Technical Report

UCAM-CL-TR-884
ISSN 1476-2986

Number 884

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Machine learning and computer algebra

Zongyan Huang

April 2016

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

http://www.cl.cam.ac.uk/

This technical report is based on a dissertation submitted August 2015 by the author for the degree of Doctor of Philosophy to the University of Cambridge, Lucy Cavendish College.

Technical reports published by the University of Cambridge Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/techreports/

ISSN 1476-2986

Abstract

Computer algebra is a fundamental research field of computer science, and computer algebra systems are used in a wide range of problems, often leading to significant savings of both time and effort. However, many of these systems offer different heuristics, decision procedures, and parameter settings to tackle any given problem, and users need to manually select them in order to use the system.

In this context, the *algorithm selection problem* is the problem of selecting the most efficient setting of the computer algebra system when faced with a particular problem. These choices can dramatically affect the efficiency, or even the feasibility of finding a solution. Often we have to rely on human expertise to pick a suitable choice as there are no fixed rules that determine the best approach, and even for experts, the relationship between the problem at hand and the choice of an algorithm is far from obvious.

Machine learning techniques have been widely applied in fields where decisions are to be made without the presence of a human expert, such as in web search, text categorization, or recommender systems. My hypothesis is that machine learning can also be applied to help solve the algorithm selection problem for computer algebra systems.

In this thesis, we perform several experiments to determine the effectiveness of machine learning (specifically using support vector machines) for the problem of algorithm selection in three instances of computer algebra applications over real closed fields. Our three applications are: (i) choosing decision procedures and time limits in MetiTarski; (ii) choosing a heuristic for CAD variable ordering; (iii) predicting the usefulness of Gröbner basis preconditioning. The results show that machine learning can effectively be applied to these applications, with the machine learned choices being superior to both choosing a single fixed individual algorithm, as well as to random choice.

Acknowledgements

First, I would like to acknowledge my supervisor Prof Lawrence C. Paulson for his continuous guidance and countless hours of supervision throughout my PhD study. I could not have asked for a more supportive and kind supervisor. My heartfelt thanks also go to my second supervisor, Dr Sean Holden. I thank him for his expert advice on machine learning topics, and for his endless patience with me, answering even my most obvious questions. I consider myself very lucky to have been their student and I am forever grateful! Furthermore I'd like to thank Prof Mike Gordon and Prof Mahesan Niranjan for being both knowledgeable and fair examiners during my PhD viva. Their suggested amendments helped to greatly improve this thesis.

I feel fortunate for the great people in the Computer Laboratory and appreciate the many fruitful discussions we had. In particular, I want to thank Dr Timothy G. Griffin who kindly provided me with his warm-hearted encouragements and insightful suggestions on this dissertation. I would also like to thank James Bridge for our discussions on machine learning, and for being a great office mate. I am also greatly indebted to Grant Passmore, for his expert advice on real closed fields, and for his valuable feedback on this thesis. I am also truly thankful to Pengming Wang and Ramana Kumar for their comprehensive proofreading and their helpful comments and to Wenda Li and Jean Pichon for their additional feedback.

Thanks go to my collaborators Dr Matthew England and Dr David Wilson, for the enjoyable collaboration, for the many helpful discussions, and for their comments on this thesis.

Finally, I would like to thank my parents for their continuous and unconditional encouragement and support.

Contents

Chapter 1

Introduction

1.1 Computer algebra systems

Computer algebra is the formal manipulation of symbols which represent mathematical objects such as numbers, polynomials, rational functions, systems of equations, and algebraic structures. Algebraic computations follow the rules of algebraic algorithms rather than using approximate numeric values. Examples of algebraic computations include factorization of polynomials differentiation, integration, series expansion of functions, and simplification of mathematical expressions [72]. With computer algebra, we can solve a wide range of problems which cannot easily be tackled with pencil and paper using traditional methods.

Computer algebra systems are software packages that are used for carrying out algebraic computations. One of the first computer algebra systems was developed in the 1960s by Martin Veltman, who designed a program for high energy physics called SCHOONSCHIP [96]. Soon after that, another computer algebra system MATHLAB [38] was created by Carl Engelman, which was already able to perform quite a few tasks such as differentiation, polynomial factorization, direct and inverse Laplace transforms and so on. Popular early computer algebra systems also include muMATH [109], REDUCE [55], DERIVE (based on muMATH) [46], and MACSYMA [89]. In the last sixty years, great progress has been made on the theoretical background of symbolic and algebraic algorithms. As of today, the most popular commercial systems are MATHEMATICA [108] and MAPLE [56], which are commonly used in almost all branches of science, including physics, mathematics, chemistry, biology, robotics and economics [45, 103, 95].

