfy the above reformulation of the given polynomial system. □

Assume that we are in the setting of the algorithm in Proposition 5.2.8. Note that if $\max\{\deg(f_i) \mid i \in \{1, \ldots, m\}\} \leq 2$ and for $i = 1, \ldots, m$, the maximum number of terms in the support of $f_i$ does not exceed 4, the algorithm works with quadratic polynomials in all of its iterations.

**Remark 5.2.9.** Assume that we are in the setting of the algorithm in Proposition 5.2.8. As in Remark 5.2.3, if we can find a feasible binary/integer-valued solution for the MILP for an arbitrary objective function, this solution can be converted into a solution for the original system. Hence it is not important to find a minimal (or maximum) solution but a feasible point. But we have three natural questions again. Which linear function might be a good objective function? Which variables should be restricted to be binary or integers? Which optimization direction (maximize or minimize) should we choose?

An objective function can affect the running time of an IP solver strongly. We try to study it with the help of computation experiences in Section 5.2.4. A partial answer to the second question could be the following. The difficulty of solving a mixed integer program depends more on the number of integer variables than on the number

of continuous variables (see [87]). Therefore our intuition tells us to keep as many variables continues as we can. As proposed by F. Glover and E. Woolsey in [87], the linear inequalities in step 4) of the algorithm keep the variables $X_{n+s+k}$ continuous. It is however necessary to keep upper bounds of 1 on these variables, as noted by A.J. Goldman [88].

In view of these remarks we fix variables as follows. The initial state variables $X_1, \ldots, X_n$ will be forced to take on binary values. All other newly introduced variables will be kept continuous in the interval $[0, 1]$. These variables depend on the initial state variables. This means that we do not have to restrict them to be integer or binary. In Section 5.2.4 we confirm our intuition by experiments.

Again we do not have an answer for the third question at this stage but we remark that it can affect the running time of an IP solver in certain cases. We try to study it with the help of computation experiences in Section 5.2.4.

To understand Proposition 5.2.8 better, we now apply it in a concrete case.

**Example 5.2.10.** Over the field $K = \mathbb{F}_2$, consider $f_1, f_2, f_3 \in K[x_1, x_2, x_3]$, where $f_1 = x_1x_2 + x_1x_3 + 1$, $f_2 = x_1x_3 + x_2x_3 + x_1$, and $f_3 = x_1x_2 + x_1x_3 + x_2 + 1$. Let us follow the steps of the algorithm in Proposition 5.2.8.

1) Let $S_1 = \{x_1x_2, x_1x_3\}$, $S_2 = \{x_1x_3, x_2x_3\}$, and $S_3 = \{x_1x_2, x_1x_3\}$. Let $S = \{x_1x_2, x_1x_3, x_2x_3\}$ and $s = 3$.

2) Introduce new indeterminates $x_1, x_2, x_3$. Form the equations $f_4' : \ x_4 = x_1x_2$, $f_5' : \ x_5 = x_1x_3$ and $f_6' : \ x_6 = x_2x_3$. Form the equations $f_1' : \ x_4 = x_5 + 1$, $f_2' : \ x_5 = x_6 + x_1$ and $f_3' : \ x_4 + x_5 = x_2 + 1$.

3) The standard representations of the equations $f_1', \ldots, f_6'$ are:

$$\begin{aligned} &F_1 \ : \ X_4 + X_5 - 1 = 0, \quad F_2 \ : \ X_5 - X_6 - X_1 + 2X_1X_6 = 0, \\ &F_3 \ : \ X_4 + X_5 - 2X_4X_5 + X_2 - 1 = 0, \quad F_4 \ : \ X_4 - X_1X_2 = 0, \\ &F_5 \ : \ X_5 - X_1X_3 = 0, \quad F_6 \ : \ X_6 - X_2X_3 = 0. \end{aligned}$$

Let $S_1' = \emptyset$, $S_2' = \{X_1X_6\}$, $S_3' = \{X_4X_5\}$, $S_4' = \{X_1X_2\}$, $S_5' = \{X_1X_3\}$ and $S_6' = \{X_2X_3\}$. Let $S' = \{X_1X_2, X_1X_3, X_1X_6, X_2X_3, X_4X_5\}$.

4) Introduce new real indeterminates $X_7, \ldots, X_{11}$ for $X_1X_2, X_1X_3, X_1X_6, X_2X_3, X_4X_5$

respectively. Using new real indeterminates linearize $F_i$ as follows

$$\begin{aligned} &F_1 \ : \ X_4 + X_5 - 1 = 0, \quad F_2 \ : \ X_5 - X_6 - X_1 + 2X_9 = 0, \\ &F_3 \ : \ X_4 + X_5 - 2X_{11} + X_2 - 1 = 0, \quad F_4 \ : \ X_4 - X_7 = 0, \\ &F_5 \ : \ X_5 - X_8 = 0, \quad F_6 \ : \ X_6 - X_{10} = 0. \end{aligned}$$

5) Form the linear inequalities

$$\begin{aligned} &I_7 \ : \ X_1 + X_2 - X_7 \leq 1, \ I_{11} \ : \ X_1 \geq X_7, \ I_{12} \ : \ X_2 \geq X_7, \\ &I_8 \ : \ X_1 + X_3 - X_8 \leq 1, \ I_{21} \ : \ X_1 \geq X_8, \ I_{23} \ : \ X_3 \geq X_8, \\ &I_9 \ : \ X_1 + X_6 - X_9 \leq 1, \ I_{31} \ : \ X_1 \geq X_9, \ I_{36} \ : \ X_6 \geq X_9. \\ &I_{10} \ : \ X_2 + X_3 - X_{10} \leq 1, \ I_{42} \ : \ X_2 \geq X_{10}, \ I_{43} \ : \ X_3 \geq X_{10}, \\ &I_{11} \ : \ X_4 + X_5 - X_{11} \leq 1, \ I_{54} \ : \ X_4 \geq X_{11}, \ I_{45} \ : \ X_5 \geq X_{11}. \end{aligned}$$

6) Let $I_1 : \ X_1 \leq 1$, $I_2 : \ X_2 \leq 1$ and $I_3 : \ X_3 \leq 1$.

7) Let $C = X_1 + X_2 + X_3$. Now use an IP solver to minimize $C$ subject to $\{F_1, \ldots, F_6, I_7, \ldots, I_{11}, I_{11}, I_{12}, I_{21}, I_{23}, I_{31}, I_{36}, I_{42}, I_{43}, I_{54}, I_{55}, I_1, I_2, I_3\}$.

8) Choose values for $X_1$, $X_2$ and $X_3$ from the solution provided by an IP solver. This will return $(1, 0, 1)$.

**Remark 5.2.11.** Integer Polynomial Conversion (IPC) introduces one new integer variable per term and per equation. In hope of getting more and stronger constraints one can do the following. Apply RPC to equations with no more than three terms. In this case the number of terms per equation and the number of new variables is the same as when using the IPC. But by replacing a quadratic term by a new variable we get three constraints instead of only the restriction that the variable is binary. It looks like that we get stronger constraints by using RPC in these cases. For equations with more than three terms we use the IPC. We call this strategy *Mixed Polynomial Conversion (MPC)* and is omitted. But computational experiences shows that MPC does not provide any improvement.

### 5.2.4 Experimental results

Now we present our observations and results from experiments with the algorithm in Proposition 5.2.8. We try to find a good objective function and variables which should be restricted to be binary. Furthermore, we try to see which optimization direction

(minimize or maximize) is a better choice. From now on we assume that we are in the setting of the conversion algorithm in Proposition 5.2.8. Furthermore, we assume that $\max\{\deg(f_i) \mid i \in \{1,\ldots,m\}\} \leq 2$. Before applying the conversion algorithm we use Remark 5.2.7 to split the polynomials $f_1,\ldots,f_m$ into polynomials which have maximum number of terms in their supports less than or equal to 4. The new variables introduced due to splitting will be called *auxiliary variables*. This enables us to work with quadratic polynomials during all iterations of the algorithm. Recall that splitting is used to keep the degree of converted polynomials quadratic.

### Optimization Direction

Assume that we choose the objective function as the sum over all the initial variables $X_1,\ldots,X_n$. We impose restrictions on variables as given in Remark 5.2.9, the initial variables will be forced to take binary values.

| | F-SBox | | H-SBox | | M-SBox | |
|---|---|---|---|---|---|---|
| System | Max | Min | Max | Min | Max | Min |
| CTC(3,3) | 48 | 39 | 19 | 21 | 16 | 15 |
| CTC(3,4) | 64 | 878 | 104 | 793 | 28 | 734 |
| CTC(4,3) | 133 | 612 | 92 | 487 | 66 | 386 |
| CTC(4,4) | 3737 | >5000 | 2599 | 5377 | 1266 | 15781 |

Table 5.8: GLPK time comparison for optimization direction using RPC

| | F-SBox | | H-SBox | | M-SBox | |
|---|---|---|---|---|---|---|
| System | Max | Min | Max | Min | Max | Min |
| CTC(3,3) | 3.7 | 3 | 3 | 3.2 | 5 | 4.4 |
| CTC(3,4) | 3 | 29 | 1.3 | 3.6 | 9 | 6.2 |
| CTC(4,3) | 5.8 | 32 | 9.5 | 17 | 11 | 21 |
| CTC(4,4) | 109 | 129 | 55 | 22 | 38 | 21 |
| CTC(4,5) | 229 | 390 | 99 | 174 | 270 | 391 |
| CTC(5,4) | 1458 | 516 | 245 | 425 | 158 | 1715 |

Table 5.9: CPLEX time comparison for optimization direction using RPC

Tables 5.8 and 5.9 show the experimental results for the two optimization directions. We run a number of experiments which show that more or less same observations can be obtained for different choices of objective functions. After observing the timings in Tables 5.8 and 5.9, we are still not able to give a concrete answer to this question but we are on the safe side if we choose *maximization* as optimization direction. Furthermore,

if we model S-boxes using first 7 equations out of 14, we can obtain better timings. Note that CPLEX provides best results if we use *minimization* as optimization direction and use minimum number of S-box equations.

### Restrictions on Variables

Assume that we choose the objective function as the sum over all the initial variables $X_1, \ldots, X_n$. We choose *maximization* as optimization direction. Since we are looking for a 0-1 valued solution, the initial variables $X_1, \ldots, X_n$ must be forced to take binary values. The variables $X_{n+j}$ and $X_{n+s+k}$, introduced by the conversion algorithm and the auxiliary variables depend on the initial variables. These variables can be restricted in the following ways.

| | F-SBox | | | H-SBox | | | M-SBox | | |
|---|---|---|---|---|---|---|---|---|---|
| System | R1 | R2 | R3 | R1 | R2 | R3 | R1 | R2 | R3 |
| CTC(3,3) | 99 | 48 | 99 | 132 | 19 | 29 | 41 | 16 | 18 |
| CTC(3,4) | 427 | 64 | 152 | 464 | 104 | 80 | 68 | 28 | 53 |
| CTC(4,3) | 427 | 133 | 179 | 447 | 92 | 97 | 219 | 66 | 101 |
| CTC(4,4) | >6000 | 3737 | >6000 | >6000 | 2599 | 3325 | 5427 | 1266 | 1231 |

Table 5.10: GLPK time comparison for restrictions on variables using RPC

| | F-SBox | | | H-SBox | | | M-SBox | | |
|---|---|---|---|---|---|---|---|---|---|
| System | R1 | R2 | R3 | R1 | R2 | R3 | R1 | R2 | R3 |
| CTC(3,3) | 4.2 | 3.7 | 6 | 4.2 | 3 | 2 | 12 | 5 | 2 |
| CTC(3,4) | 8.8 | 3 | 3.7 | 4 | 1.3 | 7.3 | 3.5 | 9 | 8 |
| CTC(4,3) | 25 | 5.8 | 4 | 12 | 9.5 | 14 | 6 | 11 | 11 |
| CTC(4,4) | 120 | 109 | 87 | 49 | 55 | 47 | 19 | 38 | 27 |
| CTC(4,5) | 429 | 229 | 516 | 120 | 99 | 111 | 36 | 270 | 49 |
| CTC(5,4) | 504 | 1458 | 1626 | 193 | 245 | 87 | 225 | 158 | 373 |

Table 5.11: CPLEX time comparison for restrictions on variables using RPC

**R1:** Force the variables $X_{n+j}$ and $X_{n+s+k}$ to take on binary values.

**R2:** Keep the variables $X_{n+j}$ and $X_{n+s+k}$ continuous in the interval $[0, 1]$.

**R3:** Force the auxiliary variables to take on binary values and keep the variables $X_{n+j}$ and $X_{n+s+k}$ continuous in the interval $[0, 1]$.

Tables 5.10 and 5.11 show that **R3** is a better choice for CPLEX if we model S-boxes using first 7 equations out of 14 and **R2** is a better choice for GLPK in all cases.

### Objective Function

Assume that we choose *maximization* as optimization direction and variables are restricted according to **R3**. The only restriction for the objective function is that it must be linear. Natural choices are as follows.

| | Obj. | CTC(3,3) | CTC(3,4) | CTC(4,3) |
|---|---|---|---|---|
| F-SBox | O1 | 100 | 159 | 179 |
| | O2 | 132 | 2254 | 7374 |
| | O3 | 165 | >1000 | >1000 |
| | O4 | 196 | 1461 | 2147 |
| | O5 | 111 | >1000 | 1517 |
| H-SBox | O1 | 29 | 80 | 97 |
| | O2 | 148 | 4134 | 2255 |
| | O3 | 91 | >1000 | >1000 |
| | O4 | 40 | 732 | 237 |
| | O5 | 79 | >1000 | >1000 |
| M-SBox | O1 | 18 | 53 | 101 |
| | O2 | 96 | 5395 | 3058 |
| | O3 | 62 | >1000 | >1000 |
| | O4 | 53 | 898 | >1000 |
| | O5 | 67 | >1000 | >1000 |

Table 5.12: GLPK time comparison for objective function using RPC

**O1:** The sum over all the initial variables $X_1, \ldots, X_n$.

**O2:** The sum over all variables.

**O3:** The sum over all variables except the initial variables.

**O4:** The sum over the initial variables and the auxiliary variables.

**O5:** The sum over all variables except the initial variables and the auxiliary variables.

Tables 5.12 and 5.13 show that **O1** and **O2** are better choices.

| | Obj. | CTC(3,3) | CTC(3,4) | CTC(4,3) | CTC(4,4) | CTC(4,5) | CTC(5,4) |
|---|---|---|---|---|---|---|---|
| F-SBox | O1 | 1.3 | 3.7 | 22 | 73 | 358 | 603 |
| | O2 | 14 | 2.3 | 32 | 79 | 987 | 2778 |
| | O3 | 16.7 | 2.4 | 34 | 192 | 380 | 1018 |
| | O4 | 18 | 11 | 35 | 104 | 655 | 1104 |
| | O5 | 10 | 11.4 | 34 | 139 | 535 | 491 |
| H-SBox | O1 | 3 | 5.8 | 17 | 6.3 | 243 | 658 |
| | O2 | 10 | 16 | 21 | 39 | 175 | 540 |
| | O3 | 3.7 | 21 | 19 | 103 | 81 | 564 |
| | O4 | 2.7 | 1.2 | 12.5 | 46 | 52 | 83 |
| | O5 | 9.8 | 2.2 | 21 | 95 | 131 | 420 |
| M-SBox | O1 | 6 | 8 | 10 | 26 | 17 | 525 |
| | O2 | 6.6 | 16 | 35 | 66 | 314 | 692 |
| | O3 | 6.7 | 15 | 27 | 105 | 566 | 901 |
| | O4 | 5.8 | 3.25 | 15 | 13 | 132 | 569 |
| | O5 | 6.6 | 14.6 | 28 | 75 | 1153 | 370 |

Table 5.13: CPLEX time comparison for objective function using RPC

## 5.3 Some Strategies for Polynomial Conversion

In this section we study strategies that enable the transformation of a 0-1 polynomial programming problem into a 0-1 linear programming problem to be effected with a reduced number of constraints. We review and investigate the field of MILP in search of more economic ways of transferring 0-1 programs into 0-1 linear programs. Towards the end of this section we spell out the generalized versions of IPC and RPC. This enables us to use several strategies and generalizes the approach in Section 5.2. At the end of this section experimental results are presented and compared.

Non-linearities in integer programming are also handled by involving the transformation of a nonlinear function into a polynomial function of 0-1 variables [92, 17], and then transforming the polynomial function into a linear function of 0-1 variables [17, 171, 175]. This approach is some times called *transformed linear approach*. The transformed linear approach involves some standard procedures for linearizing nonlinear integer problems. We study these standard approaches to achieve more economical linear representations of 0-1 polynomial programming problems. Thus the purpose is

to give procedures for achieving improved linear representations of nonlinearities by giving special attention to problems described in Section 5.2. The idea is to introduce new 0-1 variables to take the place of the nonlinear terms, simultaneously introducing auxiliary constraints to insure that the new variables will assume the appropriate values. Using this idea a quadratic polynomial of $n$ binary variables can be linearized using $\mathcal{O}(n)$ additional variables and constraints.

### 5.3.1 Polynomial Conversion Strategies

Let $\mathbb{R}[X_1, \ldots, X_n]$ be the polynomial ring over $\mathbb{R}$. Recall Definition 5.2.1, for a squarefree term $T_j \in \mathbb{T}^n$ the number of elements in the indexing set $N_j$ is the degree of $T_j$. From now on, we will be working with polynomials $F$ of the form $F = \sum_{j=1}^{s} T_j + L \in P = \mathbb{R}[X_1, \ldots, X_n]$, where $L \in P_{\leq 1}$, $T_j \in \mathbb{T}^n$ and $X_i \in \{0, 1\}$ for $i = 1, \ldots, n$. In particular, $F$ will be a squarefree polynomial. Recall that by a squarefree polynomial we mean a polynomial which has all terms squarefree in its support.

**Definition 5.3.1.** Let $F = \sum_{j=1}^{s} T_j + L \in P = \mathbb{R}[X_1, \ldots, X_n]$, where $L \in P_{\leq 1}$ and $T_j \in \mathbb{T}^n$, be a squarefree polynomial.

a) We denote the set of all nonlinear terms in the support of $F$ by $\mathcal{T} = \{T_1, \ldots, T_s\}$.

b) The **index set** of all nonlinear terms in the support of $F$ is $\mathcal{N} = \{N_1, \ldots, N_s\}$.

**Definition 5.3.2.** Let $F \in \mathbb{R}[X_1, \ldots, X_n]$ be a polynomial. A system of linear equations and inequalities $\mathcal{A}x \leq b$, where $x \in \mathbb{Z}_+^n$, $b$ is a vector over $\mathbb{R}$ and $\mathcal{A}$ is a matrix over $\mathbb{R}$, is called a **linearization** of $F$ if every tuple $(a_1, \ldots, a_n) \in \mathbb{Z}_+^n$ such that $F(a_1, \ldots, a_n) = 0$ corresponds uniquely to a non-negative integer solution of the system $\mathcal{A}x \leq b$.

Our objective is to study different strategies for formulating the polynomial $F$ as a system of linear equalities and inequalities. The following example will be used time and again to give an explicit application of the strategies studied in this section.

**Example 5.3.3.** Consider the following mixed integer nonlinear programming problem.

$$\begin{aligned} \text{Minimize} \quad & Z = X_1 + X_2 + X_3 + X_4 \\ \text{Subject to} \quad & X_1X_2X_3X_4 - X_1X_2X_3 + X_1X_2 \leq 17 \\ & X_1X_2X_3X_5 + X_1X_2X_4 - X_1X_3 + 25 \leq 73 \\ & X_1X_2X_4X_5 - X_1X_2X_5 + X_1X_4 - 43 \leq 135 \\ & 0 \leq X_1, \ldots, X_5 \leq 1 \text{ and } X_1, \ldots, X_5 \in \mathbb{R}_+ \end{aligned} \tag{5.6}$$

In the above nonlinear model we have $\mathcal{T} = \{X_1X_2X_3X_4, X_1X_2X_3, X_1X_2, X_1X_2X_4X_5, X_1X_2X_5, X_1X_4, X_1X_2X_4X_5, X_1X_2X_5, X_1X_4\}$ and $\mathcal{N} = \{\{1,2,3,4\}, \{1,2,3\}, \{1,2\}, \{1,2,4,5\}, \{1,2,5\}, \{1,4\}, \{1,2,4,5\}, \{1,2,5\}, \{1,4\}\}$.

The study of the transformation of a 0-1 polynomial programming problem into a 0-1 linear programming problem was initiated by E. Balas [17], W.I. Zangwill [175] and L.J. Watters [171]. They addressed the problem of accommodating nonlinear terms in the support of a polynomial $F \in K[X_1, \ldots, X_n]$ by introducing new constraints and variables as given by the following lemma. The objective is to replace the terms $T_j$ in the support of $F$ with new variables simultaneously introducing new inequalities.

**Lemma 5.3.4.** *Let* $F = \sum_{j=1}^{s} T_j + L \in P = \mathbb{R}[X_1, \ldots, X_n]$, *where* $L \in P_{\leq 1}$, $T_j \in \mathbb{T}^n$, *and* $X_i \in \{0,1\}$ *for* $i = 1, \ldots, n$. *For* $j = 1, \ldots, s$, *let*

$$I_{n+j} : \sum_{k \in N_j} X_k - X_{n+j} \leq |N_j| - 1, \tag{5.7}$$

$$J_{n+j} : \; -\sum_{k \in N_j} X_k + |N_j| X_{n+j} \leq 0. \tag{5.8}$$

*Then the solutions of the IP problem defined by the system* $\{\sum_{j=1}^{s} X_{n+j} + L, I_{n+j}, J_{n+j}\}$ *of equalities and inequalities correspond uniquely to the tuples* $(a_1, \ldots, a_n) \in \{0,1\}^n$ *which satisfy* $F$.

*Proof.* First note that for $i = 1 \ldots, n$, we have $X_i \in \{0,1\}$. Let $T_j = X_1 \ldots X_\mu \in \mathcal{T}$. If at least one of $X_1, \ldots, X_\mu$ is zero then 5.7 is nonrestrictive and 5.8 becomes $X_{n+j} < 1$. Therefore $X_{n+j} = 0$. On the other hand if $X_1 = \ldots = X_n = 1$ then 5.7 becomes $1 \leq X_{n+j}$ and 5.8 becomes $X_{n+j} \leq 1$. Therefore equality holds, i.e $X_{n+j} = 1$. Hence we have $X_{n+j} = 0$ or 1. □

Several papers have been devoted to obtaining smaller sets of constraints logically equivalent to the constraints in Lemma 5.3.4. In [86], Glover and Woolsey showed that

certain sets of nonlinear terms can be accommodated by introducing fewer additional constraints than proposed in [17, 175, 171]. They provide certain rules that, under certain conditions, make it possible to replace constraints 5.7 and 5.8 by some other equivalent constraints. For the substitution to be valid, either constraints of type 5.7 or of type 5.8 may be replaced but not both simultaneously. We discus in detail both cases one by one. Let us first fix constraints of type 5.8 and try to see equivalent constraints for the constraints of type 5.7. The explicit justification for the equivalences is elementary, but tedious.

**Lemma 5.3.5.** *Let* $F = \sum_{j=1}^{s} T_j + L \in P = \mathbb{R}[X_1, \ldots, X_n]$, *where* $L \in P_{\leq 1}$, $T_j \in \mathbb{T}^n$, *and* $X_i \in \{0, 1\}$ *for* $i = 1, \ldots, n$. *Let* $S = \{N_j \mid N_j = Q \cup \{k\}, k \in Q'\} \subseteq \mathcal{T}$, *where* $Q \subset \{1, \ldots, n\}$ *and* $Q' \subset \{1, \ldots, n\} \setminus Q$. *In Lemma 5.3.4 replace the constraints 5.7 for all* $N_j$ *belonging to* $S$ *by the constraints*

$$I'_{n+j} \quad : \quad |S| \sum_{k \in Q} X_k + \sum_{k \in Q'} X_k - \sum_{j | N_j \in S} X_{n+j} \leq |S| \cdot |Q|. \tag{5.9}$$

*Then the solutions of the IP problem defined by the system* $\{\sum_{j=1}^{s} X_{n+j} + L, I'_{n+j}, J_{n+j}\}$ *of equalities and inequalities correspond uniquely to the tuples* $(a_1, \ldots, a_n) \in \{0, 1\}^n$ *which satisfy* $F$.

*Proof.* We want to show that the conjunction of 5.9 and 5.8 achieves the same result as the conjunction of 5.7 and 5.8. Notice that inequalities of type 5.9 are obtained by summing inequalities of type 5.7. Further justification rests on the fact that the constraint that replaces the corresponding constraints of 5.7 compels the sum of the terms of these latter constraints to be greater than or equal to the number of such terms that equal unity. Since the constraints of type 5.8 are not replaced, the terms that should be 0 will in fact attain this value, and thus the new constraint forces all remaining terms to 1, as desired. □

**Example 5.3.6.** (Continued) Consider Example 5.3.3 again. The quadratic and cubic terms can be accommodated as follows.

a) To replace the constraints 5.7 of Lemma 5.3.4 with a single constraint 5.9, as in Lemma 5.3.5 that will accommodate the three terms $X_1X_2$, $X_1X_3$, and $X_1X_4$, let $Q = \{1\}$ and $Q' = \{2, 3, 4\}$. Then $S = \{\{1, 2\}, \{1, 3\}, \{1, 4\}\}$ and 5.6 becomes $3X_1 + X_2 + X_3 + X_4 - X_5 - X_6 - X_7 \leq 3$, where the new variables $X_5$, $X_6$, and $X_7$ correspond to the terms $X_1X_2$, $X_1X_3$, and $X_1X_4$ respectively.

b) To replace the constraints 5.7 of Lemma 5.3.4 with a single constraint 5.9, as in Lemma 5.3.5 that will accommodate the three terms $X_1X_2X_3$, $X_1X_2X_4$, and $X_1X_2X_5$, let $Q = \{1, 2\}$ and $Q' = \{3, 4, 5\}$. Then $S = \{\{1, 2, 3\}, \{1, 2, 4\}, \{1, 2, 5\}\}$ and 5.6 becomes $3X_1+X_2+X_3+X_4+X_5-X_6-X_7-X_8 \leq 6$, where the new variables $X_6$, $X_7$, and $X_8$ correspond to the terms $X_1X_2X_3$, $X_1X_2X_4$, and $X_1X_2X_5$ respectively.

**Lemma 5.3.7.** *Let* $F = \sum_{j=1}^{s} T_j + L \in P = \mathbb{R}[X_1, \ldots, X_n]$, *where* $L \in P_{\leq 1}$, $T_j \in \mathbb{T}^n$, *and* $X_i \in \{0, 1\}$ *for* $i = 1, \ldots, n$. *Consider index sets* $Q \subset \{1, \ldots, n\}$ *and* $Q' \subset \{1, \ldots, n\} \setminus Q$. *Let* $R = \{W \mid |W| = |Q'| - 1, W \subset Q'\}$. *Let* $S = \{N_j \mid N_j = W \cup Q,\ W \in R\} \subseteq \mathcal{T}$. *In Lemma 5.3.4 replace the constraints 5.7 for all* $N_j$ *belonging to* $S$ *by the constraint:*

$$I''_{n+j} \quad : \quad |S| \sum_{k \in Q} X_k + (|S| - 1) \sum_{k \in Q'} X_k - \sum_{j | N_j \in S} X_{n+j} \leq |S|(|S| + |Q| - 2). \quad (5.10)$$

*Then the solutions of the IP problem defined by the system* $\{\sum_{j=1}^{s} X_{n+j} + L, I''_{n+j}, J_{n+j}\}$ *of equalities and inequalities correspond uniquely to the tuples* $(a_1, \ldots, a_n) \in \{0, 1\}^n$ *which satisfy* $F$.

*Proof.* We want to show that the conjunction of 5.10 and 5.8 achieves the same result as the conjunction of 5.7 and 5.8. Again notice that inequalities of type 5.10 are obtained by summing inequalities of type 5.7. Further justification rests on the fact that the constraint that replaces the corresponding constraints of 5.7 compels the sum of the terms of these latter constraints to be greater than or equal to the number of such terms that equal unity. Since the constraints of type 5.8 are not replaced, the terms that should be 0 will in fact attain this value, and thus the new constraint forces all remaining terms to 1, as desired. □

Note that if the terms with index sets $Q$ and $Q'$ do not appear in Lemmas 5.3.5 and 5.3.7, constraints of type 5.4 associated with them must be added.

**Example 5.3.8.** (Continued) Consider Example 5.3.3 again. The terms of degree three and four can be accommodated as follows.

a) To replace the constraints 5.7 of Lemma 5.3.4 with a single constraint 5.10 as in Lemma 5.3.7 that will accommodate the three terms $X_1X_2X_3$, $X_1X_2X_4$, and $X_1X_3X_4$, let $Q = \{1\}$ and $Q' = \{2, 3, 4\}$. Then $R = \{\{2, 3\}, \{2, 4\}, \{3, 4\}\}$, $S = \{\{1, 2, 3\}, \{1, 2, 4\}, \{1, 3, 4\}\}$, and 5.7 becomes $3X_1 + 2X_2 + 2X_3 + 2X_4 -$

$X_5 - X_6 - X_7 \leq 6$, where the new variables $X_5$, $X_6$, and $X_7$ correspond to the terms $X_1X_2X_3$, $X_1X_2X_4$ and $X_1X_3X_4$ respectively.

b) To replace the constraints 5.7 of Lemma 5.3.4 with a single constraint 5.10 as in Lemma 5.3.5 that will accommodate the three terms $X_1X_2X_3X_4$, $X_1X_2X_3X_5$, and $X_1X_2X_4X_5$, let $Q = \{1, 2\}$ and $Q' = \{3, 4, 5\}$. Then $R = \{\{3, 4\}, \{3, 5\}, \{4, 5\}\}$, $S = \{\{1, 2, 3, 4\}, \{1, 2, 3, 5\}, \{1, 2, 4, 5\}\}$, and 5.7 becomes $3X_1+3X_2+2X_3+2X_4+2X_5 - X_6 - X_7 - X_8 \leq 9$, where the new variables $X_6$, $X_7$, and $X_8$ correspond to the terms $X_1X_2X_3X_4$, $X_1X_2X_3X_5$, and $X_1X_2X_4X_5$ respectively.

Let us start the next part of our journey. Now we fix constraints of type 5.7 and try to see equivalent constraints for the constraints of type 5.8.

**Lemma 5.3.9.** *Let $F = \sum_{j=1}^{s} T_j + L \in P = \mathbb{R}[X_1, \ldots, X_n]$, where $L \in P_{\leq 1}$, $T_j \in \mathbb{T}^n$, and $X_i \in \{0, 1\}$ for $i = 1, \ldots, n$. Let $S = \{N_j \mid N_j = Q \cup \{k\}, k \in Q'\} \subseteq \mathcal{T}$, where $Q \subset \{1, \ldots, n\}$ and $Q' \subset \{1, \ldots, n\} \setminus Q$. In Lemma 5.3.4 replace the constraints 5.8 for all $N_j$ belonging to $S$ by the constraints*

$$J'_{n+j} : \ -\sum_{k \in Q'} X_k + \sum_{j | N_j \in S} X_{n+j} \leq 0, \tag{5.11}$$

$$J'_{n+j} : \ -|S| X_Q + \sum_{j | N_j \in S} X_{n+j} \leq 0, \tag{5.12}$$

*where $X_Q$ is the variable associated with $\prod_{k \in Q} X_k$. Then the solutions of the IP problem defined by the system $\{\sum_{j=1}^{s} X_{n+j} + L, I_{n+j}, J'_{n+j}\}$ of equalities and inequalities correspond uniquely to the tuples $(a_1, \ldots, a_n) \in \{0, 1\}^n$ which satisfy $F$.*

*Proof.* Notice that a single inequality of type 5.8 is obtained by summing an inequality of type 5.11 and an inequality of type 5.12. Further justification is the same as in Lemma 5.3.5. □

**Example 5.3.10.** (Continued) Consider Example 5.3.3 again. To replace the constraints 5.8 of Lemma 5.3.4 with the constraints of Lemma 5.3.9, that will accommodate the three terms $X_1X_2$, $X_1X_3$, and $X_1X_4$, let $Q = \{1\}$ and $Q' = \{2, 3, 4\}$. We have $S = \{\{1, 2\}, \{1, 3\}, \{1, 4\}\}$. Then constraints 5.8 and 5.9 become $-X_2 - X_3 - X_4 + X_5 + X_6 + X_7 \leq 0$, and $-3X_1 + X_5 + X_6 + X_7 \leq 0$ respectively. The new variables $X_5$, $X_6$, and $X_7$ correspond to the terms $X_1X_2$, $X_1X_3$, and $X_1X_4$ respectively.

**Lemma 5.3.11.** *Let* $F = \sum_{j=1}^{s} T_j + L \in P = \mathbb{R}[X_1, \ldots, X_n]$, *where* $L \in P_{\leq 1}$, $T_j \in \mathbb{T}^n$, *and* $X_i \in \{0, 1\}$ *for* $i = 1, \ldots, n$. *Consider index sets* $Q \subset \{1, \ldots, n\}$ *and* $Q' \subset \{1, \ldots, n\} \setminus Q$. *Let* $R = \{W \mid |W| = |Q'| - 1, W \subset Q'\}$. *Let* $S = \{N_j \mid N_j = W \cup Q,\ W \in R\} \subseteq \mathcal{T}$. *In Lemma 5.3.4 replace the constraints 5.8 for all* $N_j$ *belonging to* $S$ *by the constraints:*

$$J''_{n+j}: \ -|S|X_q + \sum_{j|N_j \in S} X_{n+j} \leq 0, \tag{5.13}$$

$$J''_{n+j}: \ \sum_{k \in Q'} X_k - |S|X_{Q'} + (|S| - 1) \sum_{j|N_j \in S} X_{n+j} \leq 0, \tag{5.14}$$

*where* $X_Q$ *(*$X_{Q'}$*) is the variable associated with* $\prod_{k \in Q} X_k$ *(*$\prod_{k \in Q'} X_k$*). Then the solutions of the IP problem defined by the system* $\{\sum_{j=1}^{s} X_{n+j} + L, I_{n+j}, J''_{n+j}\}$ *of equalities and inequalities correspond uniquely to the tuples* $(a_1, \ldots, a_n) \in \{0, 1\}^n$ *which satisfy* $F$.

*Proof.* Same as the proof of Lemma 5.3.9. □

Again note that if the terms with index sets $Q$ and $Q'$ do not appear in the Lemmas 5.3.9 and 5.3.11, constraints of type 5.8 associated with them must be added. The Lemmas 5.3.5 and 5.3.7 can be viewed as a counterpart of Lemmas 5.3.9 and 5.3.11 respectively.

**Example 5.3.12.** (Continued) Consider Example 5.3.3 again. To replace the constraints 5.5 of Lemma 5.3.4 with the constraints of Lemma 5.3.11, that will accommodate the three terms $X_1X_2X_3$, $X_1X_2X_4$, and $X_1X_3X_4$, let $Q = \{1\}$ and $Q' = \{2, 3, 4\}$. We have $R = \{\{2, 3\}, \{2, 4\}, \{3, 4\}\}$ and $S = \{\{1, 2, 3\}, \{1, 2, 4\}, \{1, 3, 4\}\}$. Then 5.12 and 5.13 become $-3X_1 + X_5 + X_6 + X_7 \leq 0$ and $-X_2 - X_3 - X_4 - 3X_{Q'} + 2X_1 \leq 6$ respectively. The new variables $X_5$, $X_6$, and $X_7$ correspond to the terms $X_1X_2X_3$, $X_1X_2X_4$, and $X_1X_3X_4$ respectively. The variable $X_{Q'}$ is the variable associated with $X_2X_3X_4$.

In another paper, Glover and Woolsey [87] propose to linearize the problem by adding new continuous variables, in view of the fact that the difficulty of solving a mixed-integer program depends more on the number of integer variables than on the number of continuous ones. First note that if we allow the use of more constraints than in Lemma 5.3.4, then it is easy to change each variable $X_{n+j}$ to a continuous variable that is automatically 0-1 when the original variables are 0-1. As given by the

following lemma.

**Lemma 5.3.13.** *Let* $F = \sum_{j=1}^{s} T_j + L \in P = \mathbb{R}[X_1, \ldots, X_n]$, *where* $L \in P_{\leq 1}$, $T_j \in \mathbb{T}^n$, *and* $X_i \in \{0, 1\}$ *for* $i = 1, \ldots, n$. *For* $j = 1, \ldots, s$, *let*

$$I_{n+j} : \sum_{k \in N_j} X_k - X_{n+j} \leq |N_j| - 1, \tag{5.15}$$

$$J_{jk} : \; X_k \geq X_{n+j}, \text{ for all } k \in N_j, \tag{5.16}$$

*where* $X_{n+j}$ *is continuous in* $[0, 1]$. *Then the solutions of the IP problem defined by the system* $\{\sum_{j=1}^{s} X_{n+j} + L, I_{n+j}, J_{jk}\}$ *of equalities and inequalities correspond uniquely to the tuples* $(a_1, \ldots, a_n) \in \{0, 1\}^n$ *which satisfy* $F$.

*Proof.* First note that for $i = 1 \ldots, n$, we have $X_i \in \{0, 1\}$. Let $T_j = X_1 \ldots X_\mu \in \mathcal{T}$. If at least one of $X_1, \ldots, X_\mu$ is zero then 5.15 is nonrestrictive and 5.16 implies $X_{n+j} = 0$. On the other hand if $X_1 = \cdots = X_n = 1$ then 5.15 implies $1 \leq X_{n+j}$. Therefore equality holds i.e $X_{n+j} = 1$. Hence we have $X_{n+j} \in [0, 1]$ and it takes on a value 0 or 1. □

Note that Lemma 5.3.13 has been used in Propositions 5.2.2 and 5.2.8. It is also possible to give each $X_{n+j}$ the status of a continuous variable in a much more economical fashion due to the following lemma.

**Lemma 5.3.14.** *Let* $F = \sum_{j=1}^{s} T_j + L \in P = \mathbb{R}[X_1, \ldots, X_n]$, *where* $L \in P_{\leq 1}$, $T_j \in \mathbb{T}^n$, *and* $X_i \in \{0, 1\}$ *for* $i = 1, \ldots, n$. *For* $i = 1, \ldots, n$, *let* $S_i = \{N_j \in \mathcal{N} \mid i \in N_j\} \subseteq \mathcal{T}$ *and in Lemma 5.3.13 replace the constraints 5.16 by the constraints*

$$J_i \;\; : \;\; |S_i| X_i \geq \sum_{j | N_j \in S_i} X_{n+j}, \tag{5.17}$$

*where* $X_{n+j}$ *is continuous in* $[0, 1]$. *Then the solutions of the IP problem defined by the system* $\{\sum_{j=1}^{s} X_{n+j} + L, I_{n+j}, J_j\}$ *of equalities and inequalities correspond uniquely to the tuples* $(a_1, \ldots, a_n) \in \{0, 1\}^n$ *which satisfy* $F$.

*Proof.* We want to show that the conjunction of 5.15 and 5.17 achieves the same result as the conjunction of 5.15 and 5.16. For this we need to show that 5.17 plays the role of 5.16. Let $T_j = X_1 \ldots X_\mu \in \mathcal{T}$. If one of $X_i = 0$ then 5.17 compels $X_{n+j} = 0$ for all $j$ such that $N_j \in S_i$ and is redundant otherwise. □

**Example 5.3.15.** Let $\mathcal{T} = \{X_1X_2, X_1X_3, X_1X_4, X_2X_3, X_2X_4, X_1X_2X_4, X_2X_3X_4\}$. Then we have

$$\begin{aligned}
\mathcal{N} &= \{\{1,2\},\{1,3\},\{1,4\},\{2,3\},\{2,4\},\{1,2,4\},\{2,3,4\}\},\\
S_1 &= \{\{1,2\},\{1,3\},\{1,4\},\{1,2,4\}\},\\
S_2 &= \{\{1,2\},\{2,3\},\{2,4\},\{1,2,4\},\{2,3,4\}\},\\
S_3 &= \{\{1,3\},\{2,3\},\{2,3,4\}\}\\
S_4 &= \{\{1,4\},\{2,4\},\{1,2,4\},\{2,3,4\}\}.
\end{aligned}$$

Thus, 5.14 becomes

$$\begin{aligned}
&4X_1 \geq X_5 + X_6 + X_7 + X_{10}, \quad 5X_2 \geq X_5 + X_8 + X_9 + X_{10} + X_{11},\\
&3X_3 \geq X_6 + X_8 + X_{11}, \quad 4X_4 \geq X_7 + X_8 + X_{10} + X_{11}.
\end{aligned}$$

The new variables $X_5$, $X_6$, $X_7$, $X_8$, $X_9$, $X_{10}$ and $X_{11}$ correspond to the terms in $\mathcal{T}$ respectively.

In the following we abuse the notation $\mathbb{T}^n$. We consider $\mathbb{T}^n$ as monoid of terms of $\mathbb{F}_2[x_1,\ldots,x_n]$ and $\mathbb{R}[X_1,\ldots,X_n]$. The only distinction we make is the following. An element of the monoid of terms for $\mathbb{F}_2[x_1,\ldots,x_n]$ will be denoted by $t$ and an element of the monoid of terms for $\mathbb{R}[X_1,\ldots,X_n]$ will be denoted by $T$. To end this section, we combine the choice of a standard approach with the other steps of IPC and RPC. This generalizes the approach in Section 5.2. In the following we spell out the versions which we implemented and used for application and timings.

**Proposition 5.3.16. (Integer Polynomial Conversion (IPC))**
*Let $f_1,\ldots,f_m \in P = \mathbb{F}_2[x_1,\ldots,x_n]$. Then the following instructions define an algorithm which computes a tuple $(a_1,\ldots,a_n) \in \{0,1\}^n$ whose residue class in $\mathbb{F}_2^n$ represent a zero of the 0-dimensional radical ideal $I = \langle f_1,\ldots,f_m, x_1^2+x_1,\ldots,x_n^2+x_n\rangle$.*

1) *Reduce $f_1,\ldots,f_m$ modulo the field equations, i.e. make their support squarefree. For $i = 1,\ldots,m$ let $S_i$ be the set of terms of degree $\geq 2$ in $f_i$, $s_i = \#\mathrm{Supp}(f_i)$ and let $\mathcal{T} = \bigcup_{i=1}^m S_i$.*

2) *For $i = 1,\ldots,m$, introduce a new indeterminate $K_i$ and write down the linear inequality $I_i:\ K_i \leq \lfloor s_i/2 \rfloor$.*

3) *For every $t_j \in S$, introduce a new integer indeterminate $X_{n+j}$. For $i = 1,\ldots,m$, write $f_i = \sum_j t_j + \ell_i$ where the sum extends over all $j$ such that $t_j \in S_i$ and*

*where* $\ell_i \in P_{\leq 1}$. *Form the equation* $F_i : \sum_j X_{n+j} + L_i - 2K_i = 0$, *where* $L_i \in \mathbb{Z}[X_1, \ldots, X_n]_{\leq 1}$.

4) *The new indeterminate* $X_{n+j}$ *takes the place of the nonlinear term* $t_j \in \mathcal{T}$. *Choose a suitable linearization strategy (use one of the Lemmas 5.3.4, 5.3.5, 5.3.7, 5.3.9, 5.3.11, 5.3.13 or 5.3.14) and introduce auxiliary inequalities (constraints)* $I_{jk}$ *to insure that the new indeterminate assumes the appropriate value.*

5) *For all* $\alpha \in \{1, \ldots, n\}$, *let* $I'_\alpha : X_\alpha \leq 1$.

6) *Choose a linear polynomial* $C \in \mathbb{Z}[X_\alpha, X_{n+j}, K_i]$ *and use an IP solver to find the tuple of natural numbers* $(a_\alpha, a_{n+j}, c_i)$ *which solves the system of equations and inequalities* $\{I_i, F_i, I_{jk}, I'_\alpha\}$ *and minimizes (or maximizes)* $C$.

7) *Return* $(a_1, \ldots, a_n)$ *and stop.*

*Proof.* For $\alpha = 1, \ldots, n$, we are looking for natural numbers $a_\alpha$ for which $I'_\alpha$ holds, so we have $a_\alpha \in \{0, 1\}$. Moreover, $a_{n+j} \in \{0, 1\}$ by step 4), which is based on Lemmas 5.3.4, 5.3.5, 5.3.7, 5.3.9, 5.3.11, 5.3.13, and 5.3.14. Next it follows from $F_i$ that $F_i(a_1, \ldots, a_n) = 2K_i$ is an even number, and $I_i$ is nothing but the trivial bound for $K_i$ implied by the size of the support of $f_i$. In this way the solutions of the IP problem correspond uniquely to the tuples $(a_1, \ldots, a_n) \in \{0, 1\}^n$ which satisfy the above reformulation of the given polynomial system. □

**Proposition 5.3.17. (Real Polynomial Conversion (RPC))**
*Let* $f_1, \ldots, f_m \in P = \mathbb{F}_2[x_1, \ldots, x_n]$. *Then the following instructions define an algorithm which computes a tuple* $(a_1, \ldots, a_n) \in \{0, 1\}^n$ *whose residue class in* $\mathbb{F}_2^n$ *represent a zero of the 0-dimensional radical ideal* $I = \langle f_1, \ldots, f_m, x_1^2 + x_1, \ldots, x_n^2 + x_n \rangle$.

1) *Reduce* $f_1, \ldots, f_m$ *modulo the field equations, i.e. make their support squarefree. For* $i = 1, \ldots, m$, *let* $S_i$ *be the set of terms of degree* $\geq 2$ *in* $f_i$. *Let* $S = \bigcup_{i=1}^m S_i$ *and* $s = |S|$.

2) *For every* $t_j \in S$, *introduce a new indeterminate* $x_{n+j}$ *and form the equation* $f'_{m+j} : x_{n+j} = t_j$. *For* $i = 1, \ldots, m$, *write* $f_i = \sum_j t_j + \ell_i$ *where the sum extends over all* $j$ *such that* $t_j \in S_i$ *and where* $\ell_i \in P_{\leq 1}$. *Form the equation* $f'_i : \sum_j x_{n+j} + \ell_i = 0$.

3) *For* $i = 1, \ldots, m + s$, *let* $F_i$ *be the equation which is the standard representation of* $f'_i$. *Let* $S'_i$ *be the set of terms of degree* $\geq 2$ *in* $F_i$ *and let* $\mathcal{T} = \bigcup_{i=1}^{m+s} S'_i$.

*4) For every $T_k \in \mathcal{T}$, introduce a new indeterminate $X_{n+s+k}$. For $i = 1, \ldots, m+s$, replace $T_k \in S_i'$ by $X_{n+s+k}$ in the support of $F_i$. This makes $F_i$ linear.*

*5) The new indeterminate $X_{n+s+k}$ takes the place of the nonlinear term $T_k \in \mathcal{T}$. Choose a suitable linearization strategy (use one of the Lemmas 5.3.4, 5.3.5, 5.3.7, 5.3.9, 5.3.11, 5.3.13 or 5.3.14) and introduce auxiliary inequalities (constraints) $I_{k\alpha}$ to insure that the indeterminate $X_{n+s+k}$ assumes the appropriate value.*

*6) For all $\alpha \in \{1, \ldots, n\}$, let $I_\alpha : \ X_\alpha \leq 1$.*

*7) Choose a linear polynomial $C \in \mathbb{Q}[X_\alpha, X_{n+j}, X_{n+s+k}]$ and use an IP solver to find the tuple of natural numbers $(a_\alpha, a_{n+j}, a_{n+s+k})$ which solves the system of equations and inequalities $\{F_i, I_{k\alpha}, I_\alpha\}$ and minimizes (or maximizes) $C$.*

*8) Return $(a_1, \ldots, a_n)$ and stop.*

*Proof.* For $\alpha = 1, \ldots, n$, we are looking for natural numbers $a_\alpha$ for which $I_\alpha$ holds, therefore we have $a_\alpha \in \{0, 1\}$. Similarly, we have $a_{n+j} \in \{0, 1\}$ by $I_\alpha$ and $F_{m+j}$ where $j = 1, \ldots, s$. Moreover, $a_{n+s+k} \in \{0, 1\}$ by step 5). Step 5) is based on Lemmas 5.3.4, 5.3.5, 5.3.7, 5.3.9, 5.3.11, 5.3.13 and 5.3.14. Next it follows from standard representation 5.2.5 that $F_i \in \{0, 1\}$. In this way the solutions of the IP problem correspond uniquely to the tuples $(a_1, \ldots, a_n) \in \{0, 1\}^n$ which satisfy the above reformulation of the given polynomial system. □

Note that restrictions on new variables in Propositions 5.3.16 and 5.3.17 are chosen according to the lemma used.

### 5.3.2 Experimental Results

Now we present our observations and results from experiments with the algorithms in Propositions 5.3.16 and 5.3.17. In this section we choose the objective function as the sum over all the initial variables $X_1, \ldots, X_n$, *maximization* as optimization direction. Restrictions on variables will be imposed according to the strategy used. Note that constraints given by Lemmas 5.3.4, 5.3.5, 5.3.7, 5.3.9, and 5.3.11 hold if new variables take on binary values. Whereas constraints given by Lemmas 5.3.13 and 5.3.14 hold if the new variables take on values in the interval $[0, 1]$. If we attempt to solve systems of quadratic polynomial equations, possible strategies that can be used in the setting of the algorithms in Propositions 5.3.16 and 5.3.17 are given by the following combinations.

$$\begin{array}{llll} \text{S1} & : & (I_{n+j}, J_{n+j}), & \text{S2} \ : \ (I'_{n+j}, J_{n+j}), \\ \text{S3} & : & (I_{n+j}, J'_{n+j}), & \text{S4} \ : \ (I_{n+j}, J_{jk}), \\ \text{S5} & : & (I_{n+j}, J_i), & \text{S6} \ : \ (I'_{n+j}, J_{jk}), \\ \text{S7} & : & (I'_{n+j}, J_i) & \end{array}$$

We have not considered S1, S2 and S3 for values in the interval $[0, 1]$, since they hold only for binary values. The "$*$" in the first column for a strategy means that new variables are continuous in the interval $[0, 1]$. As usual we present experimental results for the CTC cipher and model S-boxes using 7 equations out of 14.

**Experiments With IPC**

| | Obj. | CTC(3,3) | CTC(3,4) | CTC(4,3) | CTC(4,4) | CTC(4,5) | CTC(5,4) |
|---|---|---|---|---|---|---|---|
| GLPK | S4 | 10 | 31 | 61 | 2536 | - | - |
| | S5 | 10.3 | 33 | 45 | 3047 | - | - |
| | S6 | 11 | 63 | 73 | 1948 | - | - |
| | S7 | 21 | 144 | 78 | 4262 | - | - |
| | S4* | 7.5 | 21 | 52 | 1225 | - | - |
| | S5* | 10 | 32 | 53 | 8016 | - | - |
| | S6* | 9.6 | 64 | 57 | 1233 | - | - |
| | S7* | 16 | 100 | 70 | 5700 | - | - |
| CPLEX | S4 | 1.6 | 1.6 | 8.3 | 53 | 85 | 690 |
| | S5 | 6 | 4 | 9.4 | 83 | 169 | 593 |
| | S6 | 6 | 2.2 | 11 | 128 | 209 | 560 |
| | S7 | 1.5 | 5 | 15 | 84 | 272 | 42 |
| | S4* | 8.8 | 1.3 | 17 | 171 | 348 | 781 |
| | S5* | 5.6 | 1.8 | 9.5 | 170 | 391 | 918 |
| | S6* | 4.5 | 2.4 | 9.3 | 33 | 118 | 955 |
| | S7* | 7 | 6 | 9 | 171 | 414 | 1026 |

Table 5.14: IPC time comparison using different strategies

Consider the algorithm in Proposition 5.3.16. Table 5.14 shows the timings for the strategies S4, S5, S6, and S7. Table 5.15 shows the timings for the strategies S1, S2, and S3. The results in Tables 5.14 and 5.15 show that all strategies are comparable but no one seem to have a clear advantage over the others. Therefore, it could be interesting to study the performance of these strategies on systems of polynomial equations coming from ciphers other than CTC.

| | GLPK | | | CPLEX | | |
|---|---|---|---|---|---|---|
| System | S1 | S2 | S3 | S1 | S2 | S3 |
| CTC(3,3) | 10 | 33 | 42 | 1.7 | 9.6 | 6 |
| CTC(3,4) | 32 | 68 | 123 | 1.76 | 5.9 | 0.3 |
| CTC(4,3) | 61 | 84 | 206 | 8.4 | 13 | 13 |
| CTC(4,4) | 1785 | 2052 | >20000 | 13 | 82 | 153 |
| CTC(4,5) | - | - | - | 238 | 262 | 286 |
| CTC(5,4) | - | - | - | 1030 | 635 | 1891 |

Table 5.15: IPC time comparison using different strategies

**Experiments With RPC**

| | Obj. | CTC(3,3) | CTC(3,4) | CTC(4,3) | CTC(4,4) | CTC(4,5) | CTC(5,4) |
|---|---|---|---|---|---|---|---|
| GLPK | S4 | 164 | 1773 | 473 | > 6000 | - | - |
| | S5 | 79 | 225 | 208 | > 6000 | - | - |
| | S6 | 302 | 1973 | 2310 | > 6000 | - | - |
| | S7 | 85 | 367 | 719 | > 6000 | - | - |
| | S4* | 19 | 104 | 92 | 2599 | - | - |
| | S5* | 26 | 92 | 68 | 4817 | - | - |
| | S6* | 144 | 1646 | 732 | > 10000 | - | - |
| | S7* | 81 | 645 | 730 | > 5000 | - | - |
| CPLEX | S4 | 35 | 5.6 | 7.5 | 35 | 186 | 169 |
| | S5 | 3.8 | 1.3 | 4.4 | 144 | 107 | 273 |
| | S6 | 14.8 | 5.6 | 22 | 251 | 213 | 619 |
| | S7 | 5.8 | 6.8 | 20 | 155 | 386 | 1372 |
| | S4* | 3 | 1.3 | 9.5 | 55 | 99 | 244 |
| | S5* | 2.7 | 1.5 | 13 | 30 | 102 | 899 |
| | S6* | 10 | 15 | 16 | 23 | 242 | 901 |
| | S7* | 15 | 24 | 28 | 67 | 498 | 3203 |

Table 5.16: RPC time comparison using different strategies

Now consider the algorithm in Proposition 5.3.17. Table 5.16 shows the timings for the strategies S4, S5, S6, and S7. Table 5.17 shows the timings for the strategies S1, S2, and S3. From Tables 5.16 and 5.17 once again we see that all strategies are comparable but no one seem to have a clear advantage over the others.

| | GLPK | | | CPLEX | | |
|---|---|---|---|---|---|---|
| System | S1 | S2 | S3 | S1 | S2 | S3 |
| CTC(3,3) | 35 | 112 | 133 | 3.5 | 7.8 | 14 |
| CTC(3,4) | 116 | 260 | 387 | 0.4 | 4 | 7 |
| CTC(4,3) | 270 | 1026 | 632 | 16 | 5.7 | 15 |
| CTC(4,4) | $> 8000$ | $> 6000$ | $> 5000$ | 71 | 375 | 269 |
| CTC(4,5) | - | - | - | 243 | 987 | 217 |
| CTC(5,4) | - | - | - | 351 | 5969 | 1614 |

Table 5.17: RPC time comparison using different strategies

## 5.4 New Techniques for Polynomial Conversion

In the previous sections we have seen that conversion methods strongly affect the performance of an integer programming algorithm. This section is devoted to develop new conversion methods. We present a new conversion method based on propositional logic and pseudo-boolean optimization. In particular, first we review some concepts from propositional logic and then exploit the connection between propositional clauses and 0-1 inequalities to model the polynomial system over $\mathbb{F}_2$ (boolean polynomial system) as a MILP problem. This enables us to export several strategies from propositional logic to integer programming. Experimental results are presented and compared. These experiments show that our new polynomial conversion technique is at least as good as the techniques in Sections 5.2 and 5.3 and provides better results in most of the cases.

Let us first note that problems of propositional logic can be readily expressed as nonlinear 0-1 programs by associating the values 0 and 1 to false and true and using the following relations between boolean and usual product and sum: $x \wedge y = x \cdot y$, $x \vee y = x + y - xy$ (see [92]). At first sight it seems to result in huge increase in nonlinearity. Since any 0-1 nonlinear problem can be transformed into a 0-1 linear problem, it is interesting to see what we can achieve by such conversions. Leaving this question open we address some techniques that result directly in a linear problem.

We study a recent suggestion, namely to convert the system to a set of propositional logic clauses. Then we show how to model a MILP problem from a set of propositional clauses and use an IP solver to solve this problem. The first study of efficient methods for converting boolean polynomial systems to CNF clauses was presented in [18]. Later this study was extended slightly in [19, 48] and [166] but the procedure was basically unaltered. The latest effort is due to P. Jovanovic and M. Kreuzer [106]. They examined different conversion strategies, i.e. different ways to convert the polynomial system into

a satisfiability problem. Our exposition here follows that of [106].

### 5.4.1 Logical Polynomial Conversion (LPC)

In this section we let $\mathbb{F}_2$ be the field with two elements and $f \in \mathbb{F}_2[x_1, \ldots, x_n]$ a polynomial. Usually $f$ will be a squarefree polynomial, i.e. all terms in the support of $f$ will be squarefree, but this is not an essential hypothesis. Let $X = \{X_1, \ldots, X_n\}$ be a set of boolean variables (atomic formulas), and let $\widehat{X}$ be the set of all (propositional) logical formulas that can be constructed from them, i.e. all formulas involving the operations $\neg$, $\wedge$, and $\vee$.

The conversion procedure as suggested in [18] consists of the following steps.

(1) Linearize the system by introducing a new indeterminate for each term in the support of one of the polynomials.

(2) Having written a polynomial as a sum of indeterminates, introduce new indeterminates to cut it after a certain number of terms.

(3) Convert the reduced sums into their logical equivalents using a XOR-CNF conversion.

After applying step (1) of the conversion procedure, each polynomial is a sum of indeterminates, or equivalently a logical XOR. Long XOR's are known to be hard problems for SAT solvers. In step (2) of the conversion procedure we introduced new indeterminates to cut a long XOR into smaller XOR's having number of terms equal to some number $\ell$ which is called *cutting number*. The following definition describes the relation between the zeros of a polynomial and the evaluation of a logical formula.

**Definition 5.4.1.** Let $f \in \mathbb{F}_2[x_1, \ldots, x_n]$ be a polynomial. A **logical representation** of $f$ is a logical formula $F \in \widehat{X}$ such that $\varphi_a(F) = f(a_1, \ldots, a_n) + 1$ for every $a = (a_1, \ldots, a_n) \in \mathbb{F}_2$, where $\varphi_a$ denotes the boolean value of $F$ at the tuple of boolean values $a$ with $1 = \texttt{true}$ and $0 = \texttt{false}$.

This definition plays a very important role to convert the polynomial $F$ to a set of propositional logic clauses. Actually, the boolean tuples at which $F$ is satisfied correspond uniquely to zeros of $f$ in $\mathbb{F}_2^n$. The conversion proceeds by two steps. Firstly, the system of polynomials will be converted to a linear system and a set of CNF clauses that render each term (or a suitable combination of terms) equivalent to a variable in that linear system. Secondly, the linear system will be converted to an equivalent set

of clauses. For these two purposes the following two lemmas contain useful building blocks for conversion strategies.

**Lemma 5.4.2.** *Let $f \in \mathbb{F}_2[x_1, \ldots, x_n]$ be a boolean polynomial, let $F \in \widehat{X}$ be a logical representation of $f$, let $y$ be a further indeterminate, and let $Y$ be a further boolean variable. Then the logical representation of the polynomial $g = f + y$ is $G = (\neg F \Leftrightarrow Y)$.*

*Proof.* See [106], Lemma 2. □

The preceding lemma provides a foundation for the conversion algorithm. The next lemma extends it in a useful way.

**Lemma 5.4.3.** *Let $f \in \mathbb{F}_2[x_1, \ldots, x_n, y]$ be a boolean polynomial of the form $f = \ell_1 \cdots \ell_s + y$ where $1 \leq s \leq n$ and $\ell_i \in \{x_i, x_i + 1\}$ for $i = 1, \ldots, s$. Define logical formulas $L_i = X_i$ if $\ell_i = x_i$ and $L_i = \neg X_i$ if $\ell_i = x_i + 1$. Then the logical representation of $f$ is*

$$F = (\neg Y \vee L_1) \wedge \cdots \wedge (\neg Y \vee L_s) \wedge (Y \vee \neg L_1 \vee \cdots \vee \neg L_s),$$

*such that the logical formula $F$ is in conjunctive normal form (CNF) and has $s + 1$ clauses.*

*Proof.* See [106], Lemma 3. □

Due to Lemmas 5.4.2 and 5.4.3, we can define three elementary strategies to perform the first step of the conversion algorithm i.e. for converting systems of polynomials over $\mathbb{F}_2$ into linear systems and a set of CNF clauses.

**Definition 5.4.4.** Let $f \in \mathbb{F}_2[x_1, \ldots, x_n]$ be a polynomial.

(a) Introduce a new indeterminate $y$ and a new boolean variable $Y$, for each nonlinear term $t$ in the support of $f$. Substitute $y$ for $t$ in $f$ and append the clauses corresponding to $t + y$ in Lemma 5.4.3 to the set of clauses. This is called the **standard strategy (SS)**.

(b) Assume $\deg(f) = 2$. Introduce a new indeterminate $y$ and a new boolean variable $Y$ for each combination of the form $x_i x_j + x_i$ (if exists) in the support of $f$. Replace $x_i x_j + x_i$ in $f$ by $y$ and append the clauses corresponding to $x_i(x_j + 1) + y$ in Lemma 5.4.3 to the set of clauses. This is called the **linear partner strategy (LPS)**.

(c) Assume $\deg(f) = 2$. Introduce a new indeterminate $y$ and a new boolean variable $Y$ for each combination of the form $x_i x_j + x_i + x_j + 1$ (if exists) in the support of $f$. Replace $x_i x_j + x_i + x_j + 1$ in $f$ by $y$ and append the clauses corresponding to $(x_i + 1)(x_j + 1) + y$ in Lemma 5.4.3 to the set of clauses. This is called the **double partner strategy (DPS)**.

Note that the standard strategy can be used to convert any system of polynomials over $\mathbb{F}_2$, whereas the linear partner strategy and the double partner strategy can be used if the polynomials are quadratic. If the combinations of terms required by the linear partner strategy and the double linear partner strategy do not appear in the support of polynomial $f$, the standard strategy is applied. The experimental results in [106] show that the linear partner and the double linear partner strategies provide substantial speed up of SAT solvers.

**Remark 5.4.5.** We have two more strategies for replacing purely quadratic and cubic terms as given in the following.

**Quadratic Partner Substitution:** Let $f = x_i x_j + x_i x_k + y \in \mathbb{F}_2[x_1, \ldots, x_n, y]$ be a polynomial such that $i$, $j$, $k$ are pairwise distinct. Then

$$\begin{aligned} F =&(X_i \vee \neg Y) \wedge (X_j \vee X_k \vee \neg Y) \wedge (\neg X_j \vee \neg X_k \vee \neg Y) \wedge \\ &(\neg X_i \vee \neg X_j \vee X_k \vee Y) \wedge (\neg X_i \vee X_j \vee \neg X_k \vee Y) \end{aligned}$$

is a logical representation of $f$.

**Cubic Partner Substitution:** Let $f = x_i x_j x_k + x_i x_j x_l + y \in \mathbb{F}_2[x_1, \ldots, x_n, y]$, where $i$, $j$, $k$, $l$ are pairwise distinct. Then

$$\begin{aligned} F =&(X_i \vee \neg Y) \wedge (X_j \vee \neg Y) \wedge (X_k \vee X_l \vee \neg Y) \wedge (\neg X_k \vee \neg X_l \vee \neg Y) \wedge \\ &(\neg X_i \vee \neg X_j \vee \neg X_k \vee X_l \vee Y) \wedge (\neg X_i \vee \neg X_j \vee \neg X_k \vee \neg X_l \vee Y) \end{aligned}$$

is a logical representation of $f$.

For proofs of the quadratic and cubic partner strategies we refer to [106], Propositions 6 and 8. It is straightforward to formulate a conversion strategy, called the quadratic partner strategy (QPS) (respectively cubic partner strategy (CPS)), for polynomials of degree two (respectively of degree three) based on this remark. For cubic terms, it is also possible to pair them if they have just one indeterminate in common. However, this strategy apparently does not result in useful speed-ups and is omitted.

Finally, we are ready to exploit the connection between propositional clauses and 0-1 inequalities to model the polynomial system over $\mathbb{F}_2$ (boolean polynomial system) as a MILP problem. This enables us to use the strategies above to model a MILP problem.

**Lemma 5.4.6.** *Let $C = \{X_1 \vee \cdots \vee X_r \vee \neg Y_1 \vee \cdots \vee \neg Y_s \mid 1 \leq r, s \leq n\}$ be a set of clauses. Then the set $C$ is satisfiable if and only if the system of clausal inequalities $I_c = \{X_1 + \cdots + X_r - Y_1 - \cdots - Y_s \geq 1 - s \mid 1 \leq r, s \leq n\}$ together with the bounds $0 \leq X_i, Y_j \leq 1$ for all $i, j \in \{1, \ldots, n\}$, has an integer solution.*

*Proof.* Let $c \in C$ be a clause. If $c = X_1 \vee \cdots \vee X_r$ then by the definition of satisfiability at least one of the $X_i$ is true. In other words at least one of the $X_i$ is 1. This gives us the clausal inequality $X_1 + \cdots + X_r \geq 1$ together with the bounds $0 \leq X_i \leq 1$. If $c = \neg Y_1 \vee \cdots \vee \neg Y_s$ then by the definition of satisfiability at least one of the $Y_j$ is false. In other words at least one of the $1 - Y_j$ is 1. This gives us the clausal inequality $(1 - Y_1) + \cdots + (1 - Y_s) \geq 1$ together with the bounds $0 \leq Y_j \leq 1$. If $c = X_1 \vee \cdots \vee X_r \vee \neg Y_1 \vee \cdots \vee \neg Y_s$ then it follows from the first two cases that $X_1 + \cdots + X_r + (1 - Y_1) + \cdots + (1 - Y_s) \geq 1$ is the corresponding clausal inequality together with the bounds $0 \leq X_i, Y_j \leq 1$.
Therefore, the clause

$$c = X_1 \vee \cdots \vee X_r \vee \neg Y_1 \vee \cdots \vee \neg Y_s$$

can be translated into a clausal inequality

$$X_1 + \cdots + X_r + (1 - Y_1) + \cdots + (1 - Y_s) \geq 1$$

$$\text{or} \qquad X_1 + \cdots + X_r - Y_1 - \cdots - Y_s \geq 1 - s$$

and the clause set $C$ is satisfiable if and only if the corresponding system of clausal inequalities $I_c$ together with the bounds $0 \leq X_i, Y_j \leq 1$ has an integer solution. Therefore, reasoning in propositional logic can be seen as a special case of reasoning with linear inequalities in integer variables. □

To end this section, we combine the choice of a substitution strategy with the other steps of the conversion algorithm and spell out the version which we implemented and used for the applications and timings.

**Proposition 5.4.7. (Logical Polynomial Conversion (LPC))**
*Let $f_1, \ldots, f_m \in \mathbb{F}_2[x_1, \ldots, x_n]$ be a system of polynomial which has at least one zero in $\mathbb{F}_2^n$. Let $\ell \geq 3$ be the desired cutting number. Consider the following sequence of instructions.*

1) *Let $G = \emptyset$. Perform the following steps 2)−5) for $i = 1, \ldots, m$.*

2) *Repeat the following step 3) until no polynomial $g$ can be found anymore.*

3) *Find a subset of $\mathrm{Supp}(f_i)$ which defines a polynomial $g$ of the type required by the chosen conversion strategy. Introduce a new indeterminate $y_j$, replace $f_i$ by $f_i - g + y_j$, and append $g + y_j$ to $G$.*

4) *Perform the following step 5) until $\#Supp(f_i) \leq \ell$. Then append $f_i$ to $G$.*

5) *If $\#\mathrm{Supp}(f_i) > \ell$ then introduce a new indeterminate $y_j$, let $g$ be the sum of the first $\ell - 1$ terms of $f_i$, replace $f_i$ by $f_i - g + y_j$, and append $g + y_j$ to $G$.*

6) *For each polynomial in $G$, compute a logical representation in CNF and form the set of all clauses $C$ of all these logical representations.*

7) *For each clause $c \in C$ form a clausal inequality $I_c$.*

8) *For all $\alpha \in \{1, \ldots, n\}$, let $I_\alpha : \ X_\alpha \leq 1$ and for each $j$ let $I_j : \ Y_j \leq 1$.*

9) *Choose a linear polynomial $L \in \mathbb{Q}[X_i, Y_j]$ and use an IP solver to find the tuple of natural numbers $(a_i, b_j)$ which solves the system of equations and inequalities $\{I_c, I_j, I_\alpha\}$ and minimizes $C$.*

10) *Return $(a_1, \ldots, a_n)$ and stop.*

*This is an algorithm which computes a zero of the 0-dimensional radical ideal $I = \langle f_1, \ldots, f_m, x_1^2 + x_1, \ldots, x_n^2 + x_n \rangle$.*

*Proof.* It is clear that steps 2)−3) correspond to the linearization part (1) of the procedure given in the introduction of this section, and that steps 4−5) are an explicit version of the cutting part (2) of that procedure. Moreover, step 6) is based on Lemma 5.4.2, Lemma 5.4.3, or Remark 5.4.5 for the polynomials $g + y_j$ from step 3), and on the standard XOR-CNF conversion for the linear polynomials from steps 4)−5). Finally, step 7) follows from Lemma 5.4.6. The claim follows easily from these observations. □

**Remark 5.4.8.** Assume that we are in the setting of the algorithm in Proposition 5.4.7. A natural question could be to ask about the nature of the clausal inequalities in step 7). As claimed by Lemma 5.4.6, the variables $Y_j$ are continuous in the interval $[0, 1]$. Since the initial variables are forced to be binary, the variables $Y_j$ take on integer values automatically. The good news is the continuity of these variables because the difficulty of solving a mixed-integer program depends more on the number of integer variables than on the number of continues variables. Another nice property of these conversion strategies is the possibility to reduce the number of new variables. The standard strategies used in Sections 5.2 and 5.3 reduce the number of constraints but keep the number of newly introduced variables the same. Furthermore, if we look at the literature available on transferring 0-1 programs into 0-1 linear programs, reducing the number of newly introduced variables is a hot topic. We can also profit from these strategies there. In Section 5.4.2 we confirm our observations by experiments.