Computer algebra systems are designed to help humans solve large problems. They automate the problem solving process, and usually obtain more reliable results than hand calculations. Ideally, such computer algebra systems should be fully automatic and therefore easy to use. In practice they are not; users need to choose from many decision procedures, heuristics, and parameter settings for performing the computation. However, it is not easy to make the right choice when studying a particular problem. Furthermore, these choices are often critical: some problems are infeasible with one choice but easy with another. I call the problem of finding the right setting for the computer algebra system the *algorithm selection problem* (decision procedures, heuristics, and parameter setting are referred to as algorithms).

1.2 Real closed fields (RCFs)

Computer algebra is a huge area with many applications. In this work, I will focus on three instances of the algorithm selection problem for computer algebra tasks which all relate to real closed fields (RCFs) [7]. The theory of RCFs concerns (possibly quantified) boolean combinations of polynomial equations and inequalities over real numbers. Below I give an overview of the three instances and point out the algorithm selection problem that arises from them. Details will be covered in Chapter 3, 4 and 5 respectively.

1. The first application includes making a problem-dependent selection of the best RCF decision procedure (Z3 with Strategy 1, MATHEMATICA and QEPCAD) and the best time limit (0.1s, 1s, 10s, 100s) on individual RCF subproblems in the automatic theorem prover METITARSKI.

   METITARSKI combines a resolution theorem prover (METIS) with a set of axioms and a collection of decision procedures for the theory of RCFs. During its proof search, it generates a series of RCF subproblems which are reduced to true or false using an RCF decision procedure. The theory of RCFs is decidable and there exists a variety of decision procedures to decide statements over RCFs. Specialised variations of RCF decision procedures have their own strengths and weaknesses for restricted classes of formulas. In practice, there is currently no single RCF decision procedure which is good for all problems. Which decision procedure is the most appropriate one to call is highly dependent on the (usually geometric) properties of the RCF problem being considered, which is in turn largely influenced by the structure of the original MetiTarski problem. Hence, selecting the most suitable decision procedure is a non-trivial task.

   Moreover, many of the RCF subproblems do not contribute towards METITARSKI's final proof, and time spent deciding their satisfiability is wasted. Setting a time limit on individual RCF subproblems reduces the wasted time but too tight a limit could affect METITARSKI's proof. Hence, it is important to make a good choice for the time limit on individual RCF subproblems.

2. The second application is choosing a heuristic (`sotd`, `ndrr`, `Brown`) to select the variable ordering for cylindrical algebraic decomposition (CAD).

   CAD is one of the main practical tools in computational algebraic geometry, particularly for quantifier elimination over RCFs. When using CAD, we often have a choice over the variable ordering used, which can dramatically affect the feasibility of a problem. Various heuristics have been developed to help with this choice, but no one heuristic is suitable for all problems.

3. The third application is predicting whether Gröbner basis preconditioning is useful or not on a particular problem.

   Gröbner basis computation is a key tool for solving many fundamental problems involving polynomial equations. We can apply Gröbner basis preconditioning when using CAD for a problem with multiple equalities. Wilson et al. [106] showed this usually gives a better CAD, but sometimes a worse one. There is no fixed rule for predicting whether Gröbner basis preconditioning gives a better CAD.

In all three of these tasks, we have the problem of algorithm selection. At the same time, there is no universal optimal choice for most problems. The fact that the best choice is dependent upon the problem considered makes the algorithm selection problem a good candidate for applying machine learning techniques. The following section aims to provide a high-level introduction to machine learning and motivate the use of machine learning techniques; a more thorough discussion will be given in Section 2.2.

1.3 Machine learning

Machine learning is the process of fitting a complex function based on properties learned from labelled data [9]. It is a data-driven process, as we learn everything from data, rather than following explicitly programmed instructions. Machine learning seems to be a good fit for the algorithm selection problem as it is designed to model relationships which are too complex for a complete analytical approach. My research concerns whether machine learning is applicable to the field of computer algebra. In particular, I investigated the effectiveness of machine learning techniques to the three instances of algorithm selection in computer algebra over RCFs described in the previous section.

In my work, I take some algebraic measures (features) of the problems as input and produce the predicted algorithm as output. This is a standard classification problem where each algorithm defines a class of problems for which it is the best choice. By running all possible algorithms on a large number of sample problems, we can determine the best one in each case, and produce training samples for supervised learning. In literature, selection algorithms are often ranked by their average performance over a set of benchmark problems. A selection algorithm is considered good or useful when its performance is better than any of the individual current algorithms. We show that by applying machine learning, we can indeed improve on the performance of the individual algorithms when evaluated on a set of unseen problems.