### 5.4.2 Experimental Results

Now we present our observations and results from experiments with the algorithm in Proposition 5.4.7. In steps 4)−5) of the algorithm we used cutting length 6. Note that cutting length may affect the running time of an IP solver. Actually, the timings seem to depend on the cutting number in a rather subtle and unpredictable way.

| | R1 | | | | R2 | | | |
|---|---|---|---|---|---|---|---|---|
| System | SS | LP | DLP | QPS | SS | LP | DLP | QPS |
| CTC(3,3) | 49 | 30 | 26 | 49 | 29 | 22 | 17 | 29 |
| CTC(3,4) | 207 | 18 | 19 | 207 | 71 | 13 | 12 | 71 |
| CTC(4,3) | 216 | 36 | 59 | 216 | 135 | 47 | 30 | 135 |
| CTC(4,4) | 6421 | 2566 | 1623 | 6422 | 4920 | 1663 | 1172 | 4920 |

Table 5.18: GLPK time comparison using LPC

We choose the objective function as the sum over all the initial variables $X_1, \ldots, X_n$, *maximization* as optimization direction and model S-boxes using 7 equations out of 14. Note that the inequalities $I_c$ in step 7) of the algorithm hold if the new variables take on values in the interval $[0, 1]$. We try to see whether it is an advantage to have binary restrictions only for the initial variables instead of for all. Therefore, the variables $Y_j$ can be restricted in the following two ways.

**R1:** Force the variables $Y_j$ to take on binary values.

| | R1 | | | | R2 | | | |
|---|---|---|---|---|---|---|---|---|
| System | SS | LP | DLP | QPS | SS | LP | DLP | QPS |
| CTC(3,3) | 4.7 | 2.4 | 1 | 4.7 | 3.7 | 3 | 1 | 3.7 |
| CTC(3,4) | 3.8 | 3.7 | 2.8 | 3.8 | 4.3 | 1 | 3 | 4.3 |
| CTC(4,3) | 6.4 | 3.5 | 3.8 | 6.2 | 13 | 3.6 | 3.7 | 13 |
| CTC(4,4) | 35 | 56 | 38 | 35 | 34 | 31 | 38 | 34 |
| CTC(4,5) | 121 | 85 | 74 | 121 | 62 | 85 | 74 | 62 |
| CTC(5,4) | 195 | 154 | 265 | 195 | 246 | 155 | 264 | 246 |

Table 5.19: CPLEX time comparison using LPC

**R2:** Keep the variables $Y_j$ continuous in the interval $[0, 1]$.

By looking at the Tables 5.18 and 5.19 we can see the SS and QPS conversions do not appear to provide improvements over the algorithms in Sections 5.2 and 5.3. But the LP and DLP conversions provide substantial improvements over the algorithms in Sections 5.2 and 5.3.

## 5.5 Hybrid Techniques for Polynomial Conversion

In this section we develop new hybrid techniques for modeling a MILP problem. These hybrid techniques combine the ideas studied in the previous sections to achieve efficiency. Experimental results are presented and compared which show that our new hybrid techniques result in further speed up of IP solvers. As we saw in Sections 5.2, 5.3 and 5.4, there are three types of conversion algorithms namely Integer Polynomial Conversion (IPC), Real Polynomial Conversion (RPC) and Logical Polynomial Conversion (LPC). These conversion algorithms can be equipped with different strategies to achieve efficiency. We saw such examples in Sections 5.3 and 5.4. In this section we consider the three types of conversion algorithms and equip them with all the strategies developed in Sections 5.3 and 5.4.

### 5.5.1 Hybrid Integer Polynomial Conversion (HIC)

First consider the algorithm in Proposition 5.2.2. As we mentioned before it is based on converting polynomial equations over $\mathbb{F}_2$ into polynomial equations over $\mathbb{Z}$. Secondly, consider the algorithm in Proposition 5.4.7 which is based on propositional logic and pseudo-boolean optimization. The two algorithms can be combined into one for possible improvements as follows.

**Proposition 5.5.1. (Hybrid Integer Conversion (HIC) )**
*Let $f_1, \ldots, f_m \in P = \mathbb{F}_2[x_1, \ldots, x_n]$. Then the following instructions define an algorithm which computes a tuple $(a_1, \ldots, a_n) \in \{0,1\}^n$ whose residue class in $\mathbb{F}_2^n$ represent a zero of the 0-dimensional radical ideal $I = \langle f_1, \ldots, f_m, x_1^2 + x_1, \ldots, x_n^2 + x_n \rangle$.*

1) *Reduce $f_1, \ldots, f_m$ modulo the field equations, i.e. make their support squarefree. Let $G = \emptyset$.*

2) *Repeat the following step 3) until no polynomial $g$ can be found anymore.*

3) *Find a subset of $\mathrm{Supp}(f_i)$ which defines a polynomial $g$ of the type required by the chosen conversion strategy. Introduce a new indeterminate $x_{n+j}$, replace $f_i$ by $f_i - g + x_{n+j}$, and append $g + x_{n+j}$ to $G$.*

4) *For each polynomial in $G$, compute a logical representation in CNF and form the set of all clauses $C$ of all these logical representations.*

5) *For each clause $c \in C$ form a clausal inequality $I_c$.*

6) *For $i = 1, \ldots, m$, let $S_i$ be the set of new indeterminates $x_{n+j}$ in $f_i$, and let $s_i = \#\mathrm{Supp}(f_i)$.*

7) *For $i = 1, \ldots, m$, introduce a new integer indeterminate $K_i$ and write down the linear inequality $I_i : \ K_i \leq \lfloor s_i/2 \rfloor$.*

8) *For $i = 1, \ldots, m$, write $f_i = \sum_j x_{n+j} + \ell_i$ where the sum extends over all $j$ such that $x_{n+j} \in S_i$ and where $\ell_i \in P_{\leq 1}$. Form the equation $F_i : \ \sum_j X_{n+j} + L_i - 2K_i = 0$, where $L_i \in \mathbb{Z}[X_1, \ldots, X_n]_{\leq 1}$.*

9) *For all $\alpha \in \{1, \ldots, n\}$, let $I'_\alpha : \ X_\alpha \leq 1$.*

10) *Choose a linear polynomial $L \in \mathbb{Z}[X_\alpha, X_{n+j}, K_i]$ and use an IP solver to find the tuple of natural numbers $(a_\alpha, a_{n+j}, c_i)$ which solves the system of equations and inequalities $\{I_i, F_i, I_c, I'_\alpha\}$ and minimizes $L$.*

11) *Return $(a_1, \ldots, a_n)$ and stop.*

*Proof.* It is clear that steps 2)−3) linearize the polynomials $f_i$ by introducing new indeterminates $x_{n+j}$. Moreover, step 4) is based on Lemma 5.4.2, Lemma 5.4.3, or Remark 5.4.5 for the polynomials $g + x_{n+j}$ from step 3), and step 5) follows from Lemma 5.4.6. In step 8) the polynomials $f_i$ are linear polynomials in the indeterminates $x_\alpha$

and $x_{n+j}$. Next it follows from $F_i$ that $F_i(a_1, \ldots, a_n) = 2K_i$ is an even number, and $I_i$ is nothing but the trivial bound for $K_i$ implied by the size of the support of $f_i$.

For $\alpha = 1, \ldots, n$, we are looking for natural numbers $a_\alpha$ for which $I'_\alpha$ holds, so we have $a_\alpha \in \{0, 1\}$. Moreover, we have $a_{n+j} \in \{0, 1\}$ by $I'_\alpha$ and steps 2) $-$ 5). In this way the solutions of the IP problem correspond uniquely to the tuples $(a_1, \ldots, a_n) \in \{0, 1\}^n$ which satisfy the above reformulation of the given polynomial system. The claim follows easily from these observations. □

Note that if we use the standard strategy, the algorithm coincides with the algorithm in Proposition 5.2.2.

### Experimental Results

Now we present our observations and results from experiments with the algorithm in Proposition 5.5.1.

| | R1 | | | | R2 | | | |
|---|---|---|---|---|---|---|---|---|
| System | SS | LP | DLP | QPS | SS | LP | DLP | QPS |
| CTC(3,3) | 10 | 7.8 | 4 | 10 | 9 | 7 | 4 | 9 |
| CTC(3,4) | 31 | 13 | 7.6 | 31 | 28 | 13 | 8 | 28 |
| CTC(4,3) | 61 | 26 | 19 | 65 | 47 | 25 | 16 | 47 |
| CTC(4,4) | 2536 | 413 | 489 | 2534 | 2021 | 398 | 532 | 2031 |

Table 5.20: GLPK time comparison using HIC

| | R1 | | | | R2 | | | |
|---|---|---|---|---|---|---|---|---|
| System | SS | LP | DLP | QPS | SS | LP | DLP | QPS |
| CTC(3,3) | 1.6 | 3.4 | 1 | 1.6 | 4 | 3.3 | 1 | 4 |
| CTC(3,4) | 1.6 | 0.3 | 1.6 | 1.6 | 2.6 | 0.3 | 1.6 | 2.7 |
| CTC(4,3) | 8.3 | 6.5 | 2 | 8 | 20 | 6.5 | 2 | 20 |
| CTC(4,4) | 53 | 53 | 46 | 53 | 106 | 53 | 46 | 106 |
| CTC(4,5) | 85 | 46 | 76 | 85 | 219 | 46 | 76 | 219 |
| CTC(5,4) | 690 | 337 | 308 | 693 | 744 | 337 | 308 | 744 |

Table 5.21: CPLEX time comparison using HIC

We choose the objective function as the sum over all the initial variables $X_1, \ldots, X_n$, *maximization* as optimization direction and model S-boxes using 7 equations out of 14. Note that the inequalities $I_c$ in step 5) of the algorithm hold if the new variables take on values in the interval $[0, 1]$. We try to see, whether it is an advantage to have binary

restrictions only for the initial variables instead of for all. Therefore, the variables $X_{n+j}$ can be restricted in the following two ways.

**R1:** Force the variables $X_{n+j}$ to take on binary values.

**R2:** Keep the variables $X_{n+j}$ continuous in the interval $[0, 1]$.

By looking at the Tables 5.18 and 5.19 we can see the new hybrid conversion algorithm (if we use LP or DLP strategy) completely beats all the previous versions of it.

### 5.5.2 Hybrid Real Polynomial Conversion (HRC)

First consider the algorithm in Proposition 5.2.8. As we mentioned before it is based on converting polynomial equations over $\mathbb{F}_2$ into polynomial equations over $\mathbb{R}$. Secondly, consider the algorithm in Proposition 5.4.7 which is based on propositional logic and pseudo-boolean optimization. The two algorithms can be combined into one along with strategies from Section 5.3 for possible improvements as follows.

**Proposition 5.5.2. (Hybrid Real Conversion (HRC) )**
*Let $f_1, \ldots, f_m \in P = \mathbb{F}_2[x_1, \ldots, x_n]$. Then the following instructions define an algorithm which computes a tuple $(a_1, \ldots, a_n) \in \{0, 1\}^n$ whose residue class in $\mathbb{F}_2^n$ represent a zero of the 0-dimensional radical ideal $I = \langle f_1, \ldots, f_m, x_1^2 + x_1, \ldots, x_n^2 + x_n \rangle$.*

*1) Reduce $f_1, \ldots, f_m$ modulo the field equations, i.e. make their support squarefree. Let $G = \emptyset$.*

*2) Repeat the following step 3) until no polynomial $g$ can be found anymore.*

*3) Find a subset of $\mathrm{Supp}(f_i)$ which defines a polynomial $g$ of the type required by the chosen conversion strategy. Introduce a new indeterminate $x_{n+j}$, replace $f_i$ by $f_i - g + x_{n+j}$, and append $g + x_{n+j}$ to $G$.*

*4) For each polynomial in $G$, compute a logical representation in CNF and form the set of all clauses $C$ of all these logical representations.*

*5) For each clause $c \in C$ form a clausal inequality $I_c$.*

*6) For $i = 1, \ldots, m$, let $S_i$ be the set of new indeterminates $x_{n+j}$ in $f_i$. Let $s$ be the total number of new variables $x_{n+j}$ introduced.*

*7) For $i = 1, \ldots, m$, write $f_i = \sum_j x_{n+j} + \ell_i$ where the sum extends over all $j$ such that $x_{n+j} \in S_i$ and where $\ell_i \in P_{\leq 1}$.*

*8) For $i = 1, \ldots, m$, let $F_i$ be the equation which is the standard representation of $f_i$. Let $S_i'$ be the set of terms of degree $\geq 2$ in $F_i$ and let $\mathcal{T} = \bigcup_{i=1}^{m} S_i'$.*

*9) For every $T_k \in \mathcal{T}$, introduce a new indeterminate $X_{n+s+k}$. For $i = 1, \ldots, m$, replace $T_k \in S_i'$ by $X_{n+s+k}$ in the support of $F_i$, this makes $F_i$ linear.*

*10) The new indeterminate $X_{n+s+k}$ takes the place of the nonlinear term $T_k \in \mathcal{T}$. Choose a suitable linearization strategy (use one of the Lemmas 5.3.4, 5.3.5, 5.3.7, 5.3.9, 5.3.11, 5.3.13, or 5.3.14) and introduce auxiliary inequalities (constraints) $I_{k\alpha}$ to insure that the indeterminate $X_{n+s+k}$ assumes the appropriate value.*

*11) For all $\alpha \in \{1, \ldots, n\}$, let $I_\alpha : \ X_\alpha \leq 1$.*

*12) Choose a linear polynomial $L \in \mathbb{Q}[X_\alpha, X_{n+j}, X_{n+s+k}]$ and use an IP solver to find the tuple of natural numbers $(a_\alpha, a_{n+j}, a_{n+s+k})$ which solves the system of equations and inequalities $\{I_c, F_i, I_{k\alpha}\}$ and minimizes $L$.*

*11) Return $(a_1, \ldots, a_n)$ and stop.*

*Proof.* It is clear that steps 2)−3) linearize the polynomials $f_i$ by introducing new indeterminates $x_{n+j}$. Moreover, step 4) is based on Lemma 5.4.2, Lemma 5.4.3 or Remark 5.4.5 for the polynomials $g + x_{n+j}$ from step 3). And step 7) follows from Lemma 5.4.6. In step 6) the polynomials $f_i$ are linear polynomials in the indeterminates $x_\alpha$ and $x_{n+j}$.

For $\alpha = 1, \ldots, n$, we are looking for natural numbers $a_\alpha$ for which $I_\alpha$ holds, therefore we have $a_\alpha \in \{0, 1\}$. Similarly, we have $a_{n+j} \in \{0, 1\}$ by $I_\alpha$ and steps 2) − 5). Moreover, $a_{n+s+k} \in \{0, 1\}$ by step 10), which is based on Lemmas 5.3.4, 5.3.5, 5.3.7, 5.3.9, 5.3.11, 5.3.13, and 5.3.14. Next it follows from the standard representation 5.2.5 that $F_i \in \{0, 1\}$. In this way the solutions of the IP problem correspond uniquely to the tuples $(a_1, \ldots, a_n) \in \{0, 1\}^n$ which satisfy the above reformulation of the given polynomial system. The claim follows easily from these observations. □

**Remark 5.5.3.** Assume that we are in the setting of the algorithm in Proposition 5.5.2. Note that if $\max\{\deg(f_i) \mid i \in \{1, \ldots, m\}\} \leq 2$ and for $i = 1, \ldots, m$, the maximum number of terms in the support of $f_i$ does not exceed 4, the algorithm

works with quadratic polynomials in all of its iterations. Therefore, before applying the conversion algorithm we can use Remark 5.2.7 to split the polynomials $f_1, \ldots, f_m$ into polynomials which have maximum number of terms in their supports less than or equal to 4. But this can reduce the effectiveness of step 3). Thus it would be better if we use Remark 5.2.7 after steps 1)−5). Furthermore, we remark that in some cases HRPC seems to provide infeasible MILP model. The exact reason is not known but it turns out that the new auxiliary variables introduced while splitting should be forced to take on binary values to avoid infeasibility.

**Experimental Results**

Now we present our observations and results from experiments with the algorithm in Proposition 5.5.2. From now on we assume that we are in the setting of the conversion algorithm in Proposition 5.2.8. Furthermore, we split polynomials after steps 1)−5) as explained in Remark 5.5.3. This enables us to work with quadratic polynomials in all iterations of the algorithm. Furthermore, we force the auxiliary variables introduced while splitting to take binary values. Since the effect of strategies in step 10) of the algorithm is already studied in Section 5.3.2, we only consider the strategies in step 3) of the algorithm. In step 10) we use the strategy S4 (see Section 5.3.2).

| | R1 | | | | R2 | | | |
|---|---|---|---|---|---|---|---|---|
| System | SS | LP | DLP | QPS | SS | LP | DLP | QPS |
| CTC(3,3) | 125 | 64 | 48 | 125 | 21 | 16 | 19 | 21 |
| CTC(3,4) | 321 | 74 | 51 | 321 | 76 | 10 | 15 | 76 |
| CTC(4,3) | 380 | 153 | 107 | 381 | 51 | 21 | 32 | 51 |
| CTC(4,4) | $> 15000$ | 11537 | 10976 | $> 15000$ | 3375 | 1830 | 1094 | 3375 |

Table 5.22: GLPK time comparison using HRC

| | R1 | | | | R2 | | | |
|---|---|---|---|---|---|---|---|---|
| System | SS | LP | DLP | QPS | SS | LP | DLP | QPS |
| CTC(3,3) | 3.5 | 2.8 | 3 | 3.5 | 2.2 | 6.2 | 4.8 | 2.2 |
| CTC(3,4) | 0.84 | 0.8 | 3.6 | 0.8 | 3.6 | 4.9 | 5.3 | 3.7 |
| CTC(4,3) | 18 | 14 | 6.3 | 18 | 10 | 15 | 4.3 | 10 |
| CTC(4,4) | 74 | 51 | 27 | 74 | 64 | 26 | 1.6 | 64 |
| CTC(4,5) | 91 | 111 | 367 | 91 | 128 | 59 | 50 | 128 |
| CTC(5,4) | 509 | 205 | 51 | 509 | 215 | 241 | 225 | 215 |

Table 5.23: CPLEX time comparison using HRC

We choose the objective function as the sum over all the initial variables $X_1, \ldots, X_n$, *maximization* as optimization direction and model S-boxes using 7 equations out of 14. Note that the inequalities $I_c$ in step 5) of the algorithm hold if the new variables take on values in the interval $[0, 1]$ and the inequalities $I_{k\alpha}$ in step 10) of the algorithm hold if the new variables take on values as given by the lemma used. We try to see whether it is an advantage to have binary restrictions only for the initial variables instead of for all. We force the initial variables $X_1, \ldots, X_n$ to take on binary values. The remaining new variables $X_{n+j}$ and $X_{n+s+k}$ can be restricted in the following two ways.

**R1:** Force the variables $X_{n+j}$ and $X_{n+s+k}$ to take on binary values.

**R2:** Keep the variables $X_{n+j}$ and $X_{n+s+k}$ continuous in the interval $[0, 1]$.

By looking at the Tables 5.22 and 5.23 we can see the new hybrid conversion algorithm (if we use LP or DLP strategy) surpass old boundaries set by RPC.

## 5.6 Comparison Using Plots and Tables

To conclude this chapter, we present a comparison using plots and tables of all techniques studied in this chapter. In the following we consider the CTC and the small scale AES cipher one by one and illustrate our experimental results.

### 5.6.1 The Courtois Toy Cipher (CTC)

First of all recall Tables 5.7, 5.14, and 5.23. From the tables we can see under specified choice of objective function and restrictions on variables, CTC(4,4) takes 1.6 seconds, CTC(4,5) takes 8.5 seconds and CTC(5,4) takes 42 seconds. This means that if we try all possible strategies for a particular instance of polynomial system, it could be easier to solve. In fact the idea is to attack the required corner (that gives a solution of the original system) of the polyhedron. Different strategies can lead to different consequences. The strategies which work well for one instance may not work well for the other instance. Furthermore, some times change of an objective function along with restrictions on variables dramatically decrease the running time of an IP solver. Finally, we see for sparse systems the running time of the MILP technique compare favorably to the running times of the Gröbner basis technique and the linear algebra technique in Chapter 3. Even for examples involving many indeterminates, such as

| System | CTC(5,5) | CTC(5,6) | CTC(6,5) | CTC(6,6) | CTC(6,7) | CTC(7,6) |
|---|---|---|---|---|---|---|
| Equations | 605 | 705 | 708 | 864 | 984 | 987 |
| Variables | 330 | 375 | 378 | 468 | 522 | 525 |

Table 5.24: size of CTC instances

| | IPC | RPC | LPC | | | HIC | | HRC |
|---|---|---|---|---|---|---|---|---|
| Strategy | | | SS | LP | DLP | LP | DLP | SS |
| CTC(5,5) | 2708 | 1356 | 6222 | 691 | 679 | 1798 | 552 | 480 |
| CTC(5,6) | 3088 | 1227 | 3104 | 270 | 1875 | 9332 | 2421 | 1041 |
| CTC(6,5) | 15656 | 7743 | 20452 | 15540 | 16407 | 14661 | 11621 | 10723 |
| CTC(6,6) | 45272 | 25978 | 23841 | 16941 | 12264 | 10716 | 16757 | 11572 |
| CTC(6,7) | 26209 | 9224 | 258661 | 30868 | 18660 | 2285 | 11031 | 7716 |
| CTC(7,6) | 221986 | 11904 | 55766 | 91358 | 97985 | 68146 | 9436 | 73090 |

Table 5.25: CLPEX time comparison for different conversions

given in Table 5.25, the above timings completely beat individually tailored Gröbner basis methods, such as the ones reported in [3].

In all plots we choose objective function as the sum over all the initial variables, *maximization* as optimization direction and model S-boxes using 7 equations out of 14. We plot CTC(3,3), CTC(3,4), CTC(4,3), CTC(4,4), CTC(4,5) and CTC(5,4) on horizontal axis and timings in seconds on vertical axis. Figures 5.2 and 5.3 represent the timings for IPC with binary and continues restrictions respectively. Figures 5.4 and 5.5 represent the timings for RPC with binary and continuous restrictions respectively. Furthermore, we have considered only those strategies which fit well in plots to compare results. In all plots the solid black curve represents the timings for standard versions of IPC and RPC as given in Section 5.1. All other curves and marks represent our contribution in this chapter. Furthermore, note that some IP solvers like CPLEX can be parallelized. Thus we can benefit from parallelization capabilities of IP solvers to solve systems of polynomial equations. For instance, on a computer with 48 processors, each one of 2.1 GHz (AMD Opteron 6172 processor), CTC(7,7) can be solved for one solution in 34 hours using RPC.

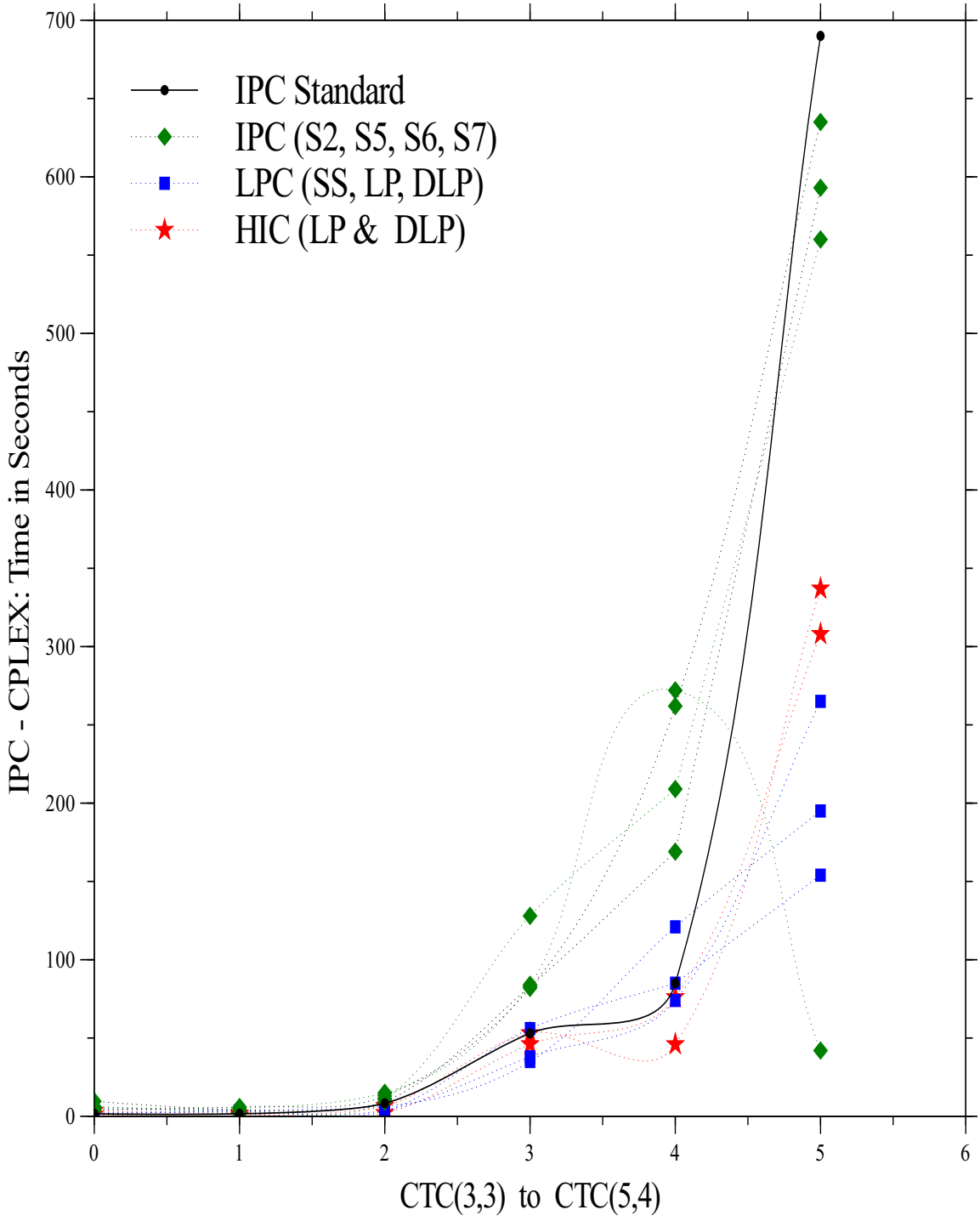



Figure 5.2: IPC with binary restrictions on variables

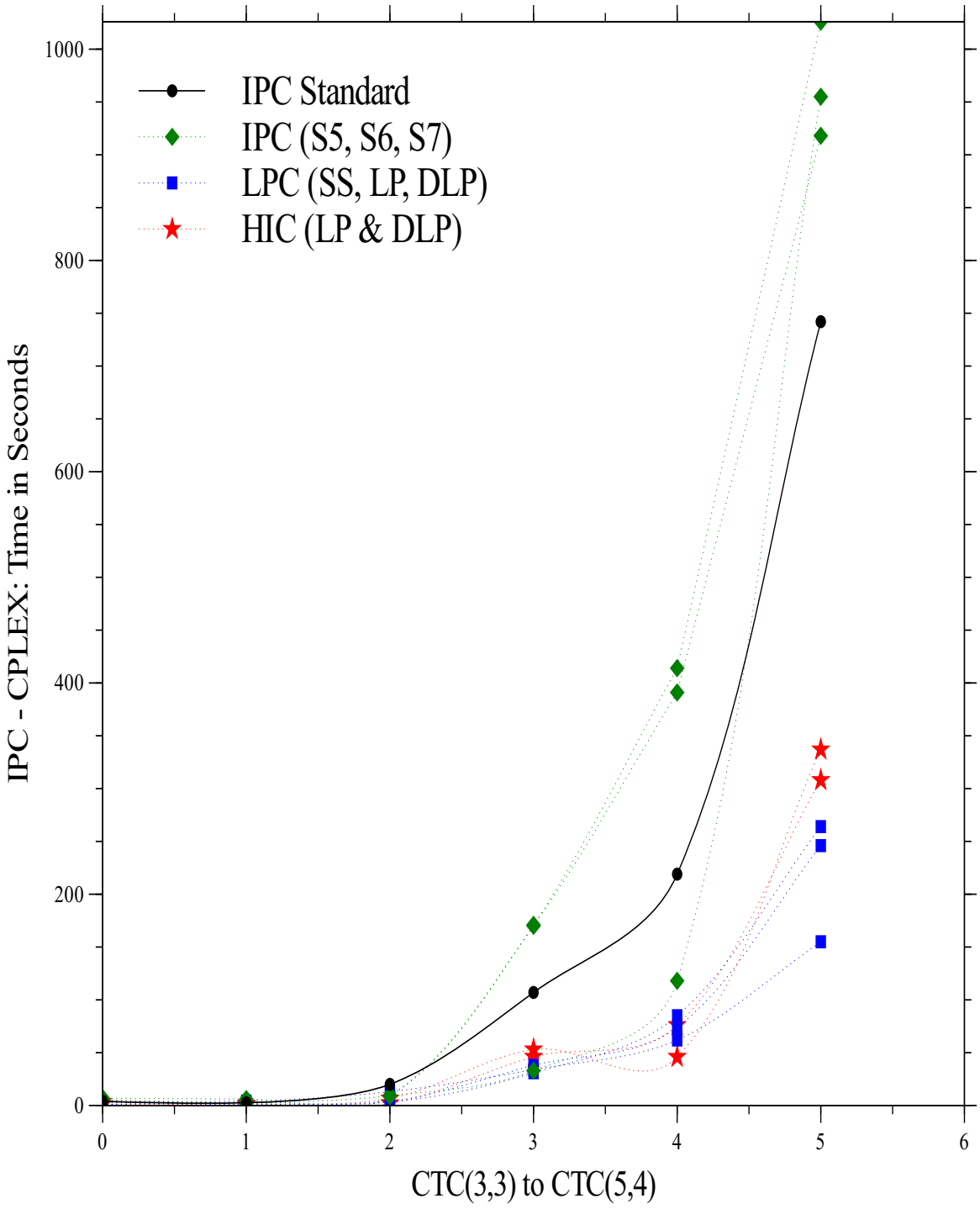



Figure 5.3: IPC with continues restrictions on variables

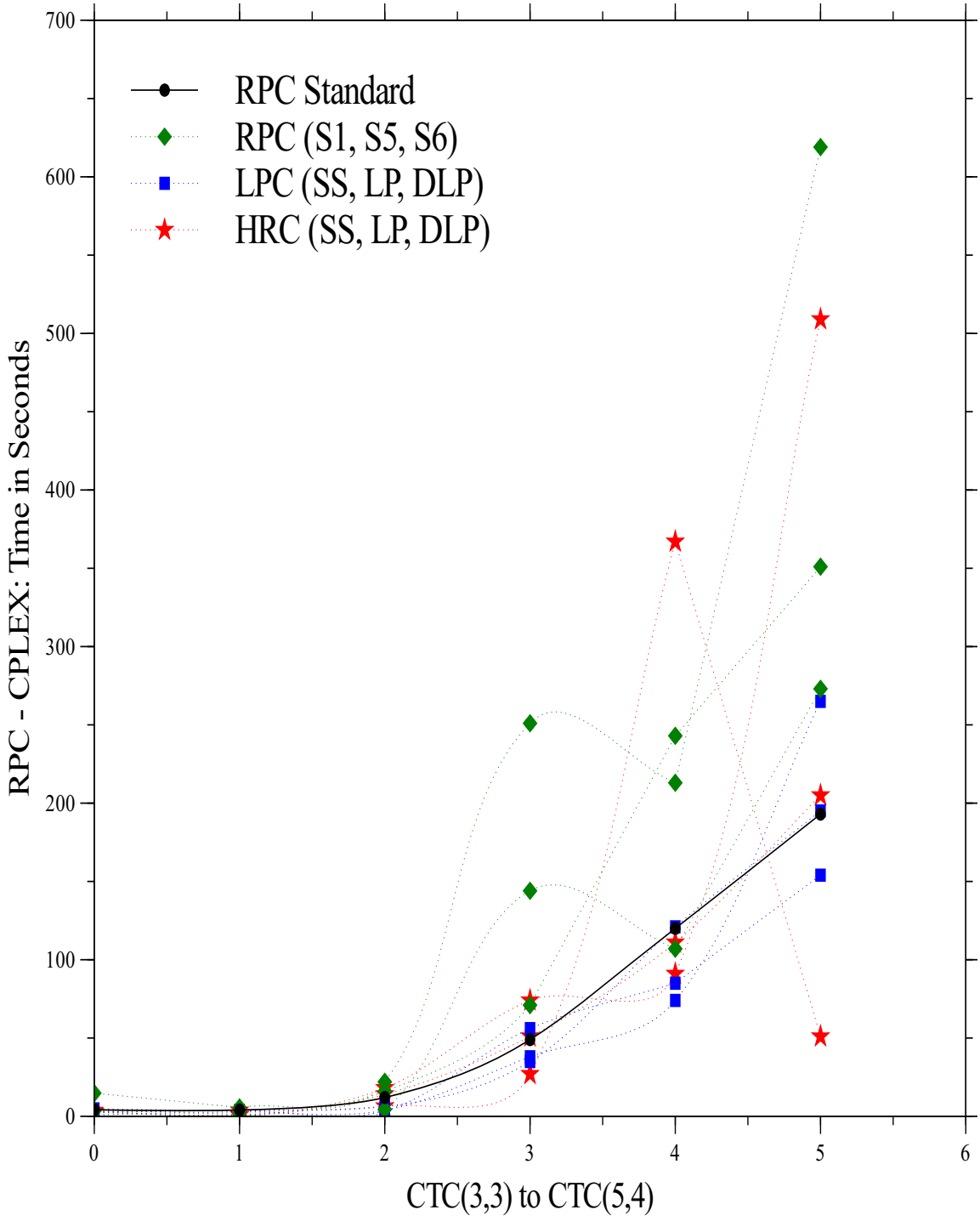



Figure 5.4: RPC with binary restrictions on variables

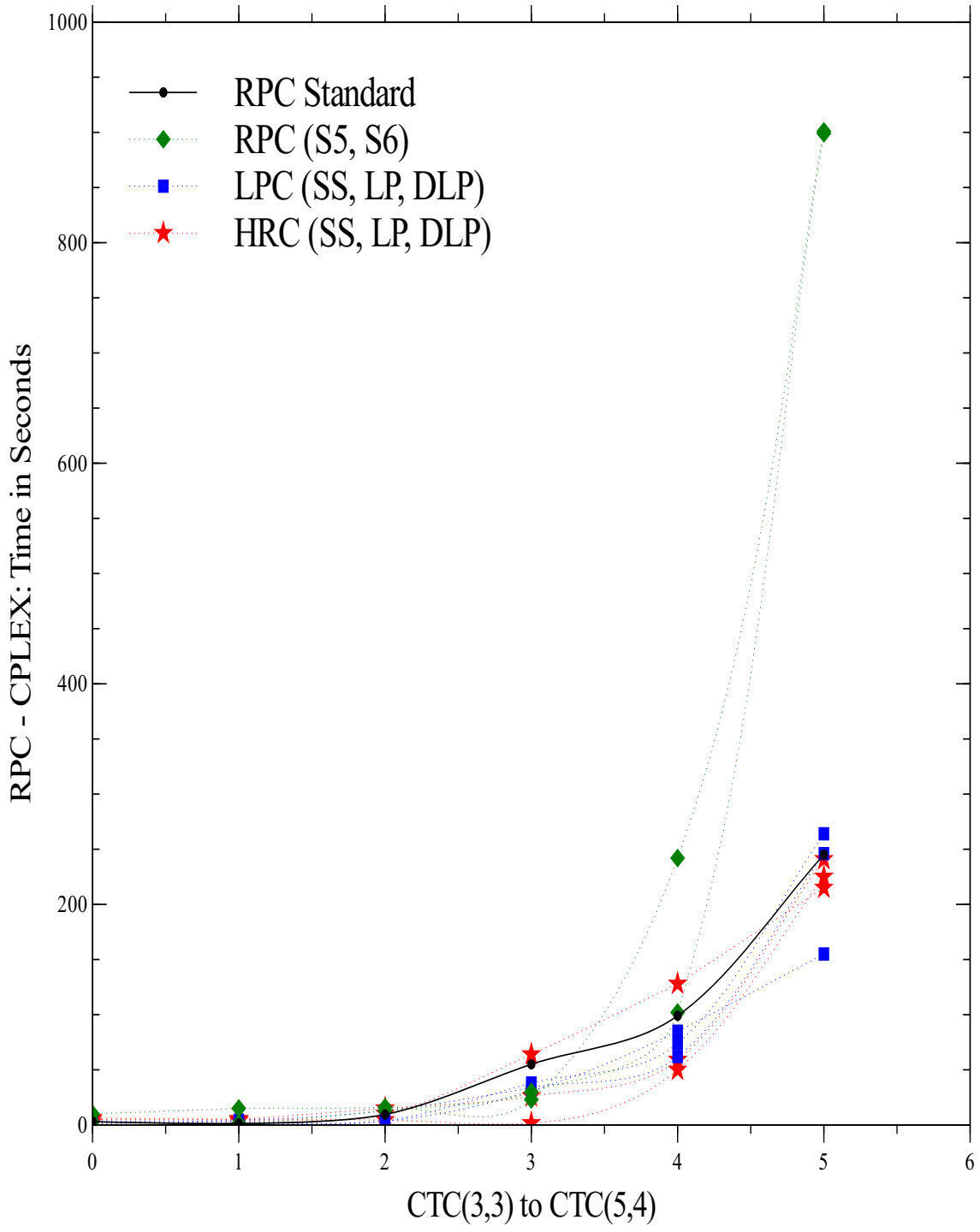



Figure 5.5: RPC with continues restrictions on variables

### 5.6.2 Small Scale AES

As we know AES is one of the important ciphers because it has been adopted by the U.S. government and is now used worldwide. In Section 2.3, we have briefly recalled the arguments of possible configurations of the small scale AES cryptosystem presented in [50]. For more details and a way to express this cipher as a multivariate equation system over $\mathbb{F}_2$ we refer to [50, 106]. Now we present experimental results for polynomial systems coming from this cipher.