1.4 The hypothesis

My hypothesis is that machine learning can be applied to help solve the algorithm selection problem in computer algebra systems. In particular, I consider a number of fundamental computer algebra problems related to RCFs, namely quantifier elimination and CAD computation, and explore the applicability of machine learning to certain algorithm selection tasks within these problems. In this thesis, I conduct an experimental study and provide evidence that leading machine learning techniques (such as support vector machines) can indeed be applied to this task, and produce better results than the baseline methods.

1.5 Contributions

The main contribution of the dissertation is to show support of my hypothesis, and the specific contributions are as follows.

- **Chapter 3: Choosing decision procedures and time limits in MetiTarski**

1. The application of machine learning to select the best decision procedure (Z3 with Strategy 1, MATHEMATICA and QEPCAD) in METITARSKI was investigated. As a benchmark, machine learning process outperformed any fixed decision procedure, and choosing the best decision procedure proved 163 out of 194 problems, showing that machine learned selection achieved an 84% optimal choice.
2. A further experiment was conducted to select between Z3 and MATHEMATICA and to set the best time limit (0.1s, 1s, 10s, 100s) on RCF calls for a given MetiTarski problem. The machine learned algorithm for selection performed better on our benchmark set than any of the individual fixed settings used in isolation.

- **Chapter 4: Choosing a heuristic for CAD variable ordering**

1. The application of machine learning to the problem of choosing a heuristic to select the variable ordering for CAD and quantifier elimination by CAD (`sotd`, `ndrr`, `Brown`) was investigated, using the `nlsat` dataset of fully existentially quantified problems (removing all quantifiers gave a corresponding problem set for evaluating CAD alone). The machine learning algorithm selected an optimal heuristic for 76% of the quantifier-free problem and 77% of the quantified problems (compared with 58% and 64% for a random choice and 64% and 74% for the best performing heuristic (`Brown`)), indicating that there is a relationship between the simple algebraic features and the best heuristic choice.
2. The experimental data showed that if machine learning is not available then `Brown` heuristic is a great alternative, with `sotd` performing only slightly worse. `Ndrr` as an individual heuristic performed rather poorly, and is best suited to be used within a hybrid heuristic to break ties.

- **Chapter 5: Predicting the usefulness of Gröbner basis preconditioning**

1. I investigated the usefulness of machine learning in the task of deciding whether to use Gröbner bases preconditioning on CAD inputs. Using machine learning yielded better results than either always using Gröbner basis preconditioning or no preconditioning.
2. Experiment results show that a reduced feature sets can be extracted that results in a more effective learning in the experiment: the feature subset suggested by the filter method successfully predicted average 79% of the problems and using the feature subset suggested by the wrapper method successfully predicted average 78% of the problems from 50 runs of the 5-fold cross validation (compared to 75% when always using Gröbner basis preconditioning by default).
3. The optimal feature subset contains algebraic properties from both the original input and its Gröbner basis. The properties related to the first projected variable affects heavily to the complexity of the rest of the algorithm.

The work described in Chapter 4 and 5 are joint work with a research group at the University of Bath (Dr Matthew England and Dr David Wilson). In Chapter 4, the work has been published already [58, 59]. The dissertation itself is my own work, with assistance from England and Wilson in providing scripts to format the examples, and to construct the CADs in QEPCAD. The three heuristics used in this experiment were implemented by England in MAPLE. The entire machine learning experiment was conducted by myself; the results were analysed and discussed by both research groups. In Chapter 5, the initial application was proposed by the Bath group. The experiment was done and analysed by me.

1.6 Organisation of the dissertation

Chapter 2 covers the important background knowledge for the three listed computer algebra applications and the background knowledge of machine learning, with an emphasis on support vector machines. Chapter 3 describes two separate but similar experiments: applying machine learning to the problem-dependent selection of the most efficient RCF decision procedure and the problem-dependent selection of the best time limit setting on individual RCF subproblems in the theorem prover METITARSKI. Chapter 4 investigates the application of machine learning to the problem of choosing a heuristic to select the variable ordering for CAD and quantifier elimination by CAD. Chapter 5 investigate the application of machine learning to the prediction of whether Gröbner basis preconditioning is useful or not. In Chapter 6, I review related work. I conclude the dissertation with a summary in Chapter 7 and also describe future directions for my work.

Chapter 2

Background

2.1 Computer algebra

I give here the relevant background to understand the tasks for which I employ machine learning techniques. For further background, we refer to the relevant text books [10, 29].