Due to the structural properties of small scale AES, the LP and DLP strategies are not effective. Therefore, we are left with the SS and QPS strategies. We choose objective function as the sum over all the initial variables, and *maximization* as optimization direction. In all conversion techniques, if possible, the restrictions on the variables will be kept continues. For the cutting numbers in LPC, we use numbers 3, 4, 5, and 6. Each timing for the LPC, in the following table, is the best of four runs using these four cutting numbers. In Table 5.26, m and n denote the number of equations and variables respectively. All timings are obtained on a computer with a 2.1 GHz AMD Opteron 6172 processor having 48 cores and 64GB RAM. For each timing, we run CPLEX in parallel on two cores. From this table, we can see, the new conversion techniques and strategies usually yield a sizeable speed-up. In extreme cases the gain resulting from our strategies can be striking.

| | m | n | LPC | | HRC | | HIC | RPC | IPC |
|---|---|---|---|---|---|---|---|---|---|
| Strategy | | | SS | QPS | SS | QPS | QPS | SS | SS |
| AES(2,1,1,4) | 288 | 144 | 3 | 3 | 3 | 3 | 3 | 3 | 2 |
| AES(4,1,1,4) | 544 | 272 | 23 | 25 | 13 | 25 | 21 | 10 | 14 |
| AES(6,1,1,4) | 800 | 400 | 67 | 119 | 419 | 170 | 6 | 553 | 44 |
| AES(8,1,1,4) | 1056 | 528 | 1975 | 3908 | 21921 | 298 | 226 | 8351 | 1986 |
| AES(9,1,1,4) | 1184 | 592 | 527 | 26406 | 2493 | 814 | 236 | 2493 | 417 |
| AES(10,1,1,4) | 1312 | 656 | 14298 | 6994 | 9521 | 13211 | 1982 | 9521 | 2655 |
| AES(4,2,1,4) | 1088 | 544 | 7416 | 1377 | 6391 | 62338 | 3147 | 6391 | 789 |
| AES(1,2,2,4) | 576 | 288 | 51 | 42 | 14 | 75 | 20 | 14 | 13 |
| AES(2,2,2,4) | 1024 | 512 | 21735 | 19970 | 19243 | 74982 | 81014 | 19243 | 7830 |
| AES(1,1,1,8) | 640 | 320 | 56815 | 42354 | 2043 | 207180 | 9323 | 10370 | 4684 |

Table 5.26: Effect of different conversions on small scale AES

In the spirit of techniques studied above, it is obviously possible to generate a number of further variations of the conversion algorithms which have the potential to speed up IP solvers. Clearly, the use of IP solvers in polynomial system solving opens up

a wealth of new possibilities. We have realized that there is a strong need to consult literature available on transformation of 0-1 programs into 0-1 linear programs to make the conversion methods more effective and take advantage of the full potential of MILP. Thus the conversion methods deserve further investigation and experimentation. Finally, we invite the reader to solve his favorite systems of polynomial equations with the techniques developed in this chapter.

Chapter 6

# Techniques Using MINLP and Linear Diophantine Equations

In this chapter we address techniques using mixed integer nonlinear programming and linear Diophantine equations. This chapter should mark a first step and offers several future research directions. Research efforts of the past fifty years have led to the development of linear integer programming as a mature discipline of mathematical optimization. Such a level of maturity has not been reached when one considers nonlinear systems subject to integrality requirements for the variables. But nonlinear integer programming is a very active area of research. So it is with this viewpoint that we present techniques coming from mixed integer nonlinear programming for solving systems of polynomial equations over $\mathbb{F}_2$. After formulating the solution of a system of polynomial equations as a mixed integer nonlinear programming problem, we can apply standard nonlinear IP solvers inside our algebraic techniques.

The second approach that we address is to transform a system of polynomial equations over $\mathbb{F}_2$ into a system of linear Diophantine equations. This enables us to use the algorithms for solving systems of linear Diophantine equations for polynomial system solving. After transformation, we apply the straightforward approach for solving systems of linear Diophantine equations for non-negative solutions. Even the straightforward approach seems to provide satisfactory results. We believe that the latest developments (which are mentioned in Section 6.2.1) can perform even better. This gives us an application of solving systems of linear Diophantine equations to cryptanalysis and a motivation for further research.

## 6.1 Techniques Using Mixed Integer Nonlinear Programming

In this section first we review some necessary concepts from the theory of mixed integer nonlinear programming. Then we focus on non-convex *Mixed Integer Nonlinear Programming (MINLP)* problems. Afterwards, we reformulate the polynomial conversion methods presented in Chapter 5 to transfer the problem of solving a system of polynomial equations over $\mathbb{F}_2$ into a MINLP problem. We try to see what can be achieved if we employ a MINLP solver instead of a MILP solver. We present experimental results using the open-source solver COUENNE [56] which solves non-convex MINLP problems. Using some concrete examples, we show that this technique seems to be rather efficient and deserves to be the subject of further investigations.

### 6.1.1 Mixed Integer Nonlinear Programming (MINLP)

In the past decade, nonlinear integer programming has gained huge research activity. Many important real life applications involve MINLP. Traditionally, nonlinear mixed integer programs have been handled in the context of the field of global optimization, where the main focus is on numerical algorithms to solve nonlinear continuous optimization problems and where integrality constraints were considered as an afterthought, using branch-and-bound over the integer variables. This is generally considered a very young field, and most of the problems and methods are not as well-understood or stable as in the case of linear mixed integer programs.

Several important practical problems are most naturally modelled as non-convex MINLP problems. Applications of such problems arise in areas like telecommunications [15, 97, 113], manufacturing and scheduling [5, 170], and epileptic seizure warning [45, 99], and subsume unconstrained 0-1 quadratic programs, 0-1 quadratic knapsack problems, and quadratic assignment problems. We study an application of MINLP to cryptanalysis. Reformulation of the conversion methods in Section 5.1 provides us a non-convex MINLP problem, therefore we are mainly interested in non-convex MINLP problems.

In particular, we focus on approaches for solving polynomial 0-1 programs directly. Such approaches were initially studied in [93, 168]. A computer study by Taha indicates that neither the direct nonlinear nor the transformed linear approach can invariably claim superiority over the other, but that the effectiveness of each depends upon the specific problem to be solved (see [168]). In practice, linear 0-1 codes are typically

easier to program and refine, and can take advantage of a larger store of developed theory than the nonlinear approaches as we saw in Chapter 5. For an overview of the subject, we refer to [107], Chapter 15. The two main types of MINLP problems are convex MINLP problems and non-convex MINLP problems. Let us first recall what we mean by a convex function.

**Definition 6.1.1.** Let $A \subset \mathbb{R}^n$ be a convex set. A function $f : A \longrightarrow \mathbb{R}$ is called **convex** if for all $\theta \in [0,1]$ and for all $x, y \in A$ we have

$$f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta)f(y).$$

The *Second order differentiability condition* says that if $f$ is twice differentiable on a convex domain $A$, then it is convex if and only if the *Hessian matrix* is positive semi-definite ($\nabla^2 f(x) \succeq 0$ for all $x \in A$).

A general model of the MINLP problem can be written as

$$\begin{array}{rl} \text{Minimize} & f(x_1, \ldots, x_n) \\ \text{Subject to} & g_1(x_1, \ldots, x_n) \leq 0 \\ & \vdots \\ & g_m(x_1, \ldots, x_n) \leq 0 \end{array} \tag{6.1}$$

where $n = n_1 + n_2$, $(x_1, \ldots, x_n) \in \mathbb{R}^{n_1} \times \mathbb{Z}^{n_2}$, and $f, g_1, \ldots, g_m : \mathbb{R}^n \longrightarrow \mathbb{R}$ are arbitrary nonlinear functions. However, we are interested in a rather restricted model of nonlinear programming. We focus on instances of polynomial programming where the functions are all quadratic. The specific form of the quadratically constrained mixed integer programming problem that we consider is

$$\begin{array}{rl} \text{Minimize} & f(x_1, \ldots, x_n) \\ \text{Subject to} & g_1(x_1, \ldots, x_n) \leq 0 \\ & \vdots \\ & g_m(x_1, \ldots, x_n) \leq 0 \\ & l_i \leq x_i \leq u_i, \text{ for } i = 1, \ldots, n, \\ & x_i \in \mathbb{R}, \text{ for } i = 1, \ldots, k, \\ & x_i \in \mathbb{Z}, \text{ for } i = k+1, \ldots, n, \end{array} \tag{6.2}$$

where $f, g_1, \ldots, g_m : \mathbb{R}^n \longrightarrow \mathbb{R}$ are quadratic polynomials, $l_i, u_i \in \mathbb{Z}$, and $l_i \leq u_i$. The MINLP problem (6.2) is *convex* if all of the functions $f, g_1, \ldots, g_m$ are convex;

otherwise it is non-convex. Note that when all of $f, g_1, \ldots, g_m$ are linear, we have a MILP problem, for which practical methods have been successfully developed in Chapter 5. We are mainly interested in non-convex MINLP, as this is the demand of the problem we are studying in this chapter. Non-convex MINLP problems are typically much harder to solve than convex ones. Indeed, given a convex MINLP problem, one can compute an initial lower bound simply by solving the continuous relaxation of the problem. This relaxation will be a convex nonlinear problem, which is likely to be relatively easy to solve. The continuous relaxation of a non-convex MINLP problem, on the other hand, is a non-convex nonlinear problem. Non-convex nonlinear problems (sometimes called global optimization problems) are themselves NP-hard (see [80] and references therein). In fact, non-convex MINLP problems are worse than NP-hard because they remind us *Hilbert's Tenth Problem* which states: Given a Diophantine equation $f(x_1, \ldots, x_n) = 0$, determine if it possesses a solution in integers. We know that there cannot be an algorithm for solving Hilbert's Tenth Problem. In other words this problem is not solvable by a Turing machine. If, however, each variable is lower- and upper-bounded explicitly, then non-convex MINLP problems become 'merely' NP-hard (see [104]) such that the search for the optimum can be limited to finite number of possibilities, then certainly the purely enumerative algorithm is available.

Moreover, it is not easy to devise effective heuristics for non-convex MINLP problems. Some exact methods for convex MINLP problems can be converted into heuristics for non-convex MINLP problems. We concentrate primarily on exact approaches. The linearization approach of Chapter 5 has been generalized to a non-convex MINLP problem by A. Billionnet et al. [28, 27] and L. Galli [80]. Finally, Saxena et al. [158], L. Burer and Galli *et al.* [80] have derived strong cutting planes for general non-convex MINLP problems. Saxena et al. do this using disjunctive programming techniques. Burer directly studies the convex hull of feasible solutions directly, using a combination of polyhedral theory and convex analysis. Galli et al. show how to adapt the 'gap inequalities', originally defined for the max-cut problems, to non-convex problems.

The most successful methodology to solve rather general MINLP problems having non-convex relaxations is known as *Spatial Branch-and-Bound* technique, which is also referred to as *Branch-and-Reduce* (see [157] and references therein). This technique has many similarities with the ordinary branch-and-bound technique. It is an elegant exact technique for both global optimization and non-convex MINLP problems. Branch-and-Reduce is the only available approach for the general case. The adaptation of MILP techniques to solve instances of MINLP is a challenging research area. The present

work lies squarely in this area, as we pursue the implementation of a general-purpose algorithm for solving MINLP based on existing software tools for MILP. We will not attempt to make any kind of details. Rather we refer to [107], Chapter 15 and the references therein. Some approaches also involve solving non-convex MINLP problems by converting them to convex MINLP problems.

There are three software packages that can solve non-convex MINLP problems to proven optimality, using branch-and-reduce techniques, namely: BARON, Alpha-BB [81] and COUENNE [56]. Some packages for convex MINLP problem can be used to find heuristic solutions for non-convex MINLP, namely BONMIN, DICOPT and LaGO [81]. Finally, GloptiPoly [81] can solve general polynomial optimization problems. Note that solvers for convex MINLP problems can be used on non-convex problems as heuristics, as they may provide a feasible solution.

For solving MINLP problems, we use the open-source solver COUENNE [56] (Convex Over and Under ENvelopes for Nonlinear Estimation). It is a spatial branch-and-bound algorithm to solve MINLP problems. COUENNE aims at finding global optima of non-convex MINLP problems. It implements linearization, bound reduction, and branching methods within a branch-and-bound framework.

### 6.1.2 Techniques for Polynomial Conversion

We reformulate the polynomial conversion methods presented in Chapter 5 to transfer the problem of solving a system of polynomial equations over $\mathbb{F}_2$ into a MINLP problem. We try to see what can be achieved if we employ a MINLP solver instead of a MILP solver. Furthermore, we investigate the choice of a suitable objective function.

Let $\mathbb{F}_2$ be the finite field with two elements and let $f_1, \ldots, f_m \in P = \mathbb{F}_2[x_1, \ldots, x_n]$ be non-zero polynomials. We are interested in finding $\mathbb{F}_2$-rational solutions of the following system of polynomial equations.

$$\begin{aligned} f_1(x_1, \ldots, x_n) &= 0 \\ &\vdots \\ f_m(x_n, \ldots, x_n) &= 0 \end{aligned}$$

We convert the polynomial equations defined over $\mathbb{F}_2$ into polynomial equations which hold over the reals (respectively over the integers). Usually $f_i$ will be a boolean polynomial (squarefree polynomial), i.e. all terms in the support of $f_i$ will be squarefree, but this is not an essential hypothesis. Let $X = \{X_1, \ldots, X_n\}$ be a set of real vari-

ables (respectively integer variables), i.e. variables over $\mathbb{R}$ (respectively over $\mathbb{Z}$). A conversion method should at least guarantee that a solution for $f_i$ results in a solution for associated polynomial (or polynomials) over $\mathbb{R}$ (respectively over $\mathbb{Z}$). The task of solving the polynomial equation system $f_1 = \ldots = f_m = 0$ can be rephrased as follows: Find a tuple $(a_1, \ldots, a_n) \in \{0,1\}^n$ such that

$$\begin{aligned} F_1(a_1, \ldots, a_n) &\equiv 0 \pmod 2 \\ &\vdots \\ F_m(a_1, \ldots, a_n) &\equiv 0 \pmod 2 \end{aligned} \tag{6.3}$$

where $F_i \in \mathbb{R}[X_1, \ldots, X_n]$ (respectively $F_i \in \mathbb{Z}[X_1, \ldots, X_n]$) is the standard representative (respectively a lifting) of $f_i$. Thus we are looking for an integer solution $(a_1, \ldots, a_n)$ of the system (6.3) which satisfies $0 \leq a_i \leq 1$. This formulation suggests to apply a MINLP algorithm for finding a solution satisfying the stated bounds. For details about the process of conversion, we refer to Section 5.1. Moreover, note that the MINLP problem given by the conversion methods in this section is a non-convex MINLP problem. This can be seen easily by Hessian matrix of the polynomials $F_i$.

In the following, we turn these ideas into effective algorithms. We denote by $\mathbb{T}^n$ the monoid of terms for $\mathbb{F}_2[x_1, \ldots, x_n]$. An element of the monoid of terms $\mathbb{T}^n$ will be denoted by $t$. Our first conversion method is a reformulation of Proposition 5.2.2 which uses Lemma **??** to convert a boolean equation to an equation over the integers. But here we use the fact that we want to use the resulting system of equalities and inequalities in an integer programming problem. This means that we can restrict some (or all) variables to be integers. Consequently while converting a Boolean equation we do not have to ensure that the equation holds over the reals but over the integers.

**Proposition 6.1.2. (Integer Polynomial Conversion (IPC))**
*Let $f_1, \ldots, f_m \in P = \mathbb{F}_2[x_1, \ldots, x_n]$. Then the following instructions define an algorithm which computes a tuple $(a_1, \ldots, a_n) \in \{0,1\}^n$ whose residue class in $\mathbb{F}_2^n$ represent a zero of the zero-dimensional radical ideal $I = \langle f_1, \ldots, f_m, x_1^2 + x_1, \ldots, x_n^2 + x_n \rangle$.*

*1) Reduce $f_1, \ldots, f_m$ modulo the field equations, i.e. make their support squarefree. For $i = 1, \ldots, m$, let $s_i = \#\mathrm{Supp}(f_i)$.*

*2) For $i = 1, \ldots, m$, let $\bar{F}_i$ represent the polynomial $f_i$ in $\mathbb{Z}[x_1, \ldots, x_n]$ such that the coefficients of $\bar{F}_i$ are in $\{0,1\}$.*

*3) For $i = 1, \ldots, m$, introduce a new integer indeterminate $K_i$ and write down the*

*linear inequality* $I_i: \ K_i \leq \lfloor s_i/2 \rfloor$.

*4) For* $i = 1, \ldots, m$, *form the equation* $F_i: \ \bar{F}_i - 2K_i = 0$.

*5) For all* $\alpha \in \{1, \ldots, n\}$, *let* $I'_\alpha: \ X_\alpha \leq 1$.

*6) Choose a linear polynomial* $C \in \mathbb{Q}[X_i, K_i]$ *and use a nonlinear IP solver to find the tuple of natural numbers* $(a_i, b_i)$ *which solves the system of equations and inequalities* $\{I_i, F_i, I_\alpha\}$ *and minimizes (or maximizes)* $C$.

*7) Return* $(a_1, \ldots, a_n)$ *and stop.*

*Proof.* Since we are looking for natural numbers $a_i$ for which $I_\alpha$ holds, we have $a_i \in \{0, 1\}$. Next if follows from $F_i$ that $f_i(a_1, \ldots, a_n) = 2K_i$ is an even number, and $I_i$ is nothing but the trivial bound for $K_i$ implied by the size of the support of $f_i$. In this way the solution of the IP problem corresponds uniquely to the tuple $(a_1, \ldots, a_n) \in \{0, 1\}^n$ which satisfies the above reformulation of the given polynomial system. □

**Remark 6.1.3.** Note that the algorithm in Proposition 6.1.2 can be extended to a prime finite field $\mathbb{F}_p$ in the straightforward way as follows. Let $p$ be a prime number and let $\mathbb{F}_p$ be the finite field with $p$ elements. Let $f_1, \ldots, f_m \in P = \mathbb{F}_p[x_1, \ldots, x_n]$ be a set of polynomials. Then the following instructions define an algorithm which computes a tuple $(a_1, \ldots, a_n) \in \mathbb{F}_p^n$ whose residue class in $\mathbb{F}_2^n$ represent a zero of the zero-dimensional radical ideal $I = \langle f_1, \ldots, f_m, x_1^p - x_1, \ldots, x_n^p - x_n \rangle$.

1) Reduce $f_1, \ldots, f_m$ modulo the field equations.

2) For $i = 1, \ldots, m$, let $\bar{F}_i$ represent the polynomial $f_i$ in $\mathbb{Z}[x_1, \ldots, x_n]$ such that the coefficients of $\bar{F}_i$ are in $\mathbb{F}_p$.

3) For $i = 1, \ldots, m$, let $s_i = eval(\bar{F}_i, (b_1, \ldots, b_n))$, where $b_j = p$ for all $j \in \{1, \ldots, n\}$.

4) For $i = 1, \ldots, m$, introduce a new integer indeterminate $K_i$ and write down the linear inequality $I_i: \ K_i \leq \lfloor s_i/p \rfloor$.

5) Form the equation $F_i: \ \bar{F}_i - pK_i = 0$.

6) For all $\alpha \in \{1, \ldots, n\}$, let $I'_\alpha: \ X_\alpha \leq p$.

7) Choose a linear polynomial $C \in \mathbb{Q}[X_i, K_i]$ and use an IP solver to find the tuple of natural numbers $(a_i, c_i)$ which solves the system of equations and inequalities $\{I_i, F_i, I'_\alpha\}$ and minimizes $C$.

8) Return $(a_1, \ldots, a_n)$ and stop.

The proof for the algorithm given by the above sequence of instructions is an analog of the proof of Proposition 6.1.2.

**Remark 6.1.4.** Assume that we are in the setting of the algorithm in Proposition 6.1.2. If we can find a feasible binary/integer-valued solution for the MINLP for an arbitrary objective function, this solution can be converted into a solution for the original system. Furthermore, note that the initial state variables $X_1, \ldots, X_n$ will be forced to take on binary values. The variables $K_1, \ldots, K_m$ will be forced to take on integer values in the interval $[0, \lfloor s_i/2 \rfloor]$.

To understand Proposition 6.1.2 better, we now apply it in a concrete case.

**Example 6.1.5.** Over the field $K = \mathbb{F}_2$, consider $f_1, f_2, f_3 \in K[x_1, x_2, x_3]$, where $f_1 = x_1x_2 + x_1x_3 + 1$, $f_2 = x_1x_3 + x_2x_3 + x_1 + x_3 + 1$, and $f_3 = x_1x_2 + x_1x_3 + x_2 + 1$. Let us follow the steps of the algorithm in Proposition 6.1.2.

1) Let $s_1 = 3$, $s_2 = 5$, and $s_3 = 4$.

2) Let $\bar{F}_1 = X_1X_2 + X_1X_3 + 1$, $\bar{F}_2 = X_1X_3 + X_2X_3 + X_1 + X_3 + 1$, and $\bar{F}_3 = X_1X_2 + X_1X_3 + X_2 + 1$.

3) Introduce new integer indeterminates $K_1, K_2, K_3$ and write down the linear inequalities $I_1 : K_1 \leq 1$, $I_2 : K_2 \leq 2$ and $I_3 : K_3 \leq 2$.

4) Form the following equations.

$$\begin{aligned} F_1 &: X_1X_2 + X_1X_3 + 1 - 2K_1 = 0 \\ F_2 &: X_1X_3 + X_2X_3 + X_1 + X_3 + 1 - 2K_2 = 0 \\ F_3 &: X_1X_2 + X_1X_3 + X_2 + 1 - 2K_3 = 0 \end{aligned}$$

5) Let $I_1' : X_1 \leq 1$, $I_2' : X_2 \leq 1$ and $I_3' : X_3 \leq 1$.

6) Let $C = X_1 + X_2 + X_3$. Now use an IP solver to minimize $C$ subject to $\{I_1, \ldots, I_3, F_1, F_2, F_3, I_1', I_2', I_3'\}$.

7) Choose values for $X_1$, $X_2$ and $X_3$ from the solution provided by an IP solver. This will return $(1, 0, 1)$.

**Remark 6.1.6.** As claimed above the polynomials $F_i$ given by the conversion algorithm in Proposition 6.1.2 are actually non-convex polynomial functions. To see this consider the polynomial $F_1$ in step 3) of Example 6.1.5. It can be easily seen that the Hessian matrix of the polynomial $F_1$ is not positive semi-definite. This shows that $F_1$ is a non-convex polynomial function.

Our second conversion method is a reformulation of Proposition 5.2.8 which uses the standard representation (see Definition 5.2.5) to convert a boolean equation to an equation (or equations) over the reals. Real that the standard representation results in increasing degree and increasing number of terms over the real domain. To control increasing degree and increasing number of variables we introduce new variables while conversion process. Furthermore, while converting a Boolean equation we have to ensure that the equation holds over the reals. The only requirement we have is that the solution of the system over $\mathbb{F}_2$ is also a solution of the real system. The additional non-binary solutions of the real system can be ignored.

**Proposition 6.1.7. (Real Polynomial Conversion (RPC))**
*Let $f_1, \ldots, f_m \in P = \mathbb{F}_2[x_1, \ldots, x_n]$. Then the following instructions define an algorithm which computes a tuple $(a_1, \ldots, a_n) \in \{0,1\}^n$ whose residue class in $\mathbb{F}_2^n$ represent a zero of the zero-dimensional radical ideal $I = \langle f_1, \ldots, f_m, x_1^2 + x_1, \ldots, x_n^2 + x_n \rangle$.*

*1) Reduce $f_1, \ldots, f_m$ modulo the field equations, i.e. make their support squarefree. For $i = 1, \ldots, m$, let $S_i$ be the set of terms of degree $\geq 2$ in $f_i$. Let $S = \bigcup_{i=1}^m S_i$ and $s = |S|$.*

*2) For every $t_j \in S$, introduce a new indeterminate $x_{n+j}$ and form the equation $f'_{m+j} : x_{n+j} = t_j$. For $i = 1, \ldots, m$, write $f_i = \sum_j t_j + \ell_i$ where the sum extends over all $j$ such that $t_j \in S_i$ and where $\ell_i \in P_{\leq 1}$. Form the equation $f'_i : \sum_j x_{n+j} + \ell_i = 0$.*

*3) For $i = 1, \ldots, m+s$, let $F_i$ be the equation which is the standard representation of $f'_i$.*

*4) For all $\alpha \in \{1, \ldots, n\}$, let $I_\alpha : X_\alpha \leq 1$.*

*5) Choose a linear polynomial $C \in \mathbb{Q}[X_\alpha, X_{n+j}]$ and use an IP solver to find the tuple of natural numbers $(a_\alpha, a_{n+j})$ which solves the system of equations and inequalities $\{F_i, I_\alpha\}$ and minimizes (or maximizes) $C$.*

*6) Return $(a_1, \ldots, a_n)$ and stop.*

*Proof.* We are looking for natural numbers $a_\alpha$ for which $I_\alpha$ holds, therefore we have $a_\alpha \in \{0, 1\}$. Similarly, we have $a_{n+j} \in \{0, 1\}$ by $I_\alpha$ and $F_{m+j}$. Next it follows from the standard representation that $F_i \in \{0, 1\}$. In this way the solution of the IP problem corresponds uniquely to the tuple $(a_1, \ldots, a_n) \in \{0, 1\}^n$ which satisfy the above reformulation of the given polynomial system. □

Assume that we are in the setting of the algorithm in Proposition 6.1.7. Note that if $\max\{\deg(f_i) \mid i \in \{1, \ldots, m\}\} \leq 2$ and for $i = 1, \ldots, m$, the maximum number of terms in the support of $f_i$ does not exceed 4, the algorithm works with quadratic polynomials in all of its iterations. Furthermore, note that all variables in the algorithm take on binary values. To understand Proposition 6.1.7 better, we now apply it in a concrete case.

**Example 6.1.8.** Over the field $K = \mathbb{F}_2$, consider $f_1, f_2, f_3 \in K[x_1, x_2, x_3]$, where $f_1 = x_1x_2 + x_1x_3 + 1$, $f_2 = x_1x_3 + x_2x_3 + x_1$, and $f_3 = x_1x_2 + x_1x_3 + x_2 + 1$. Let us follow the steps of the algorithm in Proposition 6.1.7.

1) Let $S_1 = \{x_1x_2, x_1x_3\}$, $S_2 = \{x_1x_3, x_2x_3\}$, and $S_3 = \{x_1x_2, x_1x_3\}$. Let $S = \{x_1x_2, x_1x_3, x_2x_3\}$ and $s = 3$.

2) Introduce new indeterminates $x_1, x_2, x_3$. Form the equations $f_4' : \ x_4 = x_1x_2$, $f_5' : \ x_5 = x_1x_3$ and $f_6' : \ x_6 = x_2x_3$. Form the equations $f_1' : \ x_4 = x_5 + 1$, $f_2' : \ x_5 = x_6 + x_1$ and $f_3' : \ x_4 + x_5 = x_2 + 1$.

3) The standard representations of the equations $f_1', \ldots, f_6'$ are:

$$\begin{aligned} &F_1 \ : \ X_4 + X_5 - 1 = 0, \quad F_2 \ : \ X_5 - X_6 - X_1 + 2X_1X_6 = 0, \\ &F_3 \ : \ X_4 + X_5 - 2X_4X_5 + X_2 - 1 = 0, \quad F_4 \ : \ X_4 - X_1X_2 = 0, \\ &F_5 \ : \ X_5 - X_1X_3 = 0, \quad F_6 \ : \ X_6 - X_2X_3 = 0. \end{aligned}$$

4) Let $I_1 : \ X_1 \leq 1$, $I_2 : \ X_2 \leq 1$ and $I_3 : \ X_3 \leq 1$.

5) Let $C = X_1 + X_2 + X_3$. Now use an IP solver to minimize $C$ subject to $\{F_1, \ldots, F_6, I_1, I_2, I_3\}$.

6) Choose values for $X_1$, $X_2$ and $X_3$ from the solution provided by an IP solver. This will return $(1, 0, 1)$.

### 6.1.3 Experimental Results

Now we present our observations and results from experiments with the algorithms in Propositions 6.1.2 and 6.1.7. We run COUENNE on a laptop with a 2.13 GHz Intel Pentium P6200 Dual Core processor and 4GB RAM, but we are using only a single thread out of two. See Section 5.1 for some abbreviations we have used in the following. Furthermore, the timings for the conversion algorithms were ignored, since they were not implemented efficiently and should be seen as a preprocessing step. Finally, we note that all timings can be reproduced i.e, if we run a single experiment on a particular machine $n$ times, every time we will get the same timing.

#### First Experiment

Assume that we choose the objective function as the sum over all the initial variables $X_1, \ldots, X_n$ and *maximization* as optimization direction. Then Table 6.1 gives the timings for the conversion algorithm in Proposition 6.1.2.

| Sbox | CTC(2,2) | CTC(3,3) | CTC(3,4) | CTC(4,3) | CTC(4,4) |
|---|---|---|---|---|---|
| Equations | 98 | 216 | 285 | 288 | 380 |
| Variables | 54 | 117 | 153 | 156 | 204 |
| F-Sbox | 2 | 25 | 213 | 166 | 1465 |
| H-Sbox | 1 | 15 | 250 | 198 | 2763 |
| M-Sbox | 1 | 37 | 666 | 526 | >4000 |

Table 6.1: COUENNE time comparison for Sbox using IPC

Moreover, we remark that we run a number of experiments which show that optimization direction seems to have no effect on the timings.

#### Second Experiment

As we saw in Chapter 5 the objective function strongly affects the running time of an IP solver. To see the role of the objective function in MINLP models, natural choices are as follows.

**O1:** The sum over all the initial variables $X_1, \ldots, X_n$.

**O2:** The sum over all variables.

**O3:** The sum over all new variables $K_1, \ldots, K_n$ introduced by the conversion algorithm.

| | H-SBox | | | F-SBox | | |
|---|---|---|---|---|---|---|
| System | O1 | O2 | O3 | O1 | O2 | O3 |
| CTC(2,2) | 1 | 1 | 1.7 | 2 | 2.4 | 1.7 |
| CTC(3,3) | 15 | 19 | 38 | 25 | 45 | 39 |
| CTC(3,4) | 250 | 144 | 270 | 213 | 217 | 269 |
| CTC(4,3) | 198 | 209 | 229 | 166 | 292 | 230 |
| CTC(4,4) | 2763 | 2151 | 2020 | 1465 | 2088 | 2020 |

Table 6.2: COUENNE time comparison for objective function using IPC

**Third Experiment**

The Techniques we are studying are used to solve sparse systems of polynomial equations. Most of the polynomial systems that we have considered in all experiments are defined over the field $\mathbb{F}_2$. To give you an overview how such techniques perform over $\mathbb{F}_p$, we consider some instances of polynomial systems arising the HFE cryptosystem which are relatively dense. We consider the polynomial systems defined over $\mathbb{F}_2$, $\mathbb{F}_3$, $\mathbb{F}_5$, and $\mathbb{F}_7$. We use the algorithm in Corollary 6.1.3 for polynomial conversion.

| Base Field | HFE(6) | HFE(7) | HFE(8) | HFE(9) |
|---|---|---|---|---|
| $\mathbb{F}_2$ | 0.1 | 0.3 | 0.9 | 0.4 |
| $\mathbb{F}_3$ | 1.4 | 1.8 | 14 | 58 |
| $\mathbb{F}_5$ | 1 | 238 | 1297 | 2916 |
| $\mathbb{F}_7$ | 276 | 680 | 765 | >19000 |

Table 6.3: IPC time comparison for different base fields

**Fourth Experiment**

Now assume that we are in the setting of the conversion algorithm in Proposition 6.1.7. We choose the objective function as the sum over all the initial variables $X_1, \ldots, X_n$ and *maximization* as optimization direction, and force all variables (initial, auxiliary, and introduced by the conversion algorithm) to take on binary values. This conversion algorithm seems to solve only small instances of systems of polynomial equations as given by Table 6.4.

| System | CTC(2,2) | CTC(3,3) | CTC(3,4) | CTC(4,3) | CTC(4,4) |
|---|---|---|---|---|---|
| Time | 3 | 182 | >4000 | >4000 | >4000 |

Table 6.4: COUENNE time comparison using RPC

As one can see from Tables 6.1 and 6.2, for very sparse systems the running time of the nonlinear IP technique compares favorably to the running times of the Gröbner basis techniques in Chapter 3. Even for examples involving many indeterminates, such as CTC(4,4), the above timings compete with individually tailored Gröbner basis methods, such as the ones reported in [3]. In the spirit of the techniques studied above, it is obviously possible to generate a number of further variations of the conversion algorithms which have the potential to speed up IP solvers. Thus the conversion methods deserve further investigation and experimentation. Furthermore, it could be interesting to use other software packages that can solve non-convex MINLP problems such as BARON and Alpha-BB [81].