2.1.1 RCFs

Decision procedures are of great use in the formal verification of safety-critical systems and formalized mathematics. We are concerned with decision procedures for the theory of *RCFs*, which is a theory in the language of ordered rings (that is, structures containing quantified boolean combinations of equalities and inequalities involving addition, subtraction and multiplication of multivariate polynomials with rational coefficients) about fields which share the algebraic properties of the field of real numbers.

For our purposes, the most important fact about the theory of RCFs is that it is a decidable theory. In other words, there is an algorithm for deciding the truth, in any real closed field, of any proposition in the language of ordered rings. This was proved by Tarski [99]. Completeness of the theory of RCFs means we can prove results over all RCFs, while still being sure that they are valid over $\mathbb{R}$. This is important from a computational point of view, as $\mathbb{R}$ is uncountable with uncomputable basic operations.

Two examples of RCFs are the field of real numbers $\mathbb{R}$ and the field of *real algebraic numbers* $\mathbb{R}_{alg}$. In the classical approach, we can perform computation over the real algebraic numbers $\mathbb{R}_{alg}$ instead of $\mathbb{R}$. An algebraic number is a real number that is a root of a (non-zero) univariate polynomial with rational coefficients. This structure is a countable real closed field with computable basic operations, and thus provides a logically sufficient computational substructure for making RCF decisions. If a solution exists in $\mathbb{R}_{alg}$, then this is also a solution in $\mathbb{R}$, as $\mathbb{R}_{alg}$ is a subfield of $\mathbb{R}$. Note, this field contains no *transcendental* elements such as π or e, although recent research has addressed this issue by a combination of transcendental constants and infinitesimals with nonlinear real arithmetic [2, 35].

The connection between $\mathbb{R}_{alg}$ and other RCFs is the key property which allows computer algebra systems to be used in tackling RCF problems. The rest of the chapter describes three computer algebra problems concerning RCFs.

2.1.2 Quantifier elimination (QE) and cylindrical algebraic decomposition (CAD)

The task of QE [48] for the first-order language over RCFs is a central problem in computer algebra. We start by defining the problem and then discuss the approach of solving QE via computing CAD. Here, a *first-order formula* ϕ over RCF consists of atomic formulas in the form of polynomial (in)equalities $p(x_1, \ldots, x_k) \leq 0$ or $p(x_1, \ldots, x_k) = 0$, and the standard operations of negation, conjunction, disjunction, first order universal and existential quantification applied onto atomic formulas. Then, the problem of QE is defined as follows.

Definition 1 *Let $Q_i \in \{\exists, \forall\}$ be quantifiers and ϕ be some quantifier-free formula. Then given*

$$\Phi(x_1, \ldots, x_k) := Q_{k+1}x_{k+1} \ldots Q_n x_n \, \phi(x_1, \ldots, x_n), \tag{2.1}$$

quantifier elimination *is the problem of producing a quantifier-free formula $\psi(x_1, \ldots, x_k)$ equivalent to Φ.*

Tarski proved that QE is computable in the theory of RCFs [99]. However, the complexity of Tarski's method is non-elementary (indescribable as a finite tower of exponentials). Later, a more efficient method for computing QE, using CAD, was introduced by Collins [26].

Definition 2 *A **sign-invariant CAD** is a decomposition of the n-dimensional real space into connected semialgebraic sets (described by polynomial relations), called cells, such that a given set of polynomials has constant sign on each cell. The decomposition is called cylindrical, if projections of any two cells onto their first i coordinates are either identical or disjoint (with respect to a given variable ordering).*

CADs with other invariance structures still sufficient for QE have also been investigated [74, 75], but we will work with the above variant.

Given a quantified first order formula, the CAD of its polynomial terms can be used to implement QE. The simplest approach is to ignore the quantifers of the original formula and then rebuild the quantifier-free formula based on the associated cells. More specifically, by computing the sign-invariant CAD for each polynomial term occurring in the formula, we can evaluate the quantifications separately for each polynomial (in)equality. Since in each cell each polynomial is sign-invariant, the corresponding polynomial term is truth-invariant. Hence, an equivalent quantifier-free formula is simply given by disjunctions of the truth cells of the different decompositions. Other, more sophisticated CAD algorithms use the quantifier structure of the formula for short-cuts and improvements [27, 85, 61]. The most famous example for this is Partial CAD [27] where the algorithm stops the lifting step once it already knows that cells will be false based on quantifier information.

The CAD algorithm was a major breakthrough when introduced, despite its doubly exponential complexity in the number of variables, since Tarksi's method is infeasible in practice. For some problems, QE is possible through algorithms with better complexity (see for example the survey by Basu [7]), but CAD implementations remain the best general purpose approach. Although CAD was first introduced to implement quantifier elimination over the reals, it has since been applied to applications including robot motion

planning [105], programming with complex valued functions [32], optimisation [44] and epidemic modelling [19].