By studying special cases of MINLP problems, one can do better. Our problem i.e. the problem discussed in experimental results is a special case that provides a lot of information about the model and the structure of the model. We believe that the development of techniques for this special case could be a promising direction for future research. Furthermore, there is still a clear need for improved theory, algorithms, and software for MINLP. Any improvement will also be beneficial for our special case.

The study of algorithms for solving MINLP problems is a very young and a fast developing area, we believe that the conversion methods presented in this section can take full advantage of any new development. Furthermore, note that in the modern computation world, a good IP solver is supposed to have parallelization capabilities. Thus we can think of getting benefit from parallelization capabilities of IP solvers. Therefore, we believe that if effective conversion methods are developed they can be able to solve sparse systems of polynomial equations involving a large number of indeterminates. Finally, we invite the reader to solve his favorite systems of polynomial equations using the techniques developed in this section.

## 6.2 Techniques Using Linear Diophantine Equations

First we review some necessary concepts from number theory. In particular, we focus on methods for solving systems of linear Diophantine equations. Afterwards, we reformulate the polynomial conversion methods presented in Chapter 5 to transfer a system of polynomial equations over $\mathbb{F}_2$ into a system of linear Diophantine equations. This enables us to to use the algorithms for solving systems of linear Diophantine equations for polynomial system solving. After reformulation i.e. after applying a conversion algorithm we apply the straightforward approach for solving systems of linear Diophantine equations for finding non-negative integer solutions. Even the straightforward approach seems to provide satisfactory results. Furthermore, we highlight some ideas to spark further research in this direction. We believe that the highlighted techniques can perform even better. Towards the end we show the performance of these techniques using some concrete examples.

### 6.2.1 Solving Systems of Linear Diophantine Equations

Let us recall that a *Diophantine equation* is an equation of the form $f(x_1, \ldots, x_n) = 0$, where $f$ is a polynomial with integer coefficients and the indeterminates $x_1, \ldots, x_n$ take integer values. If the polynomial $f$ has degree one then $f(x_1, \ldots, x_n) = 0$ is called a *linear Diophantine equation*. Solving systems of linear Diophantine equations (in particular for non-negative solutions) is of both of theoretical and practical importance. In particular, methods of solving such systems are used in systems of artificial intelligence and logic programming [132], computer algebra [35], automation of theorem proving with unification [16], in parallelizing programs, in Petri nets [6], etc. Furthermore, we present an application to cryptanalysis.

We are interested in algorithms which solve linear systems of Diophantine equations for non-negative solutions. A well-known approach to find non-negative integer solutions consists of two steps. First find all integer solutions and then obtain the non-negative integer solutions from the integer solutions. In other works, first step is to find a general integer solution and second step is to find a non-negative (particular) integer solution from general integer solution. We address both of these steps one by one in the following.

Let $\mathcal{A} = (a_{ij}) \in \mathrm{Mat}_{m,n}(\mathbb{Z})$ be a matrix having $m$ rows, $n$ columns, and integer entries. Furthermore, let $(b_1, \ldots, b_m) \in \mathbb{Z}^m$ be a vector having $m$ integer entries, and let $x_1, \ldots, x_n$ be indeterminates. Our goal in this section is to study the set of non-

negative integer solutions of the following system of linear Diophantine equalities and inequalities.

$$\begin{array}{rcl}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n & \leq & b_1 \\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n & \leq & b_2 \\
\vdots & \vdots & \vdots \\
a_{r1}x_1 + a_{r2}x_2 + \cdots + a_{rn}x_n & \leq & b_r \\
a_{(r+1)1}x_1 + a_{(r+1)2}x_2 + \cdots + a_{(r+1)n}x_n & = & b_{r+1} \\
\vdots & \vdots & \vdots \\
a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n & = & b_m
\end{array} \tag{6.4}$$

We can convert the above system of equalities and inequalities into a system of equations in the following way. We introduce new indeterminates $x_{n+1}, \ldots, x_{n+r}$ and consider the associated system of Diophantine equations.

$$\begin{array}{rcl}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n + x_{n+1} & = & b_1 \\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n + x_{n+2} & = & b_2 \\
\vdots & \vdots & \vdots \\
a_{r1}x_1 + a_{r2}x_2 + \cdots + a_{rn}x_n + x_{n+r} & = & b_r \\
a_{(r+1)1}x_1 + a_{(r+1)2}x_2 + \cdots + a_{(r+1)n}x_n & = & b_{r+1} \\
\vdots & \vdots & \vdots \\
a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n & = & b_m
\end{array} \tag{6.5}$$

As we have seen above the first step is to obtain a general integer solution which can be resolved by the *Hermite normal form*, the *Smith normal form*, by computing the *Kernel* or by the *LLL-Algorithm*. We use the Smith normal form to resolve the first step. The Smith normal form is a canonical diagonal form for equivalence of matrices over a principal ideal ring as given by the following proposition.

**Proposition 6.2.1.** *If $A$ is any integer matrix, there exist invertible integer matrices $P$ and $Q$, whose inverses are also integer matrices, such that*

$$PAQ = D,$$

*where $D$ is a diagonal matrix of integers with the property that*

$$D = diag\{d_{11}, \ldots, d_{rr}, 0, \ldots, 0\}$$

*with $d_{ii}$ a factor of $d_{jj}$ for $i < j$ and for which $d_{ii} \neq 0$ for $i \leq r$. The matrix $D$ is called the* ***Smith normal form*** *of the matrix $A$ and the matrices $P$ and $Q$ are called* ***transforming matrices****.*

*Proof.* See [89], Theorem 1. □

The Smith normal form was first proven to exist by Smith [164] for matrices over the integers. Once we have the Smith normal form the general integer solution can be easily obtained from it. Now there is a natural question. Is the Smith normal form the best way to solve the first step? We do not have a final answer but we choose the Smith normal form due to the following reasons.

There are polynomial time algorithms for computing the Smith normal form. The classical Smith normal form algorithms perform an elimination process with some gcd computations over the integers or modulo large primes. The first classical polynomial time algorithm for computing Smith normal forms over $\mathbb{Z}$ was given by Kannan and Bachem [109], and later improved by Chou and Collins [49]. Few years later an algorithm was given in [100] that performs all arithmetic modulo the determinant of a square nonsingular input matrix. The modular approach, which effectively controls intermediate swell, was extended to singular input matrices in [100, 90]. A further improvement of this approach is given in [167]. Last but not least, for sparse matrices one should expect to accelerate the solution by exploiting the sparsity. A theoretical study [83] shows that methods for sparse integer matrices perform substantially better than the classical methods. A most recent attempt to design a probabilistic algorithm for computing Smith normal forms using efficient sparse integer matrix computations is given in [71]. The systems of linear Diophantine equations arising from Subsections 6.2.2 and 6.2.3 are sparse and structured. Thus we can get full advantage of the recently developed probabilistic algorithms which exploit sparsity. Thus these algorithms deserve further investigation and experimentation. Therefore, we believe that if such algorithms are developed they can be able to solve sparse systems of linear Diophantine equations involving a large number of indeterminates in little time.

Now we come to the second step, i.e. how to obtain a non-negative integer solution from the general integer solution. The indeterminates $x_{n+1}, \ldots, x_{n+r}$ are usually known in the literature as *slack variables* (see [159]). It is trivial to see that that the element $(s_1, \ldots, s_n) \in \mathbb{N}^n$ is a solution of (6.4) if and only if there exist $s_{n+1}, \ldots, s_{n+r} \in \mathbb{N}$ such that $(s_1, \ldots, s_n, s_{n+1}, \ldots, s_{n+r})$ is solution of (6.5) (see for instance [156], Lemma 1). In view of this result, we shall from now on assume that our original system is in fact

a system of linear Diophantine equations, i.e. that we want to find the non-negative integer solutions $(s_1, \ldots, s_n) \in \mathbb{N}^n$ of the following system.

$$\begin{array}{rcl} a_{11}x_1 + a_{12}x_2 + \ \ldots \ + a_{1n}x_n & = & b_1 \\ a_{21}x_1 + a_{22}x_2 + \ \ldots \ + a_{2n}x_n & = & b_2 \\ \vdots \qquad\qquad\qquad\qquad\quad \vdots & \vdots & \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n & = & b_m \end{array} \tag{6.6}$$

Unfortunately, restricting the domain to the natural numbers makes the problem much more difficult. Actually, this is an NP-hard problem. We know that $\mathbb{N}$ is a monoid. We are looking for the set of solutions $\mathcal{S}$ of the system (6.6) with componentwise non-negative coordinates i.e. solutions in $\mathbb{N}^n$. One way of finding the set of non-negative integer solutions $\mathcal{S}$ of the system (6.6) is to solve the following homogenous system of Diophantine equations for non-negative integer solutions.

$$\begin{array}{rcl} a_{11}x_1 + a_{12}x_2 + \ \ldots \ + a_{1n}x_n - b_1x_{n+1} & = & 0 \\ a_{21}x_1 + a_{22}x_2 + \ \ldots \ + a_{2n}x_n - b_2x_{n+1} & = & 0 \\ \vdots \qquad\qquad\qquad\qquad\qquad\quad \vdots & \vdots & \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n - b_mx_{n+1} & = & 0 \end{array} \tag{6.7}$$

The set of non-negative integer solutions $\mathcal{S}'$ of the system (6.7) is a submonoid of $\mathbb{N}^{n+1}$. Note that there is a partial order (componentwise divisibility) on $\mathcal{S}'$ in a natural way. Since there is a partial ordering on $\mathcal{S}'$, we can speak about the minimal elements in $\mathcal{S}'$. In fact the submonoid $\mathcal{S}'$ is generated by its set of non-zero minimal elements $\mathcal{M}'$ with respect to the natural partial order. The set $\mathcal{M}'$ is also known as the basis of $\mathcal{S}'$ in the literature (see for instance [156], Lemma 2).

Note that the set of non-negative integer solutions $\mathcal{S}$ of the system (6.6) has a minimal set of generators $\mathcal{M}$ with respect to natural partial order which provides a finite and complete representation of the set $\mathcal{S}$: any solution in $\mathcal{S}$ is an $\mathbb{N}$-linear combination of the elements in $\mathcal{M}$.

As explained above, the second step is to get non-negative integer solutions from the general integer solution. The second step is equivalent to finding the feasible region for an integer programming problem. Now there are two natural question. Now we have a natural question. Why is integer programming sufficient enough to solve the secondary problem?

A first answer to the second question is the following. We are interested in finding

only one non-negative integer solution of a given system of linear Diophantine equations. Our system of linear Diophantine equations has only a few minimal non-negative integer solutions. In particular, it has a unique minimal non-negative solution. Several efficient algorithms are available in the field of discrete optimization for solving integer programs. IP solvers are very fast in practice and they can handle large scale integer programming models. Furthermore, note that some IP solvers like CPLEX can be parallelized. Thus we can benefit from parallelization capabilities of IP solvers to solve systems of polynomial equations. Thus in our opinion integer programming is the right tool for this purpose. We support this argument with the help of examples in Section 6.2.3. Furthermore, research efforts of the past fifty years have led to a development of linear integer programming as a mature discipline of mathematical optimization. Several IP solvers have been developed. Note that some IP solvers like CPLEX can be parallelized. Thus we can benefit from the parallelization capabilities of the IP solvers to solve systems of linear Diophantine equations.

### 6.2.2 Techniques for Polynomial Conversion

As usual let $\mathbb{F}_2$ be the finite field with two elements and let $f_1, \ldots, f_m \in P = \mathbb{F}_2[x_1, \ldots, x_n]$ be non-zero polynomials. We are interested in finding $\mathbb{F}_2$-rational solutions of the following system of polynomial equations.

$$\begin{aligned} f_1(x_1, \ldots, x_n) &= 0 \\ &\vdots \\ f_m(x_n, \ldots, x_n) &= 0 \end{aligned}$$

Recall that in Chapter 5 we studied ways of transferring the problem of solving a system of polynomial equations over $\mathbb{F}_2$ into a system of linear equalities and inequalities. In particular, the Integer Polynomial Conversion (IPC) (see Section 5.5) gives us a system of linear Diophantine equalities and inequalities. This formulation suggests to apply a linear Diophantine system solving algorithm for finding a solution satisfying the system of linear Diophantine inequalities and equalities. For details about the process of conversion we refer to Section 5.1.

Recall that in Section 5.5 we have developed several strategies that can be used in the settings of a conversion algorithm. In particular, we saw the standard strategy (SS), the linear partner strategy (LP), the double linear partner strategy (DLP) and quadratic partner strategy (QP). In Section 5.5 we have used these strategies to model a

system of linear Diophantine equalities and inequalities. Now we move one step further and obtain a system of linear Diophantine equations by introducing slack variables. The following proposition which uses the full potential of Sections 5.4 and 5.5 turns this idea into an effective algorithm. The notation and terminology are as defined in Sections 5.4 and 5.5.

**Proposition 6.2.2. (Hybrid Integer Conversion (HIC))**
*Let $f_1, \ldots, f_m \in P = \mathbb{F}_2[x_1, \ldots, x_n]$. Then the following instructions define an algorithm which computes a tuple $(a_1, \ldots, a_n) \in \{0,1\}^n$ whose residue class in $\mathbb{F}_2^n$ represent a zero of the zero-dimensional radical ideal $I = \langle f_1, \ldots, f_m, x_1^2 + x_1, \ldots, x_n^2 + x_n \rangle$ of $P$.*

*1) Reduce $f_1, \ldots, f_m$ modulo the field equations, i.e. make their support squarefree. Let $G = \emptyset$.*

*2) Repeat the following step 3) until no polynomial $g$ can be found anymore.*

*3) Find a subset of $\mathrm{Supp}(f_i)$ which defines a polynomial $g$ of the type required by the chosen conversion strategy. Introduce a new indeterminate $x_{n+j}$, replace $f_i$ by $f_i - g + x_{n+j}$, and append $g + x_{n+j}$ to $G$.*

*4) For each polynomial in $G$, compute a logical representation in CNF and form the set of all clauses $C$ of all these logical representations.*

*5) For each clause $c \in C$ form a clausal inequality $I_c$.*

*6) For $i = 1, \ldots, m$, let $S_i$ be the set of new indeterminates $x_{n+j}$ in $f_i$, and let $s_i = \#\mathrm{Supp}(f_i)$.*

*7) For $i = 1, \ldots, m$, introduce a new integer indeterminate $K_i$ and write down the linear inequality $I_i : \; K_i \leq \lfloor s_i/2 \rfloor$.*

*8) For $i = 1, \ldots, m$, write $f_i = \sum_j x_{n+j} + \ell_i$ where the sum extends over all $j$ such that $x_{n+j} \in S_i$ and where $\ell_i \in P_{\leq 1}$. Form the equation $F_i : \; \sum_j X_{n+j} + L_i - 2K_i = 0$, where $L_i \in \mathbb{Z}[X_1, \ldots, X_n]_{\leq 1}$ is a lifting of $\ell_i$.*

*9) Convert the inequalities $I_i, I_c$ respectively into the equations $E_i, E_c$ by introducing new indeterminates (slack variables) $Z_k$.*

*10) Solve the system of linear Diophantine equations $\{E_i, F_i, E_c\}$ over $\mathbb{Z}$ for a general integer solution $(A_\alpha, A_{n+j}, C_i, D_k)$, where $A_\alpha, A_{n+j}, C_i, D_k$ are linear Diophantine polynomials in free indeterminates.*

*11) For all $\alpha \in \{1, \ldots, n\}$, let $X_\alpha \leq 1$. For each $j$, let $X_{n+j} \leq 1$.*

*12) Consider the bounds given by steps 7) and 11) for the general integer solution $(A_\alpha, A_{n+j}, C_i, D_k)$ and formulate an IP problem. Use an IP solver to find a minimal non-negative integer solution $(a_\alpha, a_{n+j}, c_i, d_k)$ which solves the system of linear Diophantine equations $\{E_i, F_i, E_c\}$.*

*13) Return $(a_1, \ldots, a_n)$ and stop.*

*Proof.* It is clear that steps 2)−3) linearize the polynomials $f_i$ by introducing new indeterminates $x_{n+j}$. Moreover, step 4) is based on Lemma 5.4.2, Lemma 5.4.3, Proposition 5.4.5, or Remark 5.4.5 for the polynomials $g + x_{n+j}$ from step 3), and step 7) follows from Lemma 5.4.6. In step 6) the polynomials $f_i$ are linear polynomials in the indeterminates $x_\alpha$ and $x_{n+j}$. Next it follows from $F_i$ that $F_i(a_1, \ldots, a_n) = 2K_i$ is an even number, and $I_i$ is nothing but the trivial bound for $K_i$ implied by the size of the support of $f_i$.

Step 10) gives a general solution of the system of linear Diophantine equations $\{E_i, F_i, E_c\}$. Step 12) finds a minimal non-negative integer solution by considering the feasible region for the integer programming problem given by the general solution and the bounds of steps 7) and 11). In this way the minimal solutions of the system of linear Diophantine equations correspond uniquely to the tuples $(a_1, \ldots, a_n) \in \{0, 1\}^n$ which satisfy the given polynomial system. The claim follows easily from these observations. □

To understand Proposition 6.2.2 better, we now apply it in a concrete case.

**Example 6.2.3.** Over the field $K = \mathbb{F}_2$, consider $f_1, f_2 \in K[x_1, x_2]$, where $f_1 = x_1x_2 + x_2$, and $f_2 = x_1x_2 + x_1 + 1$. Let us follow the steps of the algorithm in Proposition 6.2.2.

1) Let $G = \emptyset$.

3) If we choose the standard strategy, we have to introduce one new indeterminate $x_3$. The updated polynomials are $f_1 = x_3 + x_2$ and $f_2 = x_3 + x_1 + 1$. Let $G = \{x_1x_2 + x_3\}$.

4) Let $C = \{X_1 \vee \neg X_3,\ X_2 \vee \neg X_3,\ \neg X_1 \vee \neg X_2 \vee X_3\}$.

5) The corresponding clausal inequalities $I_c$ are $-X_1 + X_3 \leq 0$, $-X_2 + X_3 \leq 0$, and $X_1 + X_2 - X_3 - 1 \leq 0$.

6) Let $S_1 = \{x_3\}$, and $S_2 = \{x_3\}$. Let $s_1 = 2$, and $s_2 = 3$.

7) Introduce new integer indeterminates $K_1, K_2$ and write down the linear inequalities $I_1 \ : \ K_1 \leq 1$, and $I_2 \ : \ K_2 \leq 1$.

8) Form the following equations.

$$F_1 \ : \ X_2 + X_3 - 2K_1 = 0$$
$$F_2 \ : \ X_1 + X_3 + 1 - 2K_2 = 0$$

9) Introduce slack variables $0 \leq Z_1, \ldots, Z_5 \leq \infty$ to form the linear equations from linear inequalities.

$$E_1 \ : \ -X_1 + X_3 + Z_1 = 0, \ E_2 \ : \ -X_2 + X_3 + Z_2 = 0,$$
$$E_3 \ : \ X_1 + X_2 - X_3 + Z_3 - 1 = 0,$$
$$E_c \ : \ K_1 + Z_4 = 1, \ E_c \ : \ K_2 + Z_5 = 1$$

10) The system of linear Diophantine equations $\{E_i, F_i, E_c\}$ is given by

$$F_1 \ : \ X_2 + X_3 - 2K_1 = 0, \ F_2 \ : \ X_1 + X_3 - 2K_2 + 1 = 0,$$
$$E_1 \ : \ -X_1 + X_3 + Z_1 = 0, \ E_2 \ : \ -X_2 + X_3 + Z_2 = 0,$$
$$E_3 \ : \ X_1 + X_2 - X_3 + Z_3 - 1 = 0,$$
$$E_c \ : \ K_1 + Z_4 = 1, \ E_c \ : \ K_2 + Z_5 = 1$$

The matrix equation of this system is $\mathcal{A}\mathcal{X} = \mathcal{B}$, where

$$\mathcal{A} = \begin{pmatrix} 0 & 1 & 1 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & -2 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \mathcal{X} = \begin{pmatrix} X_1 \\ X_2 \\ X_3 \\ K_1 \\ K_2 \\ Z_1 \\ Z_2 \\ Z_3 \\ Z_4 \\ Z_5 \end{pmatrix}, \mathcal{B} = \begin{pmatrix} 0 \\ -1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

The Smith normal form of the matrix $\mathcal{A}$ is $\mathcal{D} = \mathcal{P}\mathcal{A}\mathcal{Q}$, where

$$\mathcal{P} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}, \mathcal{D} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix},$$

$$\mathcal{Q} = \begin{pmatrix} -1 & 1 & -1 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 & 0 & 0 & 0 \\ 1 & -1 & -1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 2 & 2 & 0 & 0 & -1 & 0 & -2 & 1 & 0 & 0 \\ -2 & 0 & 2 & 1 & -2 & -2 & 0 & 0 & 1 & 0 \\ 3 & -1 & -1 & -1 & 2 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

The general solution given by the above decomposition is $X_1 = -t_1 + t_2 - t_3, X_2 = -t_1 - t_2 + t_3 + 1, X_3 = t_1 - t_2 - t_3 + 1, K_1 = -t_2 + 1, K_2 = -t_3 + 1, Z_1 = -2t_1 + 2t_2 - 1, Z_2 = -2t_1 + 2t_3, Z_3 = 3t_1 - t_2 - t_3 + 1, Z_4 = t_2, Z_5 = t_3$, where $t_1, t_2, t_3 \in \mathbb{Z}$.

11) Let $X_1 \leq 1$, $X_2 \leq 1$ and $X_3 \leq 1$.

12) Let $C = t_1 + t_2 + t_3$. Now use an IP solver to minimize $C$ subject to the feasible region for the integer programming problem given by the general solution and the bounds of steps 7), 9) and 11).

13) Choose values for $X_1$ and $X_2$ from the solution provided by an IP solver. This will return $(1, 0)$.

### 6.2.3 Experimental Results

Now we present our observations and results from experiments with the algorithms in Proposition 6.2.2. The algorithm operates in two steps. Firstly, it finds a general integer solution. Secondly, it solves a mixed integer linear programming problem to find a non-negative solution. As described in Section 6.1.1 we use the Smith normal form for resolving the primary step. We use the computer algebra system PARI/GP [149] which is designed for fast computations in number theory, for computing the Smith normal form. For resolving the secondary step we use the commercial linear optimization tool CPLEX by ILOG [102]. CPLEX has a users choice for emphasis on feasibility or optimality. We choose emphasis on feasibility because we are not interested in optimality and stop after we found the first solution because we assume that there is only one solution (in most of the cases). We use CPLEX version 12.2. We run PARI/GP and CPLEX on a laptop with a 2.13 GHz Intel Pentium P6200 Dual Core processor and 4GB RAM. Moreover, CPLEX can be parallelized. We run CPLEX in parallel using two threads whereas PARI/GP is using only a single thread. Since we are using the full potential of Section 5.5, we use the notation and terminology from that section. Furthermore, the timings for the conversion algorithms were ignored, since they were not implemented efficiently and should be seen as a preprocessing step. In the following table we use the standard strategy and collect the timings in seconds for the solution of polynomial systems with the `matsnf(...)` command of computer algebra system PARI/GP (see [149]) and with the CPLEX optimization tool. The S-boxes were modelled using the full set of 14 equations each.

| System | Equations | Variables | Matrix Size | NZPR | SNF | CPLEX |
|---|---|---|---|---|---|---|
| CTC(2,2) | 98 | 54 | 376×490 | 3.3138 | 2 | 0.6 |
| CTC(2,3) | 144 | 78 | 558×726 | 3.3279 | 6 | 8 |
| CTC(3,2) | 147 | 81 | 564×735 | 3.3156 | 7 | 2.8 |
| CTC(3,3) | 216 | 117 | 837×1089 | 3.3297 | 23 | 23 |
| CTC(3,4) | 285 | 153 | 1110×1443 | 3.3369 | 60 | 594 |
| CTC(4,3) | 288 | 156 | 1116×1452 | 3.3306 | 60 | 605 |
| CTC(4,4) | 380 | 204 | 1480×1924 | 3.3378 | 150 | 4486 |

Table 6.5: Timings for solving systems of linear Diophantine equations

In Table 6.5, NZPR denotes the average number of non-zero elements per row and SNF denotes the timings for computing Smith normal forms. As one can see from the algorithm in Proposition 6.2.2, it is also possible to use the LP, QLP and DLP

strategies. However, these strategies apparently do not result in useful speed-ups and are omitted.

As one can see from the table that the Smith normal form seems to take little time, the hard part is to solve the IP problem. We are solving standard cryptographic systems which carry additional useful information and the conversion algorithm also preserves this information. For instance, the resulting system of linear Diophantine equations from the conversion algorithm has only few (or a unique) minimal non-negative integer solutions. Furthermore, each coordinate of the minimal non-negative integer solution is 0, 1 or takes a small non-negative integer value with in the bounds given by steps 7) and 11). Furthermore, the integer of maximum magnitude in the Smith normal form is reasonably small. For instance, in case of CTC(3,3) it is 41. These observations lead us to the following remarks.

a) As we know the classical Smith normal form algorithms perform an elimination process with some gcd computations over the integers or modulo large primes (see [100, 167]). Instead of performing computations modulo large primes we can restrict ourself to small primes for our special systems of linear Diophantine equations. Therefore, the modular approach [100, 90], which has the ability to effectively control intermediate swell, could be more effective.

b) As one can see from the table the hard part is to solve the IP problem. There are several different approaches for computing the general solution, e.g. by Hermite normal forms, by computing the Kernel or by the LLL-Algorithm. A different approach for computing the general solution may provide easier IP problem.

The last thing that we would like to point out is the sparsity and structure of the resulting system of linear Diophantine equations. Sparsity and structure can be seen by Table 6.5 and Figure 6.1 respectively. Some attempts for designing algorithms for computing the Smith normal form using efficient sparse integer matrix computations can be found in [71]. For a precise determination of its pros and cons, further experiments are needed.

As one can see from Table 6.5, for very sparse systems the running time of these techniques compare favorably to the running time of the Gröbner basis techniques in Chapter 3. In the spirit of the techniques studied above, it is obviously possible to generate a number of further variations of the conversion algorithms which have the potential to speed up the computation of the Smith normal form and IP solvers. Thus these techniques deserve further investigation and experimentation. Therefore, we

Figure 6.1: Structure and sparsity of linear Diophantine System for CTC(1,1)

believe that if such techniques are developed they can be able to solve sparse systems of polynomial equations involving a large number of indeterminates.

Chapter 7

# Techniques Using Numerical Analysis

In this chapter we address some approaches to apply numerical methods for solving systems of polynomial equations over $\mathbb{F}_2$. We study recent suggestions of transferring polynomial equations over $\mathbb{F}_2$ into polynomial equations over $\mathbb{R}$. We reconsider these transformation techniques to obtain a system of nonlinear polynomial equations over $\mathbb{R}$ which can be solved using different algorithms from numerical analysis. In particular, this enables us to investigate the use of numerical analysis techniques such as the homotopy continuation methods and Newton's method. We use the full potential of Newton methods and path tracking techniques of homotopy continuation methods for polynomial system solving over $\mathbb{F}_2$. We generalize the approach in [24, 124] and extend this work further. Furthermore, we present some ad-hoc tricks for improving the performance of the above described methods.

## 7.1 Newton Methods for Nonlinear Systems

This section presents a quick overview of various Newton methods used in this chapter. We use Newton methods for approximating solutions of systems of polynomial equations. We are interested in the *special* case, as described in Section 7.3, of solving nonlinear polynomial systems of equations. For this problem we use iterative (Newton) methods which need an initial starting guess. We do not describe the methods in detail, but we explain the reasons for the choice of the specific methods. For details and proofs we refer to the books [65, 147]. For an extensive coverage of the Newton

methods and their convergence analysis we refer to [53, 52, 144, 41, 42].

Let $\mathbb{R}[X_1, \ldots, X_n]$ be the polynomial ring over the field of real numbers. Let $F_1, \ldots, F_\ell$ be polynomials in the polynomial ring $\mathbb{R}[X_1, \ldots, X_n]$. We are interested in Newton methods for solving the following nonlinear polynomial systems of equations:

$$\begin{aligned} F_1(X_1, \ldots, X_n) &= 0 \\ &\vdots \\ F_\ell(X_1, \ldots, X_n) &= 0 \end{aligned} \tag{7.1}$$

### Global and Local Convergence

There are two very important terms which are used in numerical analysis. The first one is *locally convergent*. It refers to the situation that sufficiently good initial guesses (starting values) of the solution are assumed for convergence of the solving algorithm to the solution. Finding such a starting point is obviously a non-trivial task. Especially, if the dimension of the problem is high. Also note that in high dimensions, the rate of convergence of such methods tends to be slow and they perform poorly if the problem has non-isolated solutions (see [165], Chapter 6). The second *globally convergent* means that a rather general initial guess can be used for convergence. Efficient iterative methods should be able to cope with bad initial guesses. They represent a large and difficult topic in numerical analysis. In general, a numerical (Newton's) method can solve the system (7.1) in the following two ways. One advantage of using the following techniques is that well studied and efficient implementations in mathematical software packages like MatLab$^{\text{TM}}$ are available.

### Solving Square Systems of Nonlinear Equations

If the system (7.1) is square, i.e. $\ell = n$, then Newton's method can be applied directly to it. The classical method to solve such a system is the *Newton-Raphson method* [65], which needs several properties of the system to be able to converge to the solution. The method without extensions (modifications) is locally convergent. Other Newton based methods include techniques for global convergence with specific prerequisites (see [65]). The major drawback of such methods in our special case (see Section 7.3) is that the system of equations (7.1) is not square. We make the system square by using some ad-hoc tricks as explained in Section 7.4. To solve such problems we use the *trust-region Newton's method* (see [52] for details). A detailed convergence analysis of this method is given in [53]. This method is available via the `fsolve(...)` command

of MatLab$^{TM}$.

### Solving Overdetermined Systems of Equations

If the system (7.1) is overdetermined, i.e. $\ell > n$, then it can be solved by transferring it to an optimization problem. Most modern and efficient algorithms for solving nonlinear systems of equations (7.1) proceed by minimizing a sum of squares. We investigate the following two solution approaches for solving the system (7.1).

### Model 1

The system of equations (7.1) can be modeled as a least square problem as follows.

$$\text{Minimize} \quad \sum_{i=1}^{\ell} F_i^2(X_1, \ldots, X_n) \tag{7.2}$$

Starting from an initial guess we attempt to find a global minimum. First of all note that the classical *Gauss-Newton method* used for this purpose is known to have poor performance. Therefore, we use an advanced variant of it called *Levenberg-Marquardt* algorithm [144]. This method is also available via the `fsolve(...)` command of MatLab$^{TM}$.

### Model 2

The system of equations (7.1) can be modeled as a least squares problem with upper and lower bounds as follows.

$$\begin{array}{ll} \text{Minimize} & \sum_{i=1}^{\ell} F_i^2(X_1, \ldots, X_n) \\ \text{Subject to} & l_i \leq X_i \leq u_i, \text{ for } i = 1, \ldots, n, \end{array} \tag{7.3}$$

where $l = (l_1, \ldots, l_n), u = (u_1, \ldots, u_n) \in \mathbb{R}^n$. Such a model is called *box-bounded optimization* problem. We use the *interior reflective Newton method* [52] to solve this model. One advantage of this method is the detailed convergence analysis [53], which is very valuable for a better understanding of the algorithm. In the interior reflective Newton method, all iterates stay between the bounds. This method is available via the `lsqnonlin(...)` command of MatLab$^{TM}$.

In all approaches above, the global optimum is 0, since in the solution all equations are equal to zero, and so is their norm. Global and local convergence in optimization problems have a slightly different meaning. Globally convergent optimization methods

do not guarantee to find the global optimum, but it is ensured that for a general initial guess the algorithm converges at least to a local optimum. These methods get into trouble if more that one local minimum exists. Another drawback of these methods is that they find only a local optimum which may not be a solution. Furthermore, they are known to have poor performance for high dimensions and depend highly on starting points.

## 7.2 Homotopy Continuation Methods

This section presents a quick overview of homotopy continuation methods used in this chapter. We use homotopy continuation methods for finding solutions for systems of polynomial equations. We focus on path following techniques for finding the unique isolated solution of a system of nonlinear polynomial equations. So we are interested in a *special* case of solving problem of nonlinear polynomial systems of equations. We do not describe the used methods in detail, but we explain the reasons for the choice of specific methods. For details and proofs we refer to the book [165] and Chapter 8 (and references therein) in the book [66].

To approximate all isolated solutions of systems of polynomial equations, numerical path following techniques have been proven reliable and efficient. Initially homotopy methods were developed to solve dense systems of polynomial equations. Later on, homotopy methods were developed to exploit special structures of the polynomial systems, in particular their sparsity. To solve sparse systems, the roots are counted by the mixed volume of the Newton polytopes and computed by means of polyhedral homotopies. In a 1996 paper, Andrew Sommese and Charles Wampler began developing a new area, *Numerical Algebraic Geometry* [165], which applies and integrates homotopy continuation methods to describe solution components of polynomial systems. These methods can be considered as symbolic-numeric, or as numeric-symbolic.

Let $\mathbb{R}[X_1, \ldots, X_n]$ be the polynomial ring over the field of real numbers. Let $F_1, \ldots, F_\ell \in \mathbb{R}[X_1, \ldots, X_n]$ be polynomials which generate a zero-dimensional ideal. To solve the polynomial system $F_1 = 0, \ldots, F_\ell = 0$, homotopy continuation methods operate in the following steps.

1) Exploit the structure of the system $F : F_1 = 0, \ldots, F_\ell = 0$ to find a root count.

2) Construct a suitable start system $G : G_1 = 0, \ldots, G_\ell = 0$ with $G_i \in \mathbb{C}[X_1, \ldots, X_n]$ (or possibly in $\mathbb{R}[x_1, \ldots, x_n]$) which has exactly as many regular solutions as the

root count.