I now give a brief overview over Collins' algorithm [4] for computing CADs. The algorithm works in two stages. First, the *projection* stage calculates sets of projection polynomials S_i in variables $(x_1, \ldots, x_i)$. This is achieved by repeatedly applying a projection operator to a set of polynomials, producing a set with one fewer variable. We start with the polynomials from ϕ and eliminate variables this way until we have the set of univariate polynomials S_1.

Then in the *lifting* stage, decompositions of real spaces in increasing dimensions are formed according to the real roots of those polynomials. First, the real line is decomposed according to the roots of the polynomials in S_1. Then over each cell c in that decomposition, the bivariate polynomials S_2 are taken at a sample point and a decomposition of $c \times \mathbb{R}$ is produced according to their roots. Taking the union gives the decomposition of $\mathbb{R}^2$ and we proceed this way to a decomposition of $\mathbb{R}^n$. The resulting decomposition is cylindrical and each cell is a semi-algebraic set.

2.1.3 Variable ordering in CAD

When using CAD, we have to assign an ordering to the variables. This dictates the order in which the variables are eliminated during projection and thus the sub-spaces for which CADs are produced en route to a CAD of $\mathbb{R}^n$. For some applications this order is fixed but for others there may be a free or constrained choice. When using CAD for quantifier elimination we must project quantified variables before unquantified ones. Furthermore, the quantified variables should be projected in the order they occur, unless successive ones have the same quantifier in which case they may be swapped. The ordering can have a big effect on the output and performance of CAD [18, 36, 11]. In fact, Brown and Davenport [18] present a class of problems in which one variable ordering gives an output of double exponential complexity in the number of variables and another gives an output of a constant size. Heuristics have been developed to help with this choice, with Dolzmann et al. [36] giving the best known study. However, it was shown that even the best known heuristic could be misled (see Bradford et al. [11]). There is no single heuristic which is suitable for all problems. The best heuristic to use is dependent upon the problem considered. However, the relationship between problems and heuristics is far from obvious and so we investigate whether machine learning can help with these choices. I will discuss this in detail in Chapter 4.

2.1.4 Gröbner basis

A useful tool for a more efficient computation of CADs is the Gröbner basis, first introduced by Buchberger [20], together with an algorithm to compute them (Buchberger's algorithm). Intuitively, the Gröbner basis of a set of multivariate polynomials is a generating set of a polynomial ideal that has certain nice algorithmic properties. I will not give a formal derivation of its algebraic properties here, as it is not in the scope of this thesis, but rather refer to [20] for details. However, it has been shown that the Gröbner basis is one of the main practical tools for solving systems of polynomial equations [25, 12, 84], and interestingly, computing the Gröbner basis of a system of polynomial (in)equalities can be used as a preconditioning step before applying CAD. Wilson [106] has shown that this preconditioning step often results in a sharp drop in CAD complexity and construction

time. Since the Gröbner basis generally removes any redundancies from a set of polynomials by reducing polynomials with respect to each other or identifying common factors, this will usually lead to simpler projection sets by having simpler polynomials to start with. However, computing the Gröbner basis is not universally beneficial, as the Gröbner basis might introduce extra polynomials of possibly high degree, and there is no fixed rule to decide if Gröbner preconditioning is beneficial or not. In my experiments (Chapter 5), I apply machine learning techniques to help decide whether Gröbner preconditioning is useful depending on the problem instance.

2.1.5 MetiTarski

Automated reasoning for mathematical proof is a key component that many software verification and program analysis tools rely on. Automated theorem proving (ATP) [73] involves computer programs that can prove theorems automatically. ATP has been successfully used in many fields, e.g. mathematics, computer science, engineering, and social science. Many applications require reasoning about special functions such as logarithms, sines, cosines and so forth. MetiTarski is an ATP which can prove inequalities involving those special functions [1, 2]. Paulson and Akbarpour have shown how MetiTarski can be used to prove safety properties about hybrid systems. For example, in the collision avoidance system, the key property to check is if the gap between two cars is larger than 0. As a result, the following formula with special functions is derived:

$$0 \leq x \leq 2 \implies 12 - 14.2\exp(-0.318x) + \\ (3.25\cos(1.16x) - 0.155\sin(1.16x))\exp(-1.34x) > 0$$

MetiTarski can prove this formula within a second. MetiTarski can also prove a wide variety of problems derived from the verification of Nichols plots. These typically involve the arctangent, logarithm and square root functions.