3) The start system is embedded in the homotopy

$$H_i(X_1, \ldots, X_n, t) = tG_i + \gamma(1-t)F_i, \quad t \in [0,1],$$

where $i = 1, \ldots, \ell$ and $\gamma \in \mathbb{C}$ is a random number (possibly of magnitude one).

4) As $t$ moves from 1 to 0, numerical continuation methods trace the paths that originate at the (known) solutions of the *start system* $G$ towards the solutions of the *target system* $F$.

We say that a homotopy is *optimal* if every path leads to one solution. The good properties we expect from a homotopy are:

1) **Triviality:** The solutions for $t = 0$ are trivial to find.

2) **Smoothness:** No singularities along the solution paths occur (because of $\gamma$).

3) **Accessibility:** An isolated solution of multiplicity $m$ is reached by exactly $m$ paths.

Now we sketch the standard approaches used for tracking paths. For definitions and details we refer to the book [165]. The continuation or path-following techniques lie at the heart of standard numerical techniques. The solution paths defined by the homotopy are traced using *predictor-corrector* methods. A big issue, while tracking paths, is to control diverging paths, path crossings, singularities in paths and preventing paths from turning back. Nice techniques have been developed in the theory to remove these obstacles. The smoothness of paths is achieved by working over the *complex projective spaces*. The paths never turn back due to the smoothness property of complex polynomial homotopies. The step length is determined by *adaptive step size control* while enforcing quadratic convergence in Newton's method to avoid path crossing. For a generic choice of $\gamma$, singularities do not occur for $t < 1$. Finally, the diverging paths and the paths leading to singular roots are dealt with *end games*.

The earliest applications of homotopies for solving polynomial systems belong to the dense class, where the number of paths equals the product of the degrees in the system such as multi-homogeneous homotopies, the random product homotopies, methods to construct linear-product start systems and some general approaches to exploit product structures. In practice, as well as in our special case all systems have fewer

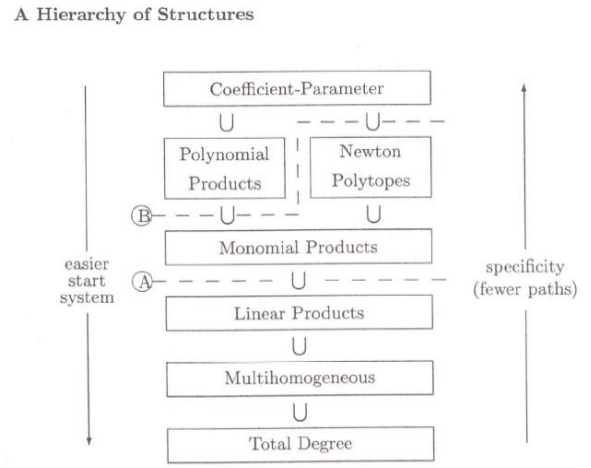



Figure 7.1: A hierarchy of homotopies.

terms than allowed by their degrees. Most interesting are the cases where we have only few (real) solutions. Polyhedral homotopies were introduced to solve such systems more efficiently. Polyhedral homotopies further specialize to cheater's homotopies and special instances of coefficient-parameter polynomial continuation. The root count requires the calculation of the mixed volume. The solving of the start system for special instances of polyhedral homotopy, cheater's and coefficient-parameter polynomial continuation may involve more work, but we may expect the homotopy to be more efficient. Schematically, a hierarchy of homotopies (and root counting methods) is given in Figure 7.1.

### Motivation for Solving Special Systems

We are interested in the *special* case of solving a system of polynomial equations which is derived from a system of polynomial equations over the field $\mathbb{F}_2$. We may assume that our system of polynomial equations has only few real solutions. In particular, it has a unique real solution. This leads us to focus on the following useful features of homotopy continuation methods to attack the unique solution.

1) For polynomial systems defined over $\mathbb{R}$, computation of a *Lex*-Gröbner basis is much slower than the homotopy continuation methods. Furthermore, parallel capabilities of continuation methods make them more powerful.

2) Since we are interested in techniques for finding the unique isolated solution of a system of nonlinear polynomial equations, why bother about all solutions.

Path-following techniques seem to be a promising idea for this purpose.

3) Using path-following techniques one may hope to get a solution of $F$ by tracking one or few paths only, especially when there is a unique solution.

4) A further promising option is the application of a user-defined homotopy. Given a parameterized family of polynomial systems, i.e. a set of polynomial systems which vary in their coefficients but not their monomials, one may first find all finite solutions of one instance from the family. Now to find all solutions for another system in the family, one need to follow only those finite solutions, and those sorts of paths tend to be very fast.

### The State of Homotopy Continuation Methods

To find real solutions homotopy continuation offers only the option of finding all roots, real and complex, and then casting out the complex ones. Unfortunately, there is no other way for finding all real solutions directly. One might hope to use continuation to follow just the real roots from the start system to the target system. As explained in [165], Section 2.2, this is *doomed to fail* as a general approach. The reason is that surprising things can happen during path following. Even if the start and target systems have the same number of real solutions. The real solutions of the start system may lead to complex solutions of the target system and vice versa (see [165], Examples 2.2.1 and 2.2.2). But if this open problem is resolved successfully, we can hope to attack the unique real solution by tracking a real path.

Since we may assume that our special system of polynomial equations has a unique real solution, we may ask about tracking one real path to get a real solution. In a private communication with C. Wampler and D. Bates, we came to know that this is one of the big open questions of the day. Suppose we have a polynomial system with total degree 1000000 but only 5 real, isolated solutions. How do we find them? So far nobody has any ideas using homotopy continuation alone. Actually, the theory underlying homotopy continuation depends on working over the complex projective spaces, and as described above real starting points of paths might lead to complex ending points and vice versa.

One more hurdle that we need to face is that homotopy continuation deals only square systems, i.e. the system should have the same number of equations as unknowns. There is a natural procedure called *randomization* for obtaining a square system from an overdetermined system (see [165], Section 13.5). The new square system has all the

properties we need to compute isolated solutions using homotopy continuation. But the new square system may have more solutions than the original overdetermined system which have to be removed afterwards. Furthermore, randomization also introduces new complex coefficients possibly of magnitude one which makes the paths smooth at the cost of working over complex projective spaces. Definitely, this does not seem to be economical.

There are three software packages that are mainly used by researchers for polynomial continuation, namely: Bertini [21], PHC [169] and HOM4PS [127]. We use Bertini for solving systems using the straightforward approach and the user-defined homotopy, and HOM4PS using polyhedral homotopies.

## 7.3 Techniques for Polynomial Conversion

In Chapters 5 and 6 we saw different conversion techniques for transferring polynomial equations over $\mathbb{F}_2$ into polynomial equations over $\mathbb{R}$. In this section we reconsider these conversion techniques to obtain as system of nonlinear polynomial equations over $\mathbb{R}$ which can be solved using different algorithms from numerical analysis. In particular, this helps us to investigate the use of numerical analysis techniques such as homotopy continuation methods and Newton's method.

As described in previous chapters, some methods for representing polynomials over $\mathbb{F}_2$ as polynomials over $\mathbb{R}$ can be found in the classical literature, but they have not been used for solving systems of polynomial equations over $\mathbb{F}_2$. In [24], an overview of possible representations is listed. Later, this study was extended slightly in [124] but the main idea behind the representation methods was basically unaltered. Recently, in [124] Lamberger et al. used the ideas mentioned in [24] for representing polynomial equations over $\mathbb{F}_2$ by polynomial equations over $\mathbb{R}$. We consider all these representation methods for formulating our conversion algorithms.

Let $\mathbb{F}_2$ be the finite field with two elements and let $f_1, \ldots, f_\ell \in \mathbb{F}_2[x_1, \ldots, x_n]$ be a set of non-zero polynomials. Let $X = \{X_1, \ldots, X_n\}$ be a set of real indeterminates, i.e. indeterminates over $\mathbb{R}$. We are interested in finding $\mathbb{F}_2$-rational solutions of the following system of polynomial equations:

$$\begin{aligned} f_1(x_1, \ldots, x_n) &= 0 \\ &\vdots \\ f_\ell(x_1, \ldots, x_n) &= 0 \end{aligned}$$

As usual the task of solving the polynomial equation system $f_1 = \cdots = f_\ell = 0$ can be rephrased as follows. Find a tuple $(a_1, \ldots, a_n) \in \{0,1\}^n$ such that

$$\begin{aligned} F_1(a_1, \ldots, a_n) &\equiv 0 \pmod 2 \\ &\vdots \\ F_\ell(a_1, \ldots, a_n) &\equiv 0 \pmod 2 \end{aligned} \tag{7.4}$$

where $F_i \in \mathbb{R}[X_1, \ldots, X_n]$ is a lifting of $f_i$ under standard representation. Thus we are looking for real solutions $(a_1, \ldots, a_n)$ of the system (7.4) which satisfy the bounds $0 \leq a_j \leq 1$. Recall Definition 5.2.5 which gives us a standard way of representing polynomials over $\mathbb{F}_2$ as polynomials over $\mathbb{R}$. Our first conversion technique in the following lemma is based on this definition.

**Proposition 7.3.1. (Real Standard Conversion (RSC))**
*Let $f_1, \ldots, f_\ell \in \mathbb{F}_2[x_1, \ldots, x_n]$ be polynomials. Then the following instructions define an algorithm which computes the standard representation of the system $f_1, \ldots, f_\ell$ of polynomials such that the residue class of a zero $(a_1, \ldots, a_n)$ of the standard representation represents a zero of the zero-dimensional radical ideal $I = \langle f_1, \ldots, f_\ell, x_1^2 + x_1, \ldots, x_n^2 + x_n \rangle$ of $\mathbb{F}_2[x_1, \ldots, x_n]$.*

1) *Reduce $f_1, \ldots, f_\ell$ modulo the field equations, i.e. make their support squarefree.*

2) *For $i = 1, \ldots, \ell$, represent $f_i$ by a polynomial $\bar{F}_i$ in $\mathbb{R}[X_1, \ldots, X_n]$ such that the coefficients of $\bar{F}_i$ are in $\{0,1\}$.*

3) *Perform the following steps 4)-6) for $i = 1, \ldots, \ell$.*

4) *Let $S_i = \mathrm{Supp}(\bar{F}_i)$. Choose a term $T \in S_i$, delete it from $S_i$, and let $F_i = T$.*

5) *Repeat the following step 5) until $S_i = \emptyset$.*

6) *Choose a term $T \in S_i$, delete it from $S_i$, and replace $F_i$ by $F_i + T - 2F_iT$.*

7) *Interreduce the system $\{F_1, \ldots, F_\ell, X_1^2 - X_1, \ldots, X_n^2 - X_n\}$.*

8) *Return the resulting system.*

*Proof.* Since we are lifting polynomials over $\mathbb{R}$ by iteratively replacing each sum $T_1 + T_2$ of terms by $T_1 + T_2 - 2T_1T_2$ in step 6), we have $F_i(a_1, \ldots, a_n) = 0$ if and only if the residue class of $(a_1, \ldots, a_n)$ in $\mathbb{F}_2^n$ represent a zero of the ideal $I$. □

**Remark 7.3.2.** As we saw in Section 5.2.3, the standard representation results in increasing degree and increasing number of terms over the real domain. To keep the degrees of the converted polynomials low, we can use the splitting introduced in Remark 5.2.7. Given a system of polynomials $f_1, \ldots, f_\ell$ over $\mathbb{F}_2$, where the degree of each $f_i$ is 2. First split these polynomials into polynomials having at most 4 terms in their support. Then consider the standard representations of the new polynomials. We shall refer it as *Splitting Real Conversion (SRC)*. Recall from Remark 5.2.7 that SRC results into a quadratic system of equations over $\mathbb{R}$.

To formulate our next conversion technique, we need to recall the following definition due to [124].

**Definition 7.3.3.** The **Fourier conversion** is given by the map $\phi : \mathbb{F}_2 = \{\overline{0}, \overline{1}\} \to \{1, -1\} \subset \mathbb{R}$ defined by $\phi(\overline{0}) = 1$ and $\phi(\overline{1}) = -1$. The map $\phi$ can be extended to a map $\Phi : \mathbb{F}_2[x_1, \ldots, x_n] \longrightarrow \mathbb{R}[X_1, \ldots, X_n]$ defined by

$$\begin{aligned} c &\mapsto \phi(c) \\ x_i &\mapsto X_i \end{aligned}$$

where $c \in \mathbb{F}_2$. Then the **Fourier representation** of a polynomial $f \in \mathbb{F}_2[x_1, \ldots, x_n]$ is $\Phi(f)$.

Note that we use the following conversion rules for addition and multiplication for Fourier representation.

$$\begin{aligned} X_i \cdot X_j &= \frac{1}{2}(1 + X_i + X_j - X_i \cdot X_j) \\ X_i + X_j &= X_i X_j \end{aligned}$$

The effect of this definition is that the Fourier representation $F$ has a solution $(a_1, \ldots, a_n) \in \{1, -1\}^n$ if and only if the tuple $(b_1, \ldots, b_n)$ with $b_i = (1 - a_i)/2$ solves $f$. The Fourier representation also results in increasing degree and increasing amount of terms over the real domain. But it has an additional advantage since variables can cancel out during the transformation, because $X_i^2 = 1$ holds. This can happen if one variable occurs in different terms in the same polynomial. The following proposition turns the idea of Fourier representation into an effective algorithm.

**Proposition 7.3.4. (Real Fourier Conversion (RFC))**

*Let $f_1, \ldots, f_\ell \in \mathbb{F}_2[x_1, \ldots, x_n]$ be polynomials. Then the following instructions define an algorithm which computes the Fourier representation $F$ of the system $f_1, \ldots, f_\ell$ of polynomials such that the tuple $(a_1, \ldots, a_n) \in \{-1, 1\}^n$ is a zero of Fourier representation if and only if the residue class of $(b_1, \ldots, b_n)$ with $b_i = (1 - a_i)/2$, in $\mathbb{F}_2^n$ represent a zero of the zero-dimensional radical ideal of $\mathbb{F}_2[x_1, \ldots, x_n]$.*

*1) Reduce $f_1, \ldots, f_\ell$ modulo the field equations, i.e. make their support squarefree.*

*2) For $i = 1, \ldots, \ell$, represent $f_i$ by a polynomial $\bar{F}_i$ in $\mathbb{R}[X_1, \ldots, X_n]$ such that the coefficients of $\bar{F}_i$ are in $\{0, 1\}$.*

*3) Let $F = \emptyset$. Perform the following steps 4)-5) for $i = 1, \ldots, \ell$.*

*4) Let $S_i = \mathrm{Supp}(\bar{F}_i)$ and let $s_i = \#S_i$.*

*5) For $j = 1, \ldots, s_i$, replace every product $X_r X_s$ by $\frac{1}{2}(1 + X_r + X_s - X_r X_s)$ in $T_j \in S_i$. Now let $F_i = \sum_{j=1}^{s_i} T_j$.*

*6) For $i = 1, \ldots, \ell$, append $F_i - 1$ to $F$.*

*7) For $\alpha = 1, \ldots, n$, append $X_\alpha^2 - 1$ to $F$.*

*8) Return the resulting system $F$.*

*Proof.* Since we are lifting polynomials over $\mathbb{R}$ by iteratively replacing every product $X_r X_s$ by $\frac{1}{2}(1 + X_r + X_s - X_r X_s)$ in step 5), we have $F_i(a_1, \ldots, a_n) = 0$ if and only if the residue class of $(b_1, \ldots, b_n)$ with $b_i = (1 - a_i)/2$, in $\mathbb{F}_2^n$ represent a zero of the zero-dimensional radical ideal $I$. □

Since we know the exact bounds on indeterminates for our problem, this property may be very useful. On the one hand it makes the analysis of the system easier, since we only have to consider the hypercube $\{0, 1\}^n$ (or $\{-1, 1\}^n$), and on the other hand it is ensured that bad properties outside the cube do not influence the convergence of algorithms.

## 7.4 Experimental Results

In this section we discuss observations and results from experiments with Newton methods and homotopy continuation methods. The softwares used for the total degree homotopy, the polyhedral homotopy and Newton's method computations are Bertini

[21], HOM4PS [127] and MatLab$^{TM}$ respectively. We run all computations on a laptop with a 2.13 GHz Intel Pentium P6200 Dual Core processor and 4GB RAM but we are using only a single thread out of two. The timings for the conversion algorithms were ignored, since they were not implemented efficiently and should be seen as a preprocessing step.

Given a system of polynomials $f_1, \ldots, f_\ell \in \mathbb{F}_2[x_1, \ldots, x_n]$. Using Propositions 7.3.1 and 7.3.4, we can compute the real (standard or Fourier) representations $F_1, \ldots, F_\ell$ of the polynomials $f_1, \ldots, f_\ell$. In particular, we have the following system of polynomial equations over $\mathbb{R}$.

$$F_1 = 0, \ldots, F_\ell = 0 \tag{7.5}$$

As given by Propositions 7.3.1 and 7.3.4, to exclude complex solutions we need to modify the system of equations (7.5) by including the equations $X_1^2 - X_1 = 0, \ldots, X_n^2 - X_n = 0$ or $X_1^2 - 1 = 0, \ldots, X_n^2 - 1 = 0$ depending on which type (standard or Fourier) of representation is used. Now the modified system can be solved using numerical methods for real solutions. From now on in this section, by $F$ we denote the system

$$F_1 = 0, \ldots, F_\ell = 0, F_{\ell+1} = 0, \ldots, F_m = 0, \tag{7.6}$$

where the polynomials $F_1, \ldots, F_\ell$ are the real (standard or fourier) representations of the polynomials $f_1, \ldots, f_\ell$, and $F_{\ell+1}, \ldots, F_m$ are the polynomials $X_1^2 - X_1, \ldots, X_n^2 - X_n$ if standard representation is used or the polynomials $X_1^2 - 1, \ldots, X_n^2 - 1$ if Fourier representation is used.

### 7.4.1 Experimental Results for Continuation Methods

Now we employ homotopy continuation methods for solving the system $F$. First note that we will not consider SRC (see Remark 7.3.2) for solving with homotopy methods because the number of paths tracked strongly depend on the number of variables involved. Therefore, increasing the number of variables to keep the degree of the polynomials $F_i$ low does not help to reduce the difficulty of solving. In the following table we use homotopy continuation methods for solving some small instances of the HFE cipher. Since we are not splitting the polynomial $f_i$ before applying the conversion algorithms, the degree of the polynomial $F_i$ is not quadratic. To perform our experiments we have developed two ApCoCoA packages called bertini and hom4ps. These packages call the full artillery of Bertini [21] and HOM4PS [127] for computations with homotopy continuation methods inside ApCoCoA [12].

| | Fourier | | | | Standard | | | |
|---|---|---|---|---|---|---|---|---|
| | Total Degree | | Polyhedral | | Total Degree | | Polyhedral | |
| System | Paths | Time | Paths | Time | Paths | Time | Paths | Time |
| HFE(4) | 16 | 0.5 | 16 | 0.07 | 16 | 0.07 | 16 | 0.06 |
| HFE(5) | 243 | 3 | 123 | 2 | 1024 | 50 | 234 | 5.7 |
| HFE(6) | 5120 | 250 | 1269 | 58 | 15625 | 1800 | 1433 | 144 |
| HFE(7) | 93312 | - | 10044 | 1540 | 93312 | - | 10068 | 2057 |

Table 7.1: Path Following Techniques

Obviously, we can not hope to solve large systems of polynomial equations with homotopy continuation methods as Table 7.1 shows. The reasons are very clear. Firstly, the homotopy continuation methods are very slow in high dimensions. Secondly, the system $F$ of equations is overdetermined, therefore it needs to be randomized before embedding it into the homotopy for tracking paths. Recall that randomization introduces a lot of junk and complexity to work smoothly over complex projective spaces. The polyhedral homotopies involve the computation of the mixed volume of Newton polytopes for finding a root count. Obviously, computing mixed volumes is a very hard problem.

The only thing that we can add here is the following. Since we are tracking paths one by one and the process of tracking paths can be parallelized, we can stop just after finding one successful path (real solution). But this will not contribute much to the difficulty of solving. The only hope left is the Coefficient-Parameter homotopy (or the user defined homotopy) which works as follows.

Given an overdetermined system of polynomials $f_1, \ldots, f_\ell \in \mathbb{F}_2[x_1, \ldots, x_n]$ having unique real zero. Using Propositions 7.3.1 and 7.3.4, obtain a system of polynomials $F : F_1, \ldots, F_m$ over $\mathbb{R}$ having a unique real zero. Now construct a start system $G : G_1, \ldots, G_m$ which varies in coefficients but not in terms with the target system $F$. Furthermore, the start system has a unique real zero. Now track the path which originates at the zero of the start system $G$ and terminates at the zero of the target system $F$. There are three issues with this approach.

Firstly, the system is overdetermined which forces us to randomize. This destroys the uniqueness of the zero very badly. Secondly, the construction of a start system $G$ which differs in coefficients but not in terms with the target system is a very hard job. The construction of start systems, especially of the type mentioned above, is a non-trivial task. There is no general method to do that. It is done on case-by-case basis. Furthermore, we also have the restriction of having a unique solution. So this is an

open problem with applications in cryptanalysis. Thirdly, tracking the paths starting at real solutions and terminating at real solutions is a big open problem in itself and nobody knows anything. One attempt for constructing start systems for a very special class of polynomial systems can be found in [22].

### 7.4.2 Experimental Results for Newton Methods

Now we investigate the use of Newton methods described in Section 7.1 and discuss some ad-hoc tricks to apply them. A first attempt to use Newton's method for our special systems can be found in [124], where Lamberger et al. discussed the approach of Section 7.3 to apply numerical methods. In particular, they apply the interior reflective Newton method by Coleman and Li [53] to attack a reduced version of *Trivium* called *Bivium A*. This approach is not really practical because we need to know 75% of the original solution for choosing a good starting point. But they believe that we can do better if we use the available knowledge in the field of numerical analysis. Therefore we realize that there is a strong need to launch further investigations in this direction. We investigate several approaches for modeling and solving our special systems. A proposal for choosing starting points and some ad-hoc tricks are given to overcome the difficulty of solving. To apply Newton methods first we need to agree on some terms and concepts that are going to be used in this section time and again. In all the following tables we use the following terms to record our observations.

- **Sol.** indicates that the solver has found the solution successfully. **Opt.** indicates that the solver has found only a local optimum. All local optima are near by a corner of the hypercube where a large part of the values are either 0 or 1 (or -1,1 for Fourier representation).
- In case the solver found an optimum, **WSol** represents the number of variables whose wrong solution value is given by the optimum.
- In case the solver found an optimum, **UEq** represents the number of unsatisfied equations at the optimum.
- **SP** is the percentage of the original solution used in a random starting point.

#### Upper and Lower Bounds

In our special case we can restrict the variables $X_1, \ldots, X_n$ to the interval $[0, 1]$ (or $[-1, 1]$ for the Fourier representation). Using this fact we can define upper and lower

bounds for our problem. The lower bound is $(0, \ldots, 0)$ (respectively $(-1, \ldots, -1)$ for the Fourier representation) and the upper bound is $(1, \ldots, 1)$ for both standard and Fourier representations. Later on, in this section we see that these restrictions to the starting point for our problem are not sufficient for solving the system. What we need really is a good starting point.

### Starting Guess

Finding a good starting point is not an easy job, especially if the dimension of the problem is high. Finding a good starting point depends on the amount of available information. As mentioned in Section 7.1, locally convergent methods require a good starting point. The starting point has a high influence on the convergence. In our case we know that the variables $X_1, \ldots, X_n$ belong to $\{0, 1\}$ (respectively $\{-1, 1\}$), so we can restrict the search area to this domain. The first thing that we have observed in our experiments is that a randomly chosen starting point in $[0, 1]$ (respectively $[-1, 1]$) provides us a local minimum which is not the solution of the system (7.7). This also confirms the results in [124]. Therefore the only possibility left is to use a part of the original solution in the starting point which is a common approach in cryptanalysis. So we use the following strategy for a good starting point. We use a part of the original solution in the starting point, say $x\%$, and the remaining $(100 - x)\%$ of the solution will be fixed in the middle of the interval $[0, 1]$ (respectively $[-1, 1]$). We performed a number of experiments with different approaches (discussed in the following) that show that this strategy has better performance than any other. Therefore we focus on this strategy for choosing a starting point.

### Solving Square Systems of Nonlinear Equations

Once again consider the overdetermined system of equations $F : F_1 = 0, \ldots, F_\ell = 0, F_{\ell+1} = 0, \ldots, F_m = 0$. This system can be modeled into the following square system.

$$\begin{array}{l} \sum_{i=1}^{\ell} F_i^2 + F_m^2 = 0 \\ F_{\ell+1} = 0, \ldots, F_{m-1} = 0 \end{array} \tag{7.7}$$

The polynomials $F_{\ell+1}, \ldots, F_{m-1}$ force the variables $X_1, \ldots, X_{n-1}$ to be 0-1 (respectively $-1$-1) if standard (respectively Fourier) representation is used. The square system (7.7) can be solved using the `fsolve(...)` command of MatLab. We use the trust-region-dogleg algorithm implemented in the `fsolve(...)` command. Note that this is the only

algorithm in MatLab that is specially designed to solve nonlinear systems of equations. All other algorithms attempt to minimize the sum of squares which we discuss later in this section.

| System | Equations | Variables | SP | Time | UEq | WSol | status |
|---|---|---|---|---|---|---|---|
| CTC(2,2) | 98 | 54 | 25% | 3 | 0 | 0 | successful |
| CTC(3,3) | 216 | 117 | 50% | 5 | 0 | 0 | successful |
| CTC(4,4) | 380 | 204 | 50% | 20 | 0 | 0 | successful |
| CTC(5,5) | 605 | 330 | 50% | 230 | 4 | 4 | premature |
| CTC(6,6) | 864 | 468 | 50% | 329 | 16 | 16 | premature |

Table 7.2: trust-region-dogleg algorithm

Table 7.2 lists the results for the Fourier representation. If we use a part of the original solution in the starting point, we need at least 50% of all variables in the system to obtain the complete solution. This percentage seems to grow with the size of the system. A little good news is that we can recover key variables even if the solver stops prematurely. Premature solutions of CTC(5,5) and CTC(6,6) contain 4 and 16 wrong solution coordinates respectively, and result in 4 and 16 unsatisfying equations respectively. The unsatisfying equations are linear ones. One can hope to perform more iterations to find the complete solution from a premature solution. But this does not help. After exceeding the default limits of parameters of the command `fsolve(...)`, the solver does not progress anymore. It seems to stuck at a premature solution. The standard representation does not provide satisfactory results and is therefore not discussed in details. For instance, consider the standard representation of CTC(2,2). After fixing 50% of the starting point the solver stops prematurely after 90 seconds and the premature solutions contain 14 wrong solution coordinates which result in 24 unsatisfying equations.

### Solving Overdetermined Systems of Nonlinear Equations

Consider the system of equations $F_1 = 0, \ldots, F_\ell = 0$ which is a real (standard or Fourier) representation of the system $f_1 = 0, \ldots, f_\ell = 0$. Most modern and efficient algorithms for solving nonlinear systems of equations proceed by minimizing the sum of squares $\sum_{i=1}^{\ell} F_i^2$. The drawback of such techniques is that they try to find a local optimum which may not be a solution of the system. But if lower and upper bounds are given explicitly (as in our case) one may hope to find the local minimum which is also a solution of the system. In the following we investigate the two optimization

models described in Section 7.1.

**Model 1**

We model the system of equations $F_1 = 0, \ldots, F_\ell = 0$ as a least squares problem as follows.

$$\text{Minimize} \quad \sum_{i=1}^{\ell} F_i^2(X_1, \ldots, X_n) \tag{7.8}$$

Staring from a starting point we attempt to find a global minimum. Actually, the global minimum in this case is zero, since in the solution all equations are equal to zero and so is their norm. As we know, globally convergent optimization methods do not guarantee to find a global optimum but they ensure that for a general initial guess the algorithm converges at least to a local optimum. Since the classical Gauss-Newton method is known to have poor performance, we use its advanced variant called Levenberg-Marquardt algorithm which is implemented in the `fsolve(...)` command of MatLab.

| System | SP | Time | UEq | WSol | status |
|---|---|---|---|---|---|
| CTC(2,2) | 25% | 2 | 0 | 0 | sol. |
| CTC(3,3) | 50% | 7 | 0 | 0 | sol. |
| CTC(4,4) | 50% | 30 | 0 | 0 | sol. |
| CTC(5,5) | 50% | 291 | 4 | 4 | opt. |
| CTC(6,6) | 50% | 740 | 29 | 22 | opt. |

Table 7.3: Levenberg-Marquardt algorithm using Fourier representation

| System | SP | Time | UEq | WSol | status |
|---|---|---|---|---|---|
| CTC(2,2) | 25% | 12 | 0 | 0 | sol. |
| CTC(3,3) | 50% | 90 | 0 | 0 | sol. |
| CTC(4,4) | 50% | 189 | 0 | 0 | sol. |
| CTC(5,5) | 50% | 3930 | 4 | 6 | opt. |
| CTC(6,6) | 50% | 6112 | 23 | 25 | opt. |

Table 7.4: Levenberg-Marquardt algorithm using standard representation

Tables 7.3 and 7.4 list the results for the Fourier representation and the standard representation respectively. The SRC (see Remark 7.3.2) seems to perform a little better as given by Table 7.5 but only up to a certain limit. For instance, any conversion technique applied to CTC(7,7) provides an optimum which does not even help to recover the key variables.

| System | SP | Time | UEq | WSol | status |
|---|---|---|---|---|---|
| CTC(2,2) | 40% | 29 | 0 | 0 | sol. |
| CTC(3,3) | 50% | 90 | 0 | 0 | sol. |
| CTC(4,4) | 50% | 494 | 0 | 0 | sol. |
| CTC(5,5) | 50% | 1483 | 0 | 0 | sol. |
| CTC(6,6) | 50% | 3027 | 0 | 0 | sol. |

Table 7.5: Levenberg-Marquardt algorithm using splitting standard representation

**Model 2**

We model the system of equations $F_1 = 0, \ldots, F_\ell = 0$ as a least squares problem with upper and lower bounds as follows.

$$\begin{array}{ll} \text{Minimize} & \sum_{i=1}^{\ell} F_i^2(X_1, \ldots, X_n) \\ \text{Subject to} & l_i \leq X_i \leq u_i, \text{ for } i = 1, \ldots, n, \end{array} \tag{7.9}$$

where $l = (l_1, \ldots, l_n), u = (u_1, \ldots, u_n) \in \mathbb{R}^n$. This model was also studied by Lamberger et al. in [124]. For Bivium A (a reduced version of Trivium) they have observed that we need at least 75% of all variables in the system to obtain the complete solution. In the following we list results for CTC. Table 7.6 lists the results for the Fourier representation, the standard representation also provides more or less the same statistics.

| System | SP | Time | UEq | WSol | status |
|---|---|---|---|---|---|
| CTC(2,2) | 25% | 2 | 0 | 0 | sol. |
| CTC(3,3) | 50% | 3 | 0 | 0 | sol. |
| CTC(4,4) | 50% | 5 | 0 | 0 | sol. |
| CTC(5,5) | 50% | 40 | 0 | 0 | sol. |
| CTC(6,6) | 50% | 88 | 0 | 0 | sol. |

Table 7.6: interior-reflective Newton method

We can summarize the numerical techniques as follows. All the techniques investigated in this section provide more or less the same results. We need at least 50% of all variables in the system to obtain the complete solution of CTC instances up to CTC(6,6). We ran a number of experiments for CTC(7,7) using 50% part of the original solution in the starting point but we were not able to solve it. This shows clearly that as the dimension grows, numerical solvers start to perform poorly. By starting at a random starting point, without using a part of original solution, we can achieve

optima but not the solution. We are not able to solve big instances of the CTC with the above methods without providing much information. The failure of continuous optimization algorithms for finding the solution of a nonlinear system compels us for developing mixed integer nonlinear programming algorithms. As we saw in Section 6.1, mixed integer nonlinear programming algorithms seem to be rather efficient and desire to be the subject of further investigations.

We hope that the investigation in this chapter will help for a better understanding of using numerical methods for solving systems over finite fields. Since we are interested in finding real solutions, there are a few other methods for finding real solutions numerically. Such as cellular exclusion, SDP methods, Cylindrical algebraic decomposition and Khovanskii-Rolle continuation (see for instance [22] and [165], Section 2.2). But none of them works in full generality or on large problems.

# Appendix A

# Packages Bertini and HOM4PS

The CAS ApCoCoA, an acronym of *Applied Computations in Commutative Algebra* is based on the CAS CoCoA. It is primarily designed for working with *real-problems* by using the symbolic computations methods of CoCoA and by developing new libraries for related computations. The CAS ApCoCoA is available free of charge via the internet and can be downloaded from the WWW page

`http://www.apcocoa.org/`

For a short introduction to CoCoA and for the help on getting started with it we refer to [120], Appendix A. The ApCoCoA works exactly in the same way as explained there.

Bertini [21] is a general-purpose solver, written in C, that was created for research on polynomial continuation. The purpose of Bertini is the numerical solution of systems of polynomial equations. HOM4PS [127] is a software package which implements the polyhedral homotopy continuation method for solving polynomial systems. The polyhedral homotopies are established to approximate all the isolated zeros of a polynomial system using the continuation method [165]. Due to fewer homotopy paths, it yields a drastic improvement over the classical linear homotopies for solving sparse polynomial systems.

In Chapter 7 we talked about techniques using numerical analysis. In Section 7.1 we have discussed homotopy methods and in Section 7.4 we have used these methods for solving systems of polynomial equations over the finite field $\mathbb{F}_2$. For using homotopy continuation methods in our algebraic settings we have developed two ApCoCoA packages `bertini` and `hom4ps`. These packages call the full artillery of Bertini and HOM4PS for computations with homotopy continuation methods inside ApCoCoA.

## A.1 Available Functions

In the following we give a short description of the functions available in the packages `bertini` and `hom4ps` for working with the homotopy continuation methods. This description is also available as a part of the documentation of these packages and can be seen from the help menu of ApCoCoA. First we describe functions which are available in the package `bertini`. All the following functions are using the ApCoCoAServer. You will have to start the ApCoCoAServer in order to use them.

`BSolve(P,SysTyp)`

**Purpose:** Solves a zero-dimensional homogeneous or non-homogeneous polynomial system of equations with default configurations.