2.1.6 RCF decision procedures

MetiTarski works by eliminating special functions, substituting rational function upper or lower bounds, transforming parts of the problem into polynomial inequalities, and finally applying a external decision procedure for the theory of RCF. RCF decision procedures are used to simplify clauses by deleting literals that are inconsistent with other algebraic facts. RCF decision procedures are also used to discard redundant clauses that follow algebraically from other clauses [2].

In this thesis, machine learning is applied to find the most efficient RCF decision procedure for a given MetiTarski problem. Three RCF decision procedures were tested: Z3 with Strategy 1 [83], Mathematica [108] and Qepcad [15]. The SMT solver Z3 [34] has an internal module called `nlsat` [63] that implements an efficient method for deciding purely existential RCF sentences. Combined with strategies tailored to the types of RCF problems generated by MetiTarski, it has been used to successfully prove problems of up to 11 variables. Z3 with Strategy 1 [83] is a refined version of Z3, which is currently the default algebraic decision procedure used by MetiTarski. Passmore et al. showed that by applying model sharing and omitting the standard test for irreducibility with Z3, its proof performance could be substantially improved. The decision procedure Qepcad [15] is an

interactive command line program for performing **Q**uantifier **E**limination with **P**artial **CAD**. It is very efficient on single variable problems, but can only deal with problems in less than four variables in reasonable time / memory constraints. MATHEMATICA [108] contains a family of highly advanced RCF decision procedures (including CAD, Gröbner Bases and more), which allow METITARSKI to handle efficiently problems with up to 4 or 5 variables.

2.2 Machine learning

Machine learning is the process of learning rules from and making predictions on data. The emphasis of machine learning is to program by example rather than to program the exact instructions to solve the task. Machine learning methods have been successfully used in many applications. For example, in speech recognition, machine learning techniques learn patterns of speech signals to understand the words [3]. In facial recognition, they work by finding patterns in images that match those of faces [82]. They are also used to develop classifiers for detection of disease in medical diagnosis [69].

Supervised learning [9] is one of the fundamental methods in machine learning. It infers a function from data labelled with the corresponding correct outputs and then can be used on new examples. Each example is a pair consisting of a set of input variables and a desired output value. By contrast, *unsupervised learning* aims to find hidden structure in unlabelled data.

The work in this dissertation uses supervised learning exclusively. The support vector machine (SVM) learning algorithm is among the best supervised learning algorithms. It was selected for the work of this dissertation. In the rest of the chapter, we will give a brief overview of the basic concepts relevant to SVM, and how to extend the basic binary SVM model to multiclass SVMs and common feature selection approaches. For a more thorough treatment, we refer to Cristianini [30] and Bishop [9].

2.2.1 Support vector machines (SVMs)

The SVM is a very popular machine learning technique. It is a supervised learning method used for classification and regression. *Classification* refers to the assignment of input examples into a given set of categories (the output being the class label). *Regression* refers to a supervised pattern analysis in which the output is real-valued. In the classification task, input data are called *training data* and each is marked as belonging to one of K classes. The SVM training model maps the data into a higher-dimensional space where the K classes are separated by a hyperplane. This higher-dimensional space is called the *transformed feature space*, as opposed to the *input space* occupied by the training instances. The goal of the SVM model is to maximise the gap between the separating hyperplane and so that the expected generalisation error is minimised.

The rest of the section presents the SVM method in more detail. We first motivate the use of SVMs, and then consider the simplest case where data is linearly separably in the input space. Next we introduction the notation of margins and margin separation. We will also give a brief discussion of kernels, which allow us to apply SVMs efficiently in very high dimensional feature spaces, and finally, soft margin SVMs, which is an approach of dealing with outliers in the dataset.

2.2.1.1 Motivation for using SVMs

SVMs were selected as the machine learning technique for this dissertation for a number of reasons. SVMs deliver a unique solution, since the optimality problem is convex. This is an advantage compared to various machine learning techniques, e.g. Neural Networks [54], which have multiple solutions associated with local minima and may not be robust over different samples. Furthermore, SVMs are memory efficient, since the decision boundary is decided only by a small subset of training points in the decision function (called *support vectors*). By introducing kernel methods, SVMs can be efficiently applied in very high dimensional feature spaces, and are flexible in modelling diverse sources of data. Also, SVMs generally provide low generalization error. In the recent paper [42], Delago et al. evaluated 179 classifiers arising from 17 families over the whole `UCI` machine learning classification database. Four classifiers of SVMs ranked within top 10 classifers among 179 classifiers, and the SVM with a radial basis function kernel achieved 92.3% of the maximum accuracy. Additionally, SVMs are well supported with existing software. The main software used in my experiments is SVMLight [62].

2.2.1.2 Linear classifiers

First, we take a look at the simplest case: the binary classification problem. This is the task of separating two sets of data points in space, each corresponding to a given class. We seek to separate the two data sets using simple boundaries. Once the boundary is found, new examples are then predicted to belong to a category based on which side of the boundary they fall. The data for a two-class learning problem consists of objects labelled with $+1$ (positive examples) or -1 (negative examples). Each data instance is represented as a vector $\boldsymbol{x}$ of real numbers (referred to as *features*). A labelled example is then denoted $(\boldsymbol{x}, y)$ where $y \in \{+1, -1\}$. We take a set of these labelled pairs and attempt to construct a discriminant function f that maps input vectors $\boldsymbol{x}$ onto labels y. The goal is to find a f which minimizes the number of errors ($f(\boldsymbol{x}) \neq y$) on future examples.

A linear classifier is based on a linear discriminant function of the form

$$f(\boldsymbol{x}) = \boldsymbol{\omega}^T \cdot \boldsymbol{x} + b, \tag{2.2}$$

where $\boldsymbol{\omega}^T \cdot \boldsymbol{x}$ denotes the inner product of the vectors $\boldsymbol{\omega}$ and $\boldsymbol{x}$. We call $\boldsymbol{\omega}$ the *weight vector*, and b the *bias*. The *decision boundary* divides the space into two sets, depending on the sign of $\boldsymbol{\omega}^T \cdot \boldsymbol{x} + b$. The linear classifier is defined as

$$h_{\boldsymbol{\omega},b}(\boldsymbol{x}) = \begin{cases} +1, & \text{for } \boldsymbol{\omega}^T \cdot \boldsymbol{x} + b \geq 0 \\ -1, & \text{for } \boldsymbol{\omega}^T \cdot \boldsymbol{x} + b < 0 \end{cases} \tag{2.3}$$

If we can find a linear classifier such that all training examples are classified correctly, then we call the data *linearly separable*.

2.2.1.3 Functional margin and geometric margin

Given a hyperplane $\boldsymbol{\omega}^T \cdot \boldsymbol{x} + b = 0$, and a training example $(\boldsymbol{x_i}, y_i)$ in the training set $S = \{(\boldsymbol{x}_i, y_i); i = 1, 2, ..., n\}$, the sign of $y_i(\boldsymbol{\omega}^T \cdot \boldsymbol{x}_i + b)$ is used to judge the correctness of the classification. If $y_i(\boldsymbol{\omega}^T \cdot \boldsymbol{x}_i + b) > 0$, then our prediction on this example is correct. This

introduces the concept of *functional margin*. The functional margin of the hyperplane $\boldsymbol{\omega}^T \cdot \boldsymbol{x} + b = 0$ with respect to the training example $(\boldsymbol{x}_i, y_i)$ is defined as

$$\hat{\gamma}_i = y_i(\boldsymbol{\omega}^T \cdot \boldsymbol{x}_i + b). \tag{2.4}$$

The functional margin of the hyperplane $(\boldsymbol{\omega}, b)$ in terms of S is the minimum value of the functional margin over all training samples:

$$\hat{\gamma} = \min_i \hat{\gamma}_i. \tag{2.5}$$

However, the functional margin is not a very good measure of confidence of the prediction. Observe that the value of $h_{\boldsymbol{\omega},b}(\boldsymbol{x})$ only depends on the sign of $\boldsymbol{\omega}^T \cdot \boldsymbol{x} + b$, but not its magnitude. We can make the functional margin arbitrarily large by scaling $\boldsymbol{\omega}$ and b without changing anything meaningful.

For the margin to be a direct measure, we are rather interested in the normalized value of the functional margin, which brings in the notion of the *geometric margin*. Taking the same training set S, the geometric margin of $(\boldsymbol{\omega}, b)$ with respect to a training example $(\boldsymbol{x}_i, y_i)$ is defined as

$$\gamma_i = y_i \frac{\boldsymbol{\omega}^T \cdot \boldsymbol{x}_i + b}{\|\boldsymbol{\omega}\|} = \frac{\hat{\gamma}_i}{\|\boldsymbol{\omega}\|}. \tag{2.6}$$

The geometric margin of $(\boldsymbol{\omega}, b)$ in terms of S is also defined as the minimum value of the geometric margin over all training samples:

$$\gamma = \min_i \gamma_i. \tag{2.7}$$

The geometric margin can be viewed as the signed distance from the point to the plane and is invariant to rescaling of the parameters.

To summarize, the functional margin gives the position of the training point with respect to the plane, independent of the magnitude, while the geometric margin gives the distance between the given training point and the given plane.