**Syntax** `Bertini.BSolve(P:LIST, SysTyp:STRING):LIST`

**Input** 1st parameter `P`, a list of polynomials. 2nd parameter `SysTyp`, type of the system `P`.

**Output** A list of lists where each list contains a finite (or real) solution of the system `P`.

**Example** Consider the polynomial ring $\mathbb{Q}[x, y]$. To solve the polynomial system of equations $x^2 + y^2 - 5 = 0, xy - 2 = 0$, one has to run the following commands in ApCoCoA interactive window:

```
Use QQ[x,y];
P := [x^2+y^2-5, xy-2];
SysTyp := "Nhom";
Bertini.BSolve(P, SysTyp);
```

The output of above commands is the following list of lists where each list gives a finite solution of the polynomial system $x^2 + y^2 - 5 = 0, xy - 2 = 0$.

```
[ [ Vector(400000000000003/200000000000000, -3416759775755413/50000
000000000000000000000000), Vector(9999999999999927/100000000000000
00, 8966048861359829/10000000000000000000000000000000) ], [ Vector(2
499999999999963/2500000000000000, 5007041073746771/1000000000000000
00000000000000), Vector(249999999999999/125000000000000, -108918318
```

```
4148021/250000000000000000000000000) ], [ Vector(-999999999999996
9/10000000000000000, 191792591213411/125000000000000000000000000000
), Vector(-1999999999999999/1000000000000000, 2443331461729629/2500
000000000000000000000000000) ], [ Vector(-250000000000001/125000000
000000, 4347064850996171/100000000000000000000000000000), Vector(-
9999999999999943/10000000000000000, -2154842536286333/5000000000000
00000000000000000) ]]
```

**Note:**

In the previous example and in all examples onward each vector in a list represents a complex number with its first coordinate as real part and second coordinate as imaginary part.

`BZCSolve(P, SysTyp, ConfigSet)`

**Purpose:** Solves a zero-dimensional homogeneous or non-homogeneous polynomial system of equations using configurations provided by the user.

**Syntax** `Bertini.BZCSolve(P:LIST, SysTyp:STRING, ConfigSet:LIST):LIST`

**Input** 1st parameter `P`, a list of polynomials. 2nd parameter `SysTyp`, type of the system `P`, 3rd parameter `ConfigSet`, a list of configuration settings.

**Output** A list of lists where each list contains a finite (or real) solution of the system `P`.

**Example** Consider the polynomial ring $\mathbb{Q}[x, y]$. To solve the polynomial system of equations $x^2 + y^2 - 5 = 0, xy - 2 = 0$, one has to run the following commands in ApCoCoA interactive window:

```
Use QQ[x,y];
P := [x^2+y^2-5, xy-2];
SysTyp := "Nhom";
ConfigSet := ["MPTYPE: 1", "PRECISION: 128"];
Bertini.BSolve(P, SysTyp, ConfigSet);
```

The output of the above commands is the following list of lists where each list gives a finite solution of the polynomial system $x^2 + y^2 - 5 = 0, xy - 2 = 0$.

```
[ [ Vector(50000000000000087008007957145675363120 9/5000000000000000
0000000000000000000000, 41243336046164965623860294533917359418 1/20
0000000000000000000000000000000000000000000000000000), Vector(19999
9999999999920289038441185562687901/1000000000000000000000000000000000
0000000, -49186133030677262498653513475068419443 03/5000000000000000
0000000000000000000000000000000000000000) ], [ Vector(99999999999999
690769169154815028376706 3/5000000000000000000000000000000000000000,
40268217839917330215650243360889592924 91/1000000000000000000000000000
0000000000000000000000000000), Vector(10000000000000081195248376154
06734621127/1000000000000000000000000000000000000000, -92028283750
00265851232972557923998357683/1000000000000000000000000000000000000000
00000000000000000)], [ Vector(-19999999999999814706219551220586458
54307/1000000000000000000000000000000000000000, -221929688059643722
09535959637382238628 47/1000000000000000000000000000000000000000000000
0000000000), Vector(-10000000000000164292809521668176191954 09/10000
0000000000000000000000000000000000, 22468952332513846015491133458 1
0086172711/1000000000000000000000000000000000000000000000000000000)]
, [ Vector(-99999999999999867144157523905695330033 43/10000000000000
0000000000000000000000000, 23763311504509275614227639972243274983
41/1000000000000000000000000000000000000000000000000000000), Vector
(-20000000000000012651527955671853917741 7/1000000000000000000000000
00000000000000, -40966133137841317749350094520432260647 3/2500000000
0000000000000000000000000000000000000000) ] ]
```

**Note:**

For details about configuration settings please refer to Bertini manual [21].

`BMSolve(P)`

**Purpose:** Solves a zero-dimensional non-homogeneous system using multi-homogenization and default configurations.

**Syntax** `Bertini.BMSolve(P:LIST):LIST`

**Input** A list of polynomials `P`.

**Output** A list of lists where each list contains a finite solution of the system `P`.

**Example** Consider the polynomial ring $\mathbb{Q}[x, y]$. To solve the polynomial system of equations $x^2 + y^2 - 5 = 0, xy - 2 = 0$, one has to run the following commands in ApCoCoA interactive window:

```
Use QQ[x,y];
P := [x^2+y^2-5, xy-2];
Bertini.BMSolve(P);
```

The output of above commands is the following list of lists where each list gives a finite solution of the polynomial system $x^2 + y^2 - 5 = 0, xy - 2 = 0$.

```
[ [ Vector(1000000000000001/1000000000000000, -2305082859180703/10
0000000000000000000000000000), Vector(1999999999999971/10000000000
00000, 4135565953005217/10000000000000000000000000000000) ], [ Vecto
r(1000000000000003/500000000000000, 2604577577014449/5000000000000
0000000000000000000), Vector(500000000000001/500000000000000, -619892
334722183/25000000000000000000000000000000) ], [ Vector(-2, 172481033
3092189/10000000000000000000000000000000), Vector(-500000000000001/
500000000000000, -355984244774691/20000000000000000000000000000000)],
 [ Vector(-9999999999999971/10000000000000000, -4053926086793577/1
0000000000000000000000000000000), Vector(-1999999999999999/10000000
00000000, -3669041992638223/50000000000000000000000000000000) ] ]
```

`BCMSolve(P, ConfigSet)`

**Purpose:** Solves a zero-dimensional non-homogeneous polynomial system of equations using multi-homogenization and user configurations.

**Syntax** `Bertini.BCMSolve(P:LIST, ConfigSet:LIST):LIST`

**Input** 1st parameter `P`, a list of polynomials. 2nd parameter `ConfigSet`, list of configuration settings.

**Output** A list of lists where each list contains a finite solution of the system `P`.

**Example** Consider the polynomial ring $\mathbb{Q}[x, y]$. To solve the polynomial system of equations $x^2 + y^2 - 5 = 0, xy - 2 = 0$, one has to run the following commands in ApCoCoA interactive window:

```
Use QQ[x,y];
P := [x^2+y^2-5, xy-2];
SysTyp := "Nhom";
ConfigSet := ["MPTYPE: 2"];
Bertini.BCMSolve(P, ConfigSet);
```

The output of above commands is the following list of lists where each list gives a finite solution of the polynomial system $x^2 + y^2 - 5 = 0, xy - 2 = 0$.

```
[ [ Vector(9999999999999999/10000000000000000, -643977180168769/125
000000000000000000000000000000), Vector(2, 1660674691787513/500000000
000000000000000000000000) ], [ Vector(-2000000000000001/1000000000000
000, 584020313856301/500000000000000000000000000000), Vector(-99999
99999999999/10000000000000000, 45486167963413/125000000000000000000
000000000) ], [ Vector(2, 2989952880295369/100000000000000000000000
0000000), Vector(9999999999999993/10000000000000000, 73225803422749
7/500000000000000000000000000000) ], [ Vector(-1, -879366755419571
/500000000000000000000000000000), Vector(-2, 4460430333228999/1000
000000000000000000000000000) ] ]
```

`BUHSolve(P, SSys, Gamma, SSol, ConfigSet)`

**Purpose:** Solves a zero-dimensional non-homogeneous polynomial system of equations by user defined homotopy.

**Syntax** `Bertini.BUHSolve(P:LIST, SSys:LIST, Gamma:STRING, SSol:LIST, ConfigSet:LIST):LIST`

**Input** 1st parameter `P`, a list of polynomials. 2nd parameter `SSys`, a list of polynomials which will be used as start system. 3rd parameter `Gamma`, a complex number possibly of magnitude one. 4th parameter `SSol`, a list of solutions of the start system `SSys`. 5th parameter `ConfigSet`, a list of configuration settings.

**Output** A list of lists where each list contains a real solution of the system `P`.

**Example** Consider the polynomial ring $\mathbb{Q}[x, y]$. To solve the polynomial system of equations $x^2 - 1 = 0, y^2 - 1 = 0$, one has to run the following commands in ApCoCoA interactive window:

```
Use QQ[x,y];
P := [x^2-1, y^2-1];
SSys := [x^2, y^2];
Gamma := "1";
SSol:=[[["-1.0","0.0"],["-1.0","0.0"]],[["1.0","0.0"],["1.0","0.0"]]];
ConfigSet := ["USERHOMOTOPY: 1"];
Bertini.BUHSolve(P, SSys, Gamma, SSol, ConfigSet);
```

The output of above commands is the following.

```
[[-250000000000021/250000000000000, -250000000000021/250000000000000],
[250000000000021/250000000000000, 250000000000021/250000000000000]]
```

`Bertini.BPCSolve(P, SysTyp, ConfigSet)`

**Purpose:** Computes numerical irreducible decomposition by finding witness point supersets of a positive dimensional homogeneous or non-homogeneous polynomial systems of equations.

**Syntax** `Bertini.BPCSolve(P:LIST, SysTyp:STRING , ConfigSet:LIST):LIST`

**Input** 1st parameter `P`, a list of polynomials. 2nd parameter `SysTyp`, type of system P. 3rd parameter `ConfigSet`, a list of configuration settings.

**Output** A list of lists containing witness point supersets of the system `P`.

**Example** Consider the polynomial ring $\mathbb{Q}[x, y]$. We want to find the numerical irreducible decomposition of the curve $(29/16)x^3 - 2xy, y - x^2$. We need to run the following commands in ApCoCoA interactive window:

```
Use QQ[x,y];
P := [(29/16)x^3-2xy, y-x^2];
ConfigSet := ["TRACKTYPE: 1", "SHARPENDIGITS: 30"];
Bertini.BPCSolve(P, SysTyp, ConfigSet);
```

The output of above commands is the following.

```
[ [ Vector(-144037547051541/250000000000000000000000000000000000000,
1194673810665003/125000000000000000000000000000000000000), Vector(
```

```
3960923778190771/10000000000000000000000000000000000000000, 2427766
741188941/10000000000000000000000000000000000000000) ] ]
```

Now we describe functions which are available in the package `hom4ps`.

`HSolve(P,HomTyp)`

**Purpose:** Solves a zero-dimensional square homogeneous or non-homogeneous polynomial system of equations.

**Syntax** `Hom.HSolve(P:LIST,HomTyp:INT):LIST`

**Input** 1st parameter `P`, a list of polynomials `P`. 2nd parameter `HomTyp`, set it to 1 for polyhedral homotopy and to 2 for classical linear homotopy.

**Output** A list of lists where each list contains a finite solution.

**Example** Consider the polynomial ring $\mathbb{Q}[x, y]$. We want to solve the system $x^2 + y^2 - 5, xy - 2$. We run the following commands in ApCoCoA interactive window:

```
Use QQ[x,y];
P := [x^2+y^2-5, xy-2];
HomTyp:=1;
Hom.HSolve(P,HomTyp);
```

The output of above commands is the following.

```
[ [ [2, 0], [1, 0]], [[-1, 0], [-2, 0]], [[-2, 0], [-1, 0]],
[[1, 0], [2, 0] ] ]
```

`LRSolve(P,HomTyp)`

**Purpose:** Solves a non-square zero-dimensional homogeneous or non-homogeneous polynomial system of equations.

**Syntax** `Hom.LRSolve(P:LIST, HomTyp:INT):LIST`

**Input** 1st parameter `P`, a list of polynomials `P`. 2nd parameter `HomTyp`, set it to 1 for polyhedral homotopy and to 2 for classical linear homotopy.

**Output** A list of lists where each list contains a finite solution.

**Example** Consider the polynomial ring $\mathbb{Q}[x_1, x_2]$. We want to solve the system $x_1^2-1 = 0, x_1x_2 - 1 = 0, x_1^2 - x_1 = 0$. We run the following commands in ApCoCoA interactive window:

```
Use QQ[x[1..2]];
P := [x[1]^2-1, x[1]x[2]-1,x[1]^2-x[1]];
HomType:= 1;
Hom.LRSolve(P,HomTyp);
```

The output of above commands is the following.

```
[ [ [-9143436298249491/20000000000000000, 9937657539108147/50000000
000000000], [-24282046571107613/50000000000000000, 1864146148586522
9/100000000000000000] ], [ [1, 0], [1, 0] ] ]
```

The following function also does the same job as the function but with a different kind of randomization. For more details we refer to ApCoCoA manual.

`SRSolve(P,HomTyp)`

**Purpose:** Solves a non-square zero-dimensional homogeneous or non-homogeneous polynomial system of equations.

**Syntax** `Hom.SRSolve(P:LIST, HomTyp:INT):LIST`

**Input** 1st parameter `P`, a list of polynomials `P`. 2nd parameter `HomTyp`, set it to 1 for polyhedral homotopy and to 2 for classical linear homotopy.

**Output** A list of lists where each list contains a finite solution.

**Example** Consider the polynomial ring $\mathbb{Q}[x_1, x_2]$. We want to solve the system $x_1^2-1 = 0, x_1x_2 - 1 = 0, x_1^2 - x_1 = 0$. We run the following commands in ApCoCoA interactive window:

```
Use QQ[x[1..2]];
P := [x[1]^2-1, x[1]x[2]-1,x[1]^2-x[1]];
HomType:= 1;
Hom.SRSolve(P,HomTyp);
```

The output of above commands is the following.

```
[[[-51917361941691031/100000000000000000, -1846796377886887/400000000
0000000], [-14765467180940843/10000000000000000, 23807586810196137/10
000000000000000]], [[1, 0], [1, 0]]]
```

Appendix B

# Implementations of Linear Algebra Techniques

In Chapters 3 and 4 we have developed linear algebra algorithms for finding $\mathbb{F}_q$-rational solution of a system of polynomial equations. For experimental results we have implemented these algorithms in CoCoAL. This implementation is available in the package `charP` of ApCoCoA [12]. In the following we provide details about the available functions and explain how to use them.

## B.1 Available Functions

In the following we give a short description of the functions available in the package `charP` for working with the linear algebra algorithms of chapters 3 and 4. This description is also available as a part of the documentation of these functions and can be seen from the help menu of ApCoCoA. Consider the polynomial ring $\mathbb{F}_2[x_1, \ldots, x_4]$. Let $f_1 = 0, \ldots, f_4 = 0$ be a set of polynomials, where

$$\begin{aligned} f_1 &= x_1x_2 + x_2x_3 + x_2x_4 + x_3x_4 + x_1 + x_3 + 1, \\ f_2 &= x_1x_2 + x_1x_3 + x_1x_4 + x_3x_4 + x_2 + x_3 + 1, \\ f_3 &= x_1x_2 + x_1x_3 + x_2x_3 + x_3x_4 + x_1 + x_4 + 1, \\ f_4 &= x_1x_3 + x_2x_3 + x_1x_4 + x_2x_4 + 1. \end{aligned}$$

In the following we solve this system for $\mathbb{F}_2$-rational solutions using different techniques of Chapter 3 which are available through the package `charP`. First we consider functions

which implement the linear algebra techniques of Chapter 3.

`NLASolve(F, Sparse)`

This function provides an implementation of the algorithm of Theorem 3.2.4. It provides the option of choosing between sparse and dense implementations depending on the given system.

**Purpose:** Computes a unique $\mathbb{F}_2$-rational zero of a given polynomial system over $\mathbb{F}_2$.

**Syntax** `CharP.NLASolve(F:LIST, Sparse:BOOL):LIST`

**Input** 1st parameter `F`, a list of polynomials over the field $\mathbb{F}_2$. 2nd parameter `Sparse`, set it to `True` if the system `F` is sparse and to `False` if the system is dense.

**Output** The unique solution of the system `F` in $\mathbb{F}_2^n$.

**Example** Consider the polynomial ring $\mathbb{F}_2[x_1, \ldots, x_4]$. To solve a polynomial system of equations one has to run the following commands in ApCoCoA interactive window:

```
Use ZZ/(2)[x[1..4]];
F:=[
    x[1]x[2] + x[2]x[3] + x[2]x[4] + x[3]x[4] + x[1] + x[3] + 1,
    x[1]x[2] + x[1]x[3] + x[1]x[4] + x[3]x[4] + x[2] + x[3] + 1,
    x[1]x[2] + x[1]x[3] + x[2]x[3] + x[3]x[4] + x[1] + x[4] + 1,
    x[1]x[3] + x[2]x[3] + x[1]x[4] + x[2]x[4] + 1
    ];
Sparse:=True;
CharP.NLASolve(F,Sparse);
```

The output of above commands is the following list containing the unique $\mathbb{F}_2$-rational solution of the above system. Along with the solution, some useful information (such as matrix sizes and the time taken to solve the matrix) will also be displayed on the screen.

```
[0, 1, 0, 1]
```

The working of all the following functions is the same. The only difference between them is that they use different algorithms.

`MNLASolve(F)`

This function provides an implementation of the algorithm of Theorem 3.4.13.

**Purpose:** Computes a unique $\mathbb{F}_2$-rational zero of a given polynomial system over $\mathbb{F}_2$.

**Syntax** `CharP.MNLASolve(F:LIST):LIST`

**Input** A list of polynomials `F` over the field $\mathbb{F}_2$.

**Output** The unique solution of the system `F` in $\mathbb{F}_2^n$.

**Example** Consider the polynomial ring $\mathbb{F}_2[x_1, \ldots, x_4]$. To solve a polynomial system of equations, one has to run the following commands in ApCoCoA interactive window:

```
Use ZZ/(2)[x[1..4]];
F:=[
    x[1]x[2] + x[2]x[3] + x[2]x[4] + x[3]x[4] + x[1] + x[3] + 1,
    x[1]x[2] + x[1]x[3] + x[1]x[4] + x[3]x[4] + x[2] + x[3] + 1,
    x[1]x[2] + x[1]x[3] + x[2]x[3] + x[3]x[4] + x[1] + x[4] + 1,
    x[1]x[3] + x[2]x[3] + x[1]x[4] + x[2]x[4] + 1
    ];
CharP.MNLASolve(F);
```

The output of above commands is the following list containing the unique $\mathbb{F}_2$-rational solution of the above system. Along with the solution, some useful information (such as matrix sizes and time taken to solve the matrix) will also be displayed on the screen.

```
[0, 1, 0, 1]
```

`IMNLASolve(F)`

This function provides an implementation of the algorithm of Theorem 3.5.4.

**Purpose:** Computes a unique $\mathbb{F}_2$-rational zero of a given polynomial system over $\mathbb{F}_2$.

**Syntax** `CharP.IMNLASolve(F:LIST):LIST`

**Input** A list of polynomials `F` over the field $\mathbb{F}_2$.

**Output** The unique solution of the system `F` in $\mathbb{F}_2^n$.

**Example** Consider the polynomial ring $\mathbb{F}_2[x_1, \ldots, x_4]$. To solve a polynomial system of equations, one has to run the following commands in ApCoCoA interactive window:

```
Use ZZ/(2)[x[1..4]];
F:=[
    x[1]x[2] + x[2]x[3] + x[2]x[4] + x[3]x[4] + x[1] + x[3] + 1,
    x[1]x[2] + x[1]x[3] + x[1]x[4] + x[3]x[4] + x[2] + x[3] + 1,
    x[1]x[2] + x[1]x[3] + x[2]x[3] + x[3]x[4] + x[1] + x[4] + 1,
    x[1]x[3] + x[2]x[3] + x[1]x[4] + x[2]x[4] + 1
    ];
CharP.IMNLASolve(F);
```

The output of above commands is the following list containing the unique $\mathbb{F}_2$-rational solution of the above system. Along with the solution, some useful information (such as matrix sizes and time taken to solve the matrix) will also be displayed on the screen.

```
[0, 1, 0, 1]
```

Now we consider functions which implement the algorithms of Chapter 4. The algorithms of Chapter 3 find a unique $\mathbb{F}_q$-rational solution of a system of polynomial equations whereas the algorithms of Chapter 4 computes a Border basis of the given system of polynomial equations. Therefore, the following functions not only find a unique $\mathbb{F}_q$-rational solution but also solve a system of polynomial equations having any number of $\mathbb{F}_q$-rational solutions.

`MBBasisF2(F, NSol)`

This function provides an implementation of the algorithm of Theorem 4.2.5 for computing a Border basis.

**Purpose:** Computes a Border basis of the given system of polynomials.

**Syntax** `CharP.MBBasisF2(F:LIST, NSol:INT):LIST`

**Input** 1st parameter `F`, a list of polynomials over the field $\mathbb{F}_2$. 2nd parameter `NSol`, the number of $\mathbb{F}_2$-rational zeros of the system `F`. If not known in advance then use only first parameter.

**Output** A Border basis of the system `F`.

**Example** Consider the polynomial ring $\mathbb{F}_2[x_1, \ldots, x_4]$. To compute a Border basis of a polynomial system of equations, one has to run the following commands in ApCoCoA interactive window:

```
Use ZZ/(2)[x[1..4]];
F:=[
    x[1]x[2] + x[2]x[3] + x[2]x[4] + x[3]x[4] + x[1] + x[3] + 1,
    x[1]x[2] + x[1]x[3] + x[1]x[4] + x[3]x[4] + x[2] + x[3] + 1,
    x[1]x[2] + x[1]x[3] + x[2]x[3] + x[3]x[4] + x[1] + x[4] + 1,
    x[1]x[3] + x[2]x[3] + x[1]x[4] + x[2]x[4] + 1
    ];
NSol:=1;
CharP.MBBasisF2(F,NSol);
```

The output of above commands is the following list that gives a Border basis of the system `F`. Along with a Border basis, some useful information (such as matrix sizes and time taken to solve the matrix) will also be displayed on the screen.

```
[ x[4] + 1, x[3], x[2] + 1, x[1] ]
```

`IMBBasisF2(F, NSol)`

This function provides an implementation of the algorithm of Theorem 4.2.11 for computing a Border basis.

**Purpose:** Computes a Border basis of the given system of polynomials.

**Syntax** `CharP.IMBBasisF2(F:LIST, NSol:INT):LIST`

**Input** 1st parameter `F`, a list of polynomials over the field $\mathbb{F}_2$. 2nd parameter `NSol`, the number of $\mathbb{F}_2$-rational zeros of the system `F`. If not known in advance then use only first parameter.

**Output** A Border basis of the system `F`.

**Example** Consider the polynomial ring $\mathbb{F}_2[x_1, \ldots, x_4]$. To compute a Border basis of a polynomial system of equations, one has to run the following commands in ApCoCoA interactive window:

```
Use Z/(2)[x[1..4]];
F:=[
    x[2]x[3] + x[1]x[4] + x[2]x[4] + x[3]x[4] + x[1] + x[2] + x[3] + x[4],
    x[2]x[3] + x[2]x[4] + x[3]x[4] + x[2] + x[3] + x[4],
    x[1]x[2] + x[2]x[3] + x[2]x[4] + x[3]x[4] + x[1] + x[2],
    x[1]x[2] + x[2]x[3] + x[2]x[4] + x[3]x[4] + x[1] + x[2]
   ];
NSol:=3;
CharP.IMBBasisF2(F,NSol);
```

The output of above commands is the following list that gives a Border basis of the system `F`. Along with a Border basis, some useful information (such as matrix sizes and time taken to solve the matrix) will also be displayed on the screen.

```
[x[3]x[4] + x[4], x[1]x[4] + x[1], x[1]x[3] + x[1], x[1]x[2] + x[1],
x[2]x[3]x[4] + x[4], x[1]x[2]x[4] + x[1]]
```

Appendix C

# Implementations of Techniques Using MILP

The GLPK (GNU Linear Programming Kit) package is intended for solving large-scale linear programming (LP), mixed integer programming (MIP), and other related problems. It is a set of routines written in ANSI C and organized in the form of a callable library. CPLEX by ILOG [102] is the commercial linear optimization tool. We use CPLEX version 12.2. Moreover, note that CPLEX can be parallelized.

Recall that in Chapter 5 we have studied recent suggestions of transferring the problem of solving a system of polynomial equations over $\mathbb{F}_2$ into a mixed integer linear programming problem. In particular, we have developed several conversion algorithms and strategies for converting the polynomial system over $\mathbb{F}_2$ to a polynomial system over $\mathbb{R}$ (respectively over $\mathbb{Z}$). The conversion algorithms are implemented in CoCoAL and these implementations are available in the package `glpk` of ApCoCoA [12]. The solution process of Chapter 5 consists of two stages: applying a conversion algorithm to prepare a MILP problem and then using an IP solver to solve the MILP problem. Our research is focused on conversion algorithms and conversion strategies. After applying a conversion algorithm, the problem can be solved using any IP solver. We have used GLPK and CPLEX to solve the IP problems. Due to the reason that CPLEX is a commercial solver, the conversion algorithms are equipped with GLPK solver only. But the code in the package `glpk` can be easily adjusted for using CPLEX as an IP solver and is available from author on request. In the following we provide details about the available functions and explain how to use them.

## C.1 Available Functions

In the following we give a short description of the functions available in the package `glpk` for conversion algorithms and using GLPK as IP solver. This description is also available as a part of the documentation of this package and can be seen from the help menu of ApCoCoA.

`IPCSolve(F, QStrategy, CStrategy, MinMax)`

This function provides an implementation of the conversion algorithms of Propositions 5.2.2 and 5.5.1. It provides the option of choosing between different conversion strategies studied in Chapter 5.

**Purpose:** Solves a system of polynomial equations for one solution in $\mathbb{F}_2^n$.

**Syntax** `GLPK.IPCSolve(F:LIST,QStrategy:INT,CStrategy:INT,MinMax:STRING):LIST`

**Input** 1st parameter `F`, a list of polynomials over the field $\mathbb{F}_2$. 2nd parameter `QStrategy`, strategy for quadratic substitution: 0 − Standard, 1 − Linear Partner, 2 − Double Linear Partner and 3 − Quadratic Partner. 3rd parameter `CStrategy`, strategy for cubic substitution: 0 − Standard, and 1 − Quadratic Partner. 4th parameter `MinMax`, optimization direction, i.e. minimization (`"Min"`) or maximization (`"Max"`).

**Output** A solution of the system `F` in $\mathbb{F}_2^n$.

**Example** Consider the polynomial ring $\mathbb{F}_2[x_1, \ldots, x_3]$. To solve a given polynomial system of equations one has to run the following commands in ApCoCoA interactive window:

```
Use ZZ/(2)[x[1..3]];
F := [ x[1]x[2]x[3] + x[1]x[2] + x[2]x[3] + x[1] + x[3] +1,
       x[1]x[2]x[3] + x[1]x[2] + x[2]x[3] + x[1] + x[2],
       x[1]x[2] + x[2]x[3] + x[2]
     ];
QStrategy:=0;
CStrategy:=1;
MinMax:="Max";
GLPK.IPCSolve(F, QStrategy, CStrategy, MinMax);
```

The result will be the following, where the return value is the list which provides a solution.

```
Modelling the system as a mixed integer programming problem.
QStrategy: Standard, CStrategy: CubicParnterDegree2.
Model is ready to solve with GLPK...
Solution Status: INTEGER OPTIMAL
Value of objective function: 1
[0, 0, 1]
```

Note that `QStrategy` and `CStrategy` deal with quadratic and cubic terms respectively. For instance, if there are no cubic terms in the support of polynomials, the `CStrategy` will have no effect on timings. The working of all the following functions is the same. The only thing we to ensure is the choice of a strategy. Different strategies lead to different timings as reported in Chapter 5

`RPCSolve(F, QStrategy, CStrategy, MinMax)`

This function provides an implementation of the conversion algorithms of Propositions 5.2.8 and 5.5.2. It provides the option of choosing between different conversion strategies studied in Chapter 5.

**Purpose:** Solves a system of polynomial equations for one solution in $\mathbb{F}_2^n$.

**Syntax** `GLPK.RPCSolve(F:LIST,QStrategy:INT,CStrategy:INT,MinMax:STRING):LIST`

**Input** 1st parameter `F`, a list of polynomials over the field $\mathbb{F}_2$. 2nd parameter `QStrategy`, strategy for quadratic substitution: 0 − Standard, 1 − Linear Partner, 2 − Double Linear Partner and 3 − Quadratic Partner. 3rd parameter `CStrategy`, strategy for cubic substitution: 0 − Standard, and 1 − Quadratic Partner. 4th parameter `MinMax`, optimization direction, i.e. minimization (`"Min"`) or maximization (`"Max"`).

**Output** A solution of the system `F` in $\mathbb{F}_2^n$.

**Example** Consider the polynomial ring $\mathbb{F}_2[x_1, \ldots, x_3]$. To solve a given polynomial system of equations one has to run the following commands in ApCoCoA interactive window:

```
Use ZZ/(2)[x[1..3]];
F := [ x[1]x[2]x[3] + x[1]x[2] + x[2]x[3] + x[1] + x[3] +1,
       x[1]x[2]x[3] + x[1]x[2] + x[2]x[3] + x[1] + x[2],
       x[1]x[2] + x[2]x[3] + x[2]
     ];
QStrategy:=1;
CStrategy:=0;
MinMax:="Max";
GLPK.RPCSolve(F, QStrategy, CStrategy, MinMax);
```

The result will be the following, where the return value is a list which provides a solution.

```
Modelling the system as a mixed integer programming problem.
QStrategy: Standard, CStrategy: CubicParnterDegree2.
Model is ready to solve with GLPK...
Solution Status: INTEGER OPTIMAL
Value of objective function: 1
[0, 0, 1]
```

`L01PSolve(F, QStrategy, CStrategy, MinMax)`

This function provides an implementation of the conversion algorithm of Proposition 5.4.7. It provides the option of choosing between different conversion strategies developed in Section 5.4.1.

**Purpose:** Solves a system of polynomial equations over $\mathbb{F}_2$ for one solution in $\mathbb{F}_2^n$.

**Syntax** `GLPK.L01PSolve(F:LIST,QStrategy:INT,CStrategy:INT,MinMax:STRING):LIST`

**Input** 1st parameter `F`, a list of polynomials over the field $\mathbb{F}_2$. 2nd parameter `QStrategy`, strategy for quadratic substitution: 0 − Standard, 1 − Linear Partner, 2 − Double Linear Partner and 3 − Quadratic Partner. 3rd parameter `CStrategy`, strategy for cubic substitution: 0 − Standard, and 1 − Quadratic Partner. 4th parameter `MinMax`, optimization direction, i.e. minimization (`"Min"`) or maximization (`"Max"`).

**Output** A solution of the system `F` in $\mathbb{F}_2^n$.

**Example** Consider the polynomial ring $\mathbb{F}_2[x_1, \ldots, x_3]$. To solve a given polynomial system of equations one has to run the following commands in ApCoCoA interactive window:

```
Use ZZ/(2)[x[1..3]];
F := [ x[1]x[2]x[3] + x[1]x[2] + x[2]x[3] + x[1] + x[3] +1,
       x[1]x[2]x[3] + x[1]x[2] + x[2]x[3] + x[1] + x[2],
       x[1]x[2] + x[2]x[3] + x[2]
     ];
QStrategy:=1;
CStrategy:=0;
MinMax:="Max";
GLPK.L01PSolve(F, QStrategy, CStrategy, MinMax);
```

The result will be the following, where the return value is a list which provides a solution.

```
Modelling the system as a mixed integer programming problem.
QStrategy: Standard, CStrategy: CubicParnterDegree2.
Model is ready to solve with GLPK...
Solution Status: INTEGER OPTIMAL
Value of objective function: 1
[0, 0, 1]
```

implementation of rules for IPC and RPC is still left to document

# Appendix D

# Some Miscellaneous Implementations

Recall that in Chapters 6 and 7 we have studied techniques using MINLP, number theory and numerical analysis. Essentially, we studied techniques of transferring the problem of solving a system of polynomial equations over $\mathbb{F}_2$ into a MINLP problem, a number theory problem or a numerical analysis problem. In the above-mentioned chapters we developed several conversion algorithms which are implemented in CoCoAL and are available from the author upon request. Unfortunately, we could not make them available as packages of ApCoCoA [12]. The reasons are given later in this appendix.

The solution processes of Chapters 6 and 7 consist of two stages: applying a conversion algorithm to model a MINLP problem, a number theory problem or a numerical analysis problem, and then using an IP solver, a number theory package or a numerical solver for solving the modeled problem. Our research is focused on conversion algorithms and conversion strategies. In the following we explain how we achieved the implementation of these tasks one by one.

## D.1 Implementations of Techniques Using MINLP

In Section 6.1 we developed techniques using MINLP. The conversion algorithms of this section are implemented in CoCoAL and are available from author upon request. Given a system of polynomial equations $F$ over $\mathbb{F}_2$. After applying the conversion algorithm, the next stage is to use a nonlinear IP solver for the solution of the modeled MINLP problem as explained in Section 6.1.

Our modeled MINLP problem is a non-convex problem. There are three software packages that can solve non-convex MINLP problems to proven optimality, using branch-and-reduce techniques, namely: BARON, Alpha-BB [81] and COUENNE [56]. Unfortunately, the first two are the commercial softwares. Therefore, for solving MINLP problems we use the open-source solver COUENNE [56]. COUENNE (Convex Over and Under ENvelopes for Nonlinear Estimation) is a spatial branch-and-bound algorithm to solve MINLP problems. COUENNE aims at finding global optima of non-convex MINLP problems. It implements linearization, bound reduction, and branching methods within a branch-and-bound framework. Note that some packages for convex MINLP problem can be used to find heuristic solutions for non-convex MINLP, namely BONMIN, DICOPT and LaGO [81]. Finally, GloptiPoly [81] can solve general polynomial optimization problems. Note that solvers for convex MINLP problems can be used on non-convex problems as heuristics, as they may provide a feasible solution.

COUENNE is a software package that can be used only in combination with a modeling language software package such as AMPL [10]. AMPL is a comprehensive and powerful algebraic modeling language for linear and nonlinear optimization problems, in discrete or continuous variables. COUENNE is a software package for which interface to AMPL is available. To the best of authors knowledge this is the only way to use COUENNE because it accepts as input only AMPL stub (.nl) file. Such files can be generated from AMPL with the command "write gfilename;" and then used as input files for COUENNE. Further information on using COUENNE and AMPL can be found at the web sites [56, 10]. A more convenient way to use COUENNE is to install an interface to AMPL as described at AMPL web page [10].

Given a system of polynomial equations $F$ over $\mathbb{F}_2$. We can solve this system as follows. Apply the conversion algorithm and prepare an input file for AMPL containing the modeled MINLP problem. Now use the COUENNE interface to AMPL to solve the nonlinear problem. Now the solution of MINLP problem provided by AMPL can be easily translated into a solution of the given system of polynomial equations $F$ over $\mathbb{F}_2$. If COUENNE interface to AMPL is not installed then the process of solving the system may consist of the following steps.

1) Use CoCoAL code (which implements the conversion algorithm and prepares an input file) to generate an input file say `Finpu.mod` for AMPL which models $F$ as a MINLP problem.

2) Use AMPL to prepare a stub (.nl) file `Finput.nl` which is used as an input file for COUENNE. If you are running AMPL at the command line then you need

to execute the following commands for this purpose.

```
model Finput.mod
write gFinput;
```

This will create a file `Finput.nl` which can be used as a input file for COUENNE.

3) Use COUENNE to solve the MINLP model given by the input file `Finput.nl`. If you are in the directory where COUENNE.exe resides then you need to execute the command

```
./couenne Finput.nl
```

at the command line. This will solve the model and create a solution file called `Finput.sol` in the same directory.

4) Use AMPL to read the solution file `Finput.sol`. If you are running AMPL at the command line then you need to execute the following commands for this purpose.

```
model Finput.mod
solution Finput.sol;
```

Now by issuing the command `display` the values of the indeterminates (optimal solution) can be obtained. For instance, `display x1;` will display the value of the indeterminate $x_1$ to the screen.

The solution process explained above and the (commercial) availability of software packages clearly shows that why it was not possible making these implementations available as a package of ApCoCoA. As a final remark note that the commercial non-linear IP solvers such as BARON, Alpha-BB [81] can provide a sufficient speed up to our technique.

## D.2 Implementations of Techniques Using Linear Diophantine Equations

In Section 6.1 we developed techniques using number theory. The conversion algorithms of this section are implemented in CoCoAL and are available from author upon request.

Given a system of polynomial equations $F$ over $\mathbb{F}_2$. After applying the conversion algorithm, the next stage is to solve a system of linear Diophantine equations for a (unique) positive solution.

The positive solution of a system of linear Diophantine equations is achieved in two steps. Firstly, a general integer solution is obtained. Secondly, a mixed integer linear programming problem is solved for finding a non-negative integer solution. As described in Section 6.1.1 we use the Smith normal form for resolving the first step. We use the computer algebra system PARI/GP [149] which is designed for fast computations in number theory, for computing the Smith normal form. To resolve the second step we use the commercial linear optimization tool CPLEX by ILOG [102].

Given a system of polynomial equations $F$ over $\mathbb{F}_2$, and assume that PARI/GP and CPLEX are installed on your computer. Then the implementation of the process of solving, given by the algorithm of Proposition 6.2.2 consists of the following steps.

1) Use CoCoAL code (which implements the conversion algorithm and prepares an input file) to generate an input file say `Finput.gp` for PARI/GP. This input file must contain the matrix $\mathcal{M}$ of the homogeneous part of the system of linear Diophantine equations and a PARI/GP command to compute the Smith normal form of $\mathcal{M}$. For the matrix $\mathcal{M}$ the contents of this file will look like `matsnf(`$\mathcal{M}$`,flag =1);`. For more details about calculating the Smith normal form using PARI/GP we refer to PARI/GP manual [149].

2) Use PARI/GP to compute and obtain the Smith normal form on a file say `Foutput`. The output on the file `Foutput` will consist of three matrices $[\mathcal{U}, \mathcal{V}, \mathcal{D}]$, where $\mathcal{U}$ and $\mathcal{V}$ are two unimodular matrices such that $\mathcal{UMV}$ is the diagonal matrix $\mathcal{D}$. Since $\mathcal{M}$ is not a square matrix, $\mathcal{D}$ will be a square diagonal matrix padded with zeros on the left or the top. If you are running PARI/GP at the command line then you need to execute the following commands for this purpose.

```
\r Finput.gp
\w Foutput;
```

The first command reads the file `Finput.gp`. The second command writes the Smith normal form into a file named `Foutput`. Note that if the size of the matrix $\mathcal{M}$ is big then you may need to adjust the stack memory of PARI/GP. This can be achieved by the command `allocatemem();`. For more details refer to PARI/GP manual.

3) Use CoCoAL code (which implements the conversion algorithm, prepares an input file and reads the Smith normal form from the file `Foutput`), and perform the remaining steps of the algorithm of Proposition 6.2.2.

4) Now the algorithm finds a general solution of the system of linear Diophantine equations and formulates a MILP problem having a (unique) positive (minimal) solution of the system of linear Diophantine equations.

5) Finally, the MILP problem can be solved as explained in Appendix A. The optimal solution of the MILP problem is actually a (unique minimal) solution of the system of linear Diophantine equations which can be easily translated into the solution of the system $F$.

## D.3 Implementations of Techniques Using Numerical Analysis

In Chapter 7 we developed techniques using numerical analysis. The conversion algorithms of this chapter are implemented in CoCoAL and are available from author upon request. Given a system of polynomial equations $F$ over $\mathbb{F}_2$. After applying the conversion algorithm, the next stage is to solve a system of nonlinear polynomial equations for a real solution.

Mainly we used two numerical approaches for solving a system of nonlinear polynomial equations for a real solution. The first one is based on homotopy continuation methods and the second one is based on variants of Newton's method. For using homotopy continuation methods in our algebraic settings we have developed two ApCoCoA packages `bertini` and `hom4ps`. These packages call the full artillery of Bertini and HOM4PS for computations with homotopy continuation methods inside ApCoCoA. Using these packages we can write a CoCoAL code, which implements the conversion algorithms of Chapter 7 and solves the resulted system.

One advantage of using Newton methods is that their well-studied and efficient implementations in mathematical software MatLab are available. Therefore, we decided to use MatLab for solving a system of nonlinear polynomial equations for a real solution. The implementations of these techniques consist of two steps. Firstly, use CoCoAL code which implements the conversion algorithm and writes the resulting system of nonlinear equations on a file called `mfile`. Recall that a mfile is used as an input file for MatLab. For details refer to MatLab manual. Secondly, use MatLab to

solve the system on mfile using a variant of Newton's method. The algorithms that we have used are available via the `lsqnonlin(...)` and the `fsolve(...)` commands of MatLab.

# Bibliography


[1] J. Abbott, C. Fassino and M.-L. Torrente, Stable border bases for ideals of points, J. Symbolic Computation 43 (2008), 883-894.

[2] W.P. Adams and H.D. Sherali, A tight linearization and an algorithm for zero-one quadraic programming problems, Management Science 32 (1986), 1274-1290.

[3] M. Albrecht, Algebraic attacks on the Courtois Toy Cipher, diploma thesis, Universität Bremen 2006.

[4] M. Albrecht and G. Bard, M4RI - Linear Algebra over GF(2), available at `http://www.m4ri.sagemath.org/index.html`, 2008.

[5] B. Alidaee, G. Kochenberger and A. Ahmadian, 0-1 quadratic programming approach for the optimal solution of two scheduling problems, International Journal of Systems Science 25 (1994), 401-408.

[6] R. Allen and K. Kennedy, Automatic translation of a FORTRAN program to a vector form, ACM Trans. on Programming Languages and Systems 9 (1987), 491-542.

[7] N. Alon, Combinatorial Nullstellensatz, Combinatorics, Probability and Computing 8 (1999), 7-29.

[8] N. Alon, M.B. Nathanson and I.Z. Ruzsa, The polynomial method and restricted sums of congruence classes, J. Number Theory 56 (1996), 404-417.

[9] N. Alon and M. Tarsi, Colorings and orientations of graphs, Combinatorica 12 (1992), 7-29.

[10] AMPL: A Modeling Language for Mathematical Programming, available at `http://www.ampl.com`.

[11] K.M. Anstreicher and N.W. Brixius, A new bound for the quadratic assignment problem based on convex quadratic programming, Mathematical Programming 89 (2001), 341-357.

[12] The ApCoCoA Team, ApCoCoA: Approximate Computations in Commutative Algebra, available at `http://www.apcocoa.org`.

[13] W. Auzinger and H.J. Stetter, An elimination algorithm for the computation of all zeros of a system of multivariate polynomial equations, In Proceedings of the International Conference on Numerical Mathematics, National University of Singapore, May 31-June 4, 1988, Birkhuser, pp. 11-30.

[14] D. Avais and K. Fukuda, A pivioting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra, Discrete Comput. Geom. 8 (1992), 295-313.

[15] T. Aykin, On a quadratic integer program for the location of interacting hub facilities, European Journal of Operational Research 46 (1990), 409-411.

[16] F. Baader and J. Ziekmann, Handbook of logic in artificial intelligence and logic programming, Oxford University Press, 1994.

[17] E. Balas, Extension de l'algorithme additif à la programmation en nombres entiers et à la programmation non linéaire, C. R. Acad. Sc. Paris, May 1964.

[18] G. Bard, N. Courtois and C. Jafferson, Efficient methods for conversion and solution of sparse systems of low-degree multivariate polynomials over GF(2) via SAT-Solvers, Cryptology ePrint Archive 2007(24), 2007.

[19] G. Bard, Algebraic cryptanalysis, Springer-Verlag, 2009.

[20] M. Bardet, J.-C. Faugère and B. Salvy, On the complexity of Gröbner basis computation of semi-regular overdetermined algebraic equaitons, In Proc. ICPSS International Conference on Polynomial System Solving Paris, November 2004.

[21] D.J. Bates, J.D. Hauenstein, A.J. Sommese and C.W. Wampler, Bertini: software for numerical algebraic geometry, available at `http://www.nd.edu/$\sim$sommese/bertini`.

[22] D.J. Bates and F. Sottile, Khovanskii-Rolle continuation for real solutions, preprint 2009.

[23] D.A. Bayer, The division algorithm and the Hilbert scheme, Ph.D. thesis, Harvard University, 1982.

[24] R. Beigel, The polynomial method in circuit complexity, Structure in Complexity Theory Conference, 1993, pp. 82-95.

[25] D.J. Bernstein, J. Buchmann, and E. Dahmen, Post quantum cryptography, Springer, 2008.

[26] A. Billionnet and S. Elloumi, Using a mixed integer quadratic programming solver for the unconstrained quadratic 0-1 problem, Technical Report CEDRIC 466, available at `http://cedric.cnam.fr/PUBLIS/RC466.pdf`, 2003.

[27] A. Billionnet, S. Elloumi and A. Lambert, Extending the QCR method to general mixed-integer programs, Math. Program. 2010, to appear.

[28] A. Billionnet, S. Elloumi and M.-C. Plateau, Improving the performance of standard solvers for quadratic 0-1 programs by a tight convex reformulation: the QCR method, Discr. Appl. Math. 157 (2009), 1185-1197.

[29] A. Billionnet and E. Soutif, An exact method based on lagrangian decomposition for the 0-1 quadratic knapsack problem, European Journal of Operational Research 157 (2004), 565-575.

[30] M. Borges-Quintana, M.A. Borges-Trenard, and E. Martnez-Moro, An application of Mllers algorithm to coding theory, in: M. Sala, T. Mora, L. Perret, S. Sakata and C. Traverso (eds.), Gröbner Bases, Coding, and Cryptography, Springer, 2009, pp. 379-384.

[31] J. Borghoff, L.R. Knudsen and M. Stolpe, Bivium as a Mixed-Integer linear programming problem, in: M. G. Parker (ed.), Cryptography and Coding, LNCS 5921, Springer Verlag, Berlin 2009, pp. 133-152.

[32] W. Bosma, J. Cannon, and C. Playoust, The Magma algebra system I: the user language, J. Symbolic Computation 24 (1997), 235-265.

[33] W.D. Brownawell, Bounds for the degrees in Nullstellensatz, Ann. of Math. 126 (1987), 577-592.

[34] B. Buchberger, Ein Algorithmus zum Auffinden der Basiselemente des Restklassenrings nach einem nulldimensionalen Polynomideal, Ph.D. thesis, Universität Innsbruck 1965.

[35] B. Buchberger, G. Collins and R. Loos (eds.), Computer algebra: symbolic and algebraic computations, Mir, Moscow, 1986.

[36] J. Buchmann, D. Cabarcas, J. Ding and M.S.E. Mohamed, Flexible partial enlargement to accelerate Gröbner basis computation over $\mathbb{F}_2$, In Proceedings of the 3rd Africacrypt Conference (Africacrypt 2010), LNCS, Springer-Verlag, Berlin, 2010, pp. 69-81.

[37] J. Buchmann and J. Ding, M.S.E. Mohamed and W.S.A Mohamed, MutantXL: solving multivariate polynomial equations for cryptanalysis, in: H. Handschuh, S. Lucks, B. Preneel and P. Rogaway (eds.), Symmetric Cryptography, Dagstuhl Seminar Proceedings, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany 2009.

[38] R. Burden and J.D. Faires, Numerical analysis, 7th ed., PWS-Kent Publ. Co., 2000.

[39] S. Burer, R.D.C. Monteiro and Y. Zhang, Rank-two relaxation heuristics for MAX-CUT and other binary quadratic programs, SIAM Journal on Optimization 12 (2001), 503-521.

[40] S. Buss and T. Pitassi, Good degree bounds on Nullstellensatz refutations of the induction principle, IEEE Conference on Computational Complexity (1996), 233-242.

[41] R.H. Byrd, E.H. Mary and J. Nocedal, An interior point algorithm for large-scale nonlinear programming, SIAM Journal on Optimization 9 (1999), 877-900.

[42] R.H. Byrd, J.C. Gilbert and J. Nocedal, A trust region method based on interior point techniques for nonlinear programming, Mathematical Programming 89 (2000), 149-185.

[43] P. Carraresi and F. Malucelli, A new lower bound for the quadratic assignment problem problem, Operations Research 40 (1992), S22-S27.

[44] M.W. Carter, The indefinite zero-one quadratic problem, Discrete Applied Mathematics 7 (1984), 23-44.

[45] W. Chaovalitwongse, P.M. Pardalos, L.D. Iasemidis, D.S. Shiau and J.C. Sackellares, Dynamical approaches and multi-quadratic integer programming for seizure prediction, Optimization Methods and Software 20 (2005), 389-400.

[46] W. Chaovalitwongse, P. M. Pardalos and O. A. Prokopyev, A new linearization technique for multi-quadratic 0-1 programming problems, Operations Research Letters 32 (2004), 517-522.

[47] P. Chardaire and A. Sutter, A decomposition method for quadratic zero-one programming, Management Science 41 (1995), 704-712.

[48] B. Chen, Strategies on algebraic attacks using SAT solvers, in: 9th Int. Conf. for Young Computer Scientists, IEEE Press, 2008.

[49] T.W.J. Chou and G.E. Collins, Algorithms for the solutions of linear Diophantine equations, SIAM Journal of Computing 11 (1982), 687-708.

[50] C. Cid, S. Murphy and M. Robshaw, Small scale variants of the AES, In: Fast Software Encryption: 12th International Workshop, Springer Verlag, Heidelberg 2005, pp. 145-162.

[51] The CoCoA Team, CoCoA: a system for doing Computations in Commutative Algebra, available at http://cocoa.dima.unige.it.

[52] T.F. Coleman and Y. Li, On the convergence of reflective Newton methods for large-scale nonlinear minimization subject to bounds, Mathematical Programming 67 (1994), 189-224.

[53] T.F. Coleman and Y. Li, An interior trust region approach for nonlinear minimization subject to bounds, SIAM Journal on Optimization 6 (1996), 418-445.

[54] S.A. Cook, The complexity of theorem-proving procedures, Proceedings of the Third Annual ACM Symposium on the Theory of Computing (Shaker Heights, Ohio 1971), ACM, New York, 1971, pp. 151-158.

[55] D. Coppersmith, Solving homogeneous linear equations over $\mathbb{F}_2$ via Block Wiedemann algorithm, Mathematics of Computation 62 (1994), 333-350.

[56] Couenne: Convex Over and Under ENvelopes for Nonlinear Estimation, available at `https://projects.coin-or.org/Couenne`.

[57] N.T. Courtois, How fast can be algebraic attacks on block ciphers, in: E. Biham, H. Handschuh, S. Lucks, and V. Rijmen (eds.), Symmetric Cryptography (Dagstuhl 2007), Dagstuhl Sem. Proc. 7021, available at `http://drops.dagstuhl.de/opus/volltexte/2007/1013`.

[58] N. Courtois, L. Goubin, W. Meier and J.-D. Tacier, Solving underdefined systems of multivariate quadratic equations, in: D. Naccache and P.Paillier (eds.), In Public Key Cryptography PKC 2002, LNCS 2274, Springer, 2002, pp. 211227.

[59] N.T. Courtois, A. Klimov, J. Patarin and A Shamir, Efficient algorithm for solving overdetermined systems of multivariate polynomial equations, in: B. Preneel (ed.), Advances in Cryptology-EUROCRYPT 2000, LNCS 1807, Springer-Verlag, Berlin 2000, pp. 392-407.

[60] Y. Crama, P. Hansen and B. Jaumard, The basic algorithm for pseudo-boolean programming revisited, Discrete Applied Mathematics 29 (1990), 171-185.

[61] J.A. De Loera, J. Lee, P. Malkin and S. Margulies, Computing infeasibility certificates for combinatorial problems through Hilberts Nullstellensatz, J. Symbolic Computation 46 (2011), 12601283.

[62] J.A. De Loera, Gröbner bases and graph colorings, Beitrage zur Algebra and Geometrie 36 (1995), 89-96.

[63] J.A. De Loera, J. Lee, S. Margulies and S. Onn, Expressing combinatorial problems by systems of polynomial equations and Hilbert's Nullstellensatz, Combinatorics, Probability and Computing 18 (2009), 551-582.

[64] J.A. De Loera, P.N. Malkin and P.A. Parrilo, Computation with polynomial equations and inequalities arising in combinatorial optimization, preprint, 2009.

[65] P. Deuflhard, Newton methods for nonlinear problems, Springer, Berlin 2004.

[66] A. Dickenstein and I.Z. Emiris, Solving polynomial equations, Springer, Berlin 2005.

[67] J. Ding, Mutants and its impact on polynomial solving strategies and algorithms, Privately distributed research note, University of Cincinnati and Technical University of Darmstadt 2006.

[68] J. Ding, J. Buchmann, M.S.E. Mohamed, W.S.A. Mohamed and R.-P. Weinmann, MutantXL, in: J.-C. Faugère and L. Perret (eds.), Proc. Conf. Symbolic Computation and Cryptoghraphy (Beijing 2008), Math. in Comp. Science, Birkhäuser 2009.

[69] J. Ding, J. E. Gower and D. Schmidt, Multivariate public key cryptosystems, Springer, New York 2006.

[70] I.S. Duff, A.M. Erismann and J.K. Ried, Direct methods for sparse matrices, Oxford Science Publication, 1986.

[71] J.-G. Dumas, B.D. Saunders and G. Villard, On efficient sparse integer matrix Smith normal form computations, J. Symbolic Computation 32 (2001), 71-99.

[72] S. Eliahou, An algebraic criterion for a graph to be four-colourable, Aportaciones Matemáticas, Soc. Matemática Mexicana, Notas de Investigacion 6 (1992), 3-27.

[73] J.-C. Faugère, A new efficient algorithm for computing Gröbner basis (F4), J. Pure Appl. Alg. 139 (1999), 61-88.

[74] J.-C. Faugère, A new efficient algorithm for computing Gröbner basis without reduction to zero (F5), in: Symbolic and algebraic computation, Proc. Int. Symp. ISSAC 2002, ACM Press, New York 2002.

[75] J.-C. Faugère and S. Lachartre, Parallel Gaußian elimination for Gröbner bases computations in finite fields, ACM proceedings of The International Workshop on Parallel and Symbolic Computation (PASCO), 2010, pp.1-10.

[76] K.G. Fischer, Symmetric polynomials and Hall's theorem, Discrete Math. 69 (1988), 225-234.

[77] A. Faye and F. Roupin, Partial lagrangian and semidefinite relaxations of quadratics programs, In 6èeme congrèes ROADEF, Tours, 14-16 féevrier, 2005.

[78] R. Fortet. L'algebre de boole et ses applications en recherche operationnelle. Cahiers du Centre d'Etudes de Recheche Operationnelle 1 (1959), 5-36.

[79] A.S. Fraenkel and Y. Yesha, Complexity of problems in games, graphs and algebraic equations, Discrete Applied Mathematics 1 (1979), 1530.

[80] L. Galli and A.N. Letchford, Reformulating mixed-integer quadratically constrained quadratic programs, preprint.

[81] GAMS Development Corp., available at `http://www.gams.com/solvers/index.html`.

[82] M.R. Garey and D.S. Johnson, Computers and intractability: a guide to the theory of NP-completeness, W.H. Freeman and Company, 1979.

[83] M.W. Giesbrecht, Probabilistic computation of the Smith normal form of a sparse integer matrix, LNCS 1122, pp. 173-186.

[84] GNU Linear Programming Kit, available at `http://www.gnu.org/software/glpk/glpk.html`.

[85] F. Glover, Improved linear integer programming formulations of nonlinear integer problems, Management Science 22 (1975), 455-460.

[86] F. Glover and E. Woolsey, Further reduction of zero-one polynomial programming problems to zero-one linear programming problems, Operations Research 21 (1973), 156-161.

[87] F. Glover and E. Woolsey, Converting the 0-1 polynomial programming problem to a 0-1 linear program, Operations Research 22 (1974), 180-182.

[88] A.J. Goldman, Linearization in 0-1 variables: a clarification, Operations Research 31 (1983), 946-947.

[89] R.N. Greenwell and S. Kertzner, Solving linear Diophantine matrix equations using the Smith normal form (more or less), preprint 2009.

[90] J.L. Hafner and K.S. Macurley, Asymptotically fast triangularization of matrices over rings, SIAM Journal of Computing 20 (1991), 1068-1083.

[91] P.L. Hammer and A.A. Rubin, Some remarks on quadratic programming with 0-1 variables, RAIRO 3 (1970), 67-79.

[92] P.L. Hammer and S. Rudeanu, Boolean methods in operations research, Springer, Berlin 1968.

[93] P. Hansen, Nonlinear 0-1 programming by implicit enumeration, 7th Mathematical Programming Symposium, The Hague, Sept. 1971.

[94] J. Håstad, S. Philips, and S. Safra, A Well-Characterized approximation problem, Information Processing Letters 47 (1993), 301-305.

[95] D. Heldt, M. Kreuzer, S. Pokutta and H. Poulisse, Approximate computation of zero-dimensional polynomial ideals, J. Symbolic Computation 44 (2009), 1566-1591.

[96] C. Helmberg and F. Rendl, Solving quadratic (0,1)-problems by semidefinite programs and cutting planes, Math. Programming 8 (1998), 291-315.

[97] M.P. Helme and T.L. Magnanti, Designing satellite communication networks by zero-one quadratic programming, Networks 19 (1989), 427-450.

[98] C.J. Hillar and T. Windfeldt, An algebraic characterization of uniqely vertex colorable graphs, J. Combinatorial Theory, Series B, 98 (2008), 400-414.

[99] L.D. Iasemidis, P.M. Pardalos, J.C. Sackellares and D.-S. Shiau, Quadratic binary programming and dynamical system approach to the predictability of epileptic seizures, J. Combinatorial Optimization 5 (2001), 9-26.

[100] C.S Iliopoulos, Worst-cast complexity bounds on algorithms for computing the canonical structure of finite abelian groups and the Hermite and Smith normal forms of an integer matrix, SIAM Journal of Computing 18 (1989), 658-669.

[101] C.S Iliopoulos, Worst-cast complexity bounds on algorithms for computing the canonical structure of infinite abelian groups and solving systems of linear Diophantine equations, SIAM Journal of Computing 18 (1989), 670-678.

[102] ILOG CPLEX, available at `http://www.ilog.com/products/cplex/`.

[103] P. Impagliazzo, P. Pudlák and J. Sgall, Lower bounds for polynomial calculus and the Gröbner basis algorithm, Computational Complexity 8 (1999), 127-144.

[104] R.G. Jeroslow, There cannot be any algorithm for integer programming with quadratic constraints, Operations Research 21 (1973), 221-224.

[105] D.R. Jones, C.D. Perttunen and B.E. Stuckman, Lipschitzian optimization without the Lipschitz constant, J. Optim. Theory Appl. 79 (1993), 157-181.

[106] P. Jovanovic, M. Kreuzer, Algebraic attacks using AST-Solvers, Groups - Complexity - Cryptology 2 (2010), 247-259.

[107] M. Jünger, T. Liebling, D. Naddef, G.L. Nemhauser, W.R. Pulleyblank, G. Reinelt, G. Rinaldi and L.A. Wolsey, 50 Years of Integer Programming 1958-2008, Springer, Heidelberg 2010.

[108] B. Kalantari and A. Bagchi, An algorithm for quadratic zero-one programs, Naval Research Logistics 37 (1990), 527-538.

[109] R. Kannan and A. Bachem, Polynomial algorithms for computing the Smith and Hermite normal forms of an integer matrix, SIAM Journal of Computing 8 (1979), 499-507.

[110] A. Kehrein and M. Kreuzer, Characterizations of border bases, J. Pure Appl. Alg. 196 (2005), 251-270.

[111] A. Kehrein and M. Kreuzer, Computing border bases, J. Pure Appl. Alg. 205 (2006), 279-295.

[112] A. Kehrein, M. Kreuzer and L. Robbiano, An algebraist's view on border bases, In Solving Polynomial Equations: Foundations, Algorithms and Applications, Springer, 2005, 169-202.

[113] M.E. O'Kelly, A quadratic integer program for the location of interacting hub facilities, European Journal of Operational Research 32 (1987), 393-404.

[114] N. Koblitz, Algebraic aspects of cryptography, Algorithms and Computation in Mathematics, Springer, 1997.

[115] G.A. Kochenberger, F. Glover, B. Alidaee and C. Rego, An unconstrained quadratic binary programming approach to the vertex coloring problem, Annals of Operations Research 139 (2005), 229-241.

[116] J. Kollár, Sharp effective Nullstellensatz, J. Amer. Math. Soc. 1 (1988), 963-975.

[117] M. Kreuzer, Algebraic attacks galore!, Groups - Complexity - Cryptology 1 (2009), 231-259.

[118] M. Kreuzer, S. Kühling, Logik für Informatiker, München, Pearson Studium, 2006.

[119] M. Kreuzer and H. Poulisse, Subideal border bases, preprint 2009.

[120] M. Kreuzer and L. Robbiano, Commputational commutative algebra 1, Springer, Heidelberg 2000.

[121] M. Kreuzer and L. Robbiano, Commputational commutative algebra 2, Springer, Heidelberg 2005.

[122] M. Kreuzer and L. Robbiano, Deformations of border bases, Collectanea Mathematica 59 (2008), 275-297.

[123] B. LaMacchia and A. Odlyzko, Solving large sparse linear systems over finite fields, CRYPTO'90, LNCS 537, pp.109-133.

[124] M. Lamberger, T. Nad and V. Rijmen, Numerical solvers in cryptanalysis, J. Math. Cryptology 3 (2009), 249-263.

[125] D. Lazard, Algèbre linéaire sur $K[x_1, \ldots, x_n]$ et élimination, Bulletin de las S.M.F 105 (1977), 165-190.

[126] D. Lazard, Gaußian elimination and resolution of systems of algebraic equations, In Proc. EUROCAL 83, LNCS 162 (1983), pp. 146-157.

[127] T.L. Lee, T.Y. Li and C.H. Tsai, HOM4PS-2.0: a software package for solving polynomial systems by the polyhedral homotopy continuation method, Computing 83 (2008), 109-133. available at `http://www.math.msu.edu/~li/Software.htm`

[128] J.K. Lenstra and A.H.G. Rinnooy Kan, Computational complexity of discrete optimization problems, Annals of Discrete Mathematics, North-Holland Publishing Company, 1979.

[129] S.R. Li and W.W. Li, Independence numbers of graphs and generators of ideals, Combinatorica 1 (1981), 55-61.

[130] R. Lidl and H. Niederreiter, Introduction to finite fields and their applications, Cambridge University Press, 1986.

[131] J. Limbeck, Implementation und optimierung algebraischer angriffe, diploma thesis, Universität Passau 2008.

[132] J. Lloyd, Foundations of logic programming, Springer, New York 1987.

[133] L. Lovász, Stable sets and polynomials, Discrete Mathematics 124 (1994), 137-154.

[134] Y. Matiyasevich, A criteria for colorability of vertices stated in terms of edge orientations, (in Russian), Discrete Analysis (Novosibirsk) 26 (1974), 65-71.

[135] Y. Matiyasevich, Some algebraic methods for calculation of the number of colorings of a graph, (in Russian), Zapiski Nauchnykh Seminarov POMI 293 (2001), 193-205.

[136] R.D. Mcbride and J.S. Yormark, An implicit enumeration algorithm for quadratic integer programming, Management Science 26 (1980).

[137] A.J. Menezes, P.C.V. Oorschot and S.A. Vanstone, Handbook of applied cryptography, CRC Press, 1996, available at `http://www.cacr.math.uwaterloo.ca/hac/`.

[138] M. Mnuk, Representing graph properties by polynomial ideals, in: V. G. Ganzha, E. W. Mayr and E. V. Vorozhtsov (eds.), Computer Algebra in Sienific Computing, CASC 2001, Proc. Fourth Int. Workshop on Computer Algebra in Scientific Computing, Konstanz, Springer-Verlag, (2001), pp. 431-444.

[139] H.M. Möller, Systems of algebraic equations solved by means of endomorphisms, LNCS 673 (1993), 43-56.

[140] M.S.E. Mohamed, J. Ding and J. Buchmann, Algebraic cryptanalysis of MQQ public key cryptosystem by MutantXL, Technical Report 2008/451, Cryptology ePrint Archive 2008.

[141] M.S.E. Mohamed, W.S.A. Mohamed, J. Ding and J. Buchmann, The complexity analysis of the MutantXL family, preprint 2010.

[142] M.S.E. Mohamed, W.S.A. Mohamed, J. Ding and J. Buchmann, MXL2: solving polynomial equations over GF(2) using an impreved mutant strategy, Proceedings of the Second international Workshop on Post-Quantum Cryptography (PQCrypto08), LNCS, Cincinnati, USA, Springer-Verlag, Berlin (2008), pp.203-215.

[143] M.S.E. Mohamed, D. Cabarcas, J. Ding, J. Buchmann and S. Bulygin, MXL3: An efficient algorithm for computing Gröbner bases of zero dimensional ideals,

In Proceedings of the 12th International Conference on Information Security and Cryptology, (ICISC 2009), LNCS 5984, Springer-Verlag, Berlin (2010), pp.87-100.

[144] J. J. Moré, The Levenberg-Marquardt algorithm: implementation and theory, Numerical Analysis, in: G. A. Watson (ed.), Lecture Notes in Mathematics 630, Springer-Verlag, 1977, 105-116.

[145] B. Mourrain, A new criterion for normal form algorithms, in: M. Fossorier, H. Imai, S. Lin, A. Poli (eds.), Proc. Conf. AAECC-13, Honolulu 1999, LNCS 1719, Springer, Heidelberg 1999, pp. 440-443.

[146] S. Murphy and M. Robshaw, Essential algebraic structure within the AES, In Proceedings of Crypto 2002, LNCS 2442, Springer, (2002), pp. 1-16.

[147] J. Nocedal and S. J. Wright, Numerical optimization, Springer, New York 2006.

[148] S. Onn, Nowhere-zero flow polynomials, Journal of Combinatorial Theory, Series A, 108 (2004), 205-215.

[149] PARI/GP, version `2.3.5`, Bordeaux, 2010, available at `http://pari.math.u-bordeaux.fr/`.

[150] J. Patarin, Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): Two new families of asymmetric algorithms, in: EUROCRYPT, (1996), pp. 33-48, Extended version available at `http://www.minrank.org/hfe.pdf`.

[151] J. Patarin and L. Goubin, Trapdoor one-way permutations and multivariate polynomials, In International Conference on Information Security and Cryptology 1997, LNCS 1334, International Communications and Information Security Association, Springer, 1997, pp. 356368.

[152] M.C. Plateau, A. Billionnet and S. Elloumi, Eigenvalue methods for linearly constrained quadratic 0-1 problems with application to the densest k-subgraph problem. In 6èeme congrèes ROADEF, Tours, 14-16 féevrier, Presses Universitaires Francois Rabelais, 2005, pp. 55-66.

[153] C. Pomerance and J.W. Smith, Reduction of huge, sparse matrices over finite fields via created catastrophes, Experimental Mathematics 1 (1992), 89-94.

[154] H. Raddum, Cryptanalytic results on Trivium, eSTREAM report 2006/039 (2006), available at `http://www.ecrypt.eu.org/stream/triviump3.html`

[155] H. Raddum and I. Semaev, Solving multiple right hand sides linear equations, Des. Codes Cryptogr. 49 (2008), pp. 147-160.

[156] J.C. Rosales, P.A. García-Sánchez, J.I. García-García and M.B. Branco, Systems of inequalities and numerical semigroups, J. London Math. Soc. 65 (2002), 611-623.

[157] N.V. Sahinidis, Global optimization and constraint satisfaction: the branch-and reduce approach, in: C. Bliek, C. Jermann and A. Neumaier (eds.), Global Optimization and Constraint Satisfaction, LNCS 2861, Springer Verlag, 2003, pp. 1-16.

[158] A. Saxena, P. Bonami and J. Lee, Convex relaxations of non-convex mixed integer quadratically constrained programs: extended formulations, Mathematical Programming 124 (2010), 383-411.

[159] A. Schrijver, Theory of linear and integer programming, Wiley, 1986.

[160] C.E. Shannon, Communication theory of secrecy systems, Bell System Tech. J. 28 (1949), 656-715.

[161] H.D. Sherali and W.P. Adams, A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems, SIAM Journal on Discrete Mathematics 3 (1990), 411-430.

[162] H.D. Sherali and J.C. Smith, An improved linearization strategy for zero-one quadratic programming problems, Optimization Letters 1 (2007), 33-47.

[163] A. Simis, W. Vasconcelos and R. Villarreal, On the ideal theory of graphs, J. Algebra 167 (1994), 389-416.

[164] H.J.S. Smith, On systems of linear indeterminate equations and congruences, Phil. Trans. Roy. Soc. London 151 (1861), 293-326.

[165] A.J. Sommese and C.W. Wampler, The numerical solution of systems of polynomials: arising in engineering and science, World Scientific Publishing Company, 2005.

[166] M. Soos, K. Nohl and C. Castelluccia, Extending SAT solvers to cryptographic problems, in: O. Kullmann (ed.), Theory and Applications of Satisfiability Testing - SAT 2009, LNCS 5584, Springer-Verlag, 2009.

[167] A. Storjohann, Near optimal algorithms for computing Smith normal forms of integer matrix, In Proc. International symposium on Symbolic and algebraic computation, ISSAC '96, New York, 1996.

[168] H.A. Taha, A Balasian-Based algorithm for 0-1 polynomial programming, Research Report No. 70-2, University of Arkansas, May 1970.

[169] J. Verschelde, Algorithm 795: PHCpack: a general-purpose solver for polynomial systems by homotopy continuation. ACM Trans. Math. Softw. 25 (1999), 251-276. Available at `http://www.math.uic.edu/~jan/download`.

[170] S. Viswanathan, Configuring cellular manufacturing systems: a quadratic integer programming formulation and a simple interchange heuristic, International Journal of Production Research 33 (1995), 361-376.

[171] L.J. Watters, Reduction of integer polynomial programming problems to Zero-One linear Programming problems, Opns. Res. 15 (1967), 1171-1174.

[172] D. Wiedemann, Solving sparse linear equations over finite fields, IEEE Transaction on Information Theory, v. IT-32 1 (1976), pp. 54-62.

[173] L.A. Wolsey and G.L. Nemhauser, Integer and combinatorial optimization, Wiley Interscience, Hoboken 1999.

[174] H. Xiaozheng, A. Chen, W.A. Chaovalitwongse and H.X. Liu, An improved linearization technique for a class of quadratic 0-1 programming problems, Optimization Letters, Published Online, (2010).

[175] W.I. Zangwill, Media selection by decision programming, J. Advert. Res. 5 (1965), 23-27.