2.2.1.4 Maximum margin separation

Often, there are multiple solutions for a classifier that can classify the data in the training set. In these cases, we want to find the one with the smallest generalization error. The support vector machine approaches this problem through *maximum margin separation*, where the hyperplane that has the largest distance to the nearest training data point of any class (geometric margin) is chosen. Intuitively, a large margin represents high confidence in a classification decision. Recall that scaling $(\boldsymbol{\omega}, b)$ does not change the geometric margin, but does scale the functional margin. Hence, without loss of generality, we can introduce scaling constraints to set the functional margin of $(\boldsymbol{\omega}, b)$ to 1. More specifically, we assume that the constraints $y_i(\boldsymbol{\omega}^T \cdot \boldsymbol{x}_i + b) \geq 1$ always hold.

Figure 2.1 shows a maximum margin boundary computed by a linear SVM. Circles are positive examples and squares are negative examples. The region between the two dashed lines defines the margin area with

$$-1 \leq \boldsymbol{\omega}^T \cdot \boldsymbol{x} + b \leq 1. \tag{2.8}$$

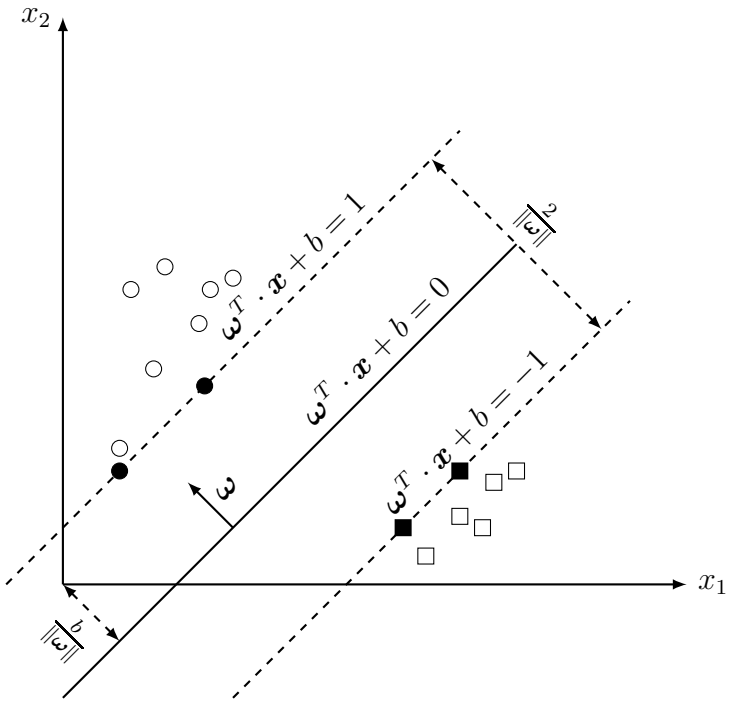


Figure 2.1: Maximum margin separation

The solid line in the middle is the hyperplane, which is defined by the set of nearest examples. These nearest examples are called the *support vectors*. In the figure, there are four support vectors (two solid circles and two solid squares) on the edge of the margin area ($f(\boldsymbol{x}) = -1$ or $f(\boldsymbol{x}) = +1$). The *margin* is defined as the perpendicular distance between the decision boundary and the closest of the data points. To maximise the margin value, we basically seek to solve the following optimization problem:

$$\arg\max_{\boldsymbol{\omega},b} \frac{1}{\|\boldsymbol{\omega}\|}, \tag{2.9}$$

subject to

$$y_i(\boldsymbol{\omega}^T \cdot \boldsymbol{x}_i + b) \geq 1. \tag{2.10}$$

The optimisation problem of maximising $\frac{1}{\|\boldsymbol{\omega}\|}$ is equivalent to minimising $\frac{1}{2}\|\boldsymbol{\omega}\|^2$, and we now have the following:

$$\arg\min_{\boldsymbol{\omega},b} \frac{1}{2}\|\boldsymbol{\omega}\|^2, \tag{2.11}$$

subject to the constrains given by Equation 2.10. This is a constrained quadratic programming optimization problem, for which several standard algorithms exist. However, in the following I will give a different representation of the optimization, which will be very useful when introducing the concept of kernels.

In particular, we introduce *Lagrange multipliers* together with *Karush-Kuhn-Tucker (KKT)* conditions to obtain the *dual form* of our optimization problem, which is referred to as the *primal*. I will not give a formal derivation of the theory of Lagrange duality here, as it is not in the scope of this thesis, but rather refer to [43] for details. Here,

we are mainly interested in showing how this can be applied to our maximum margin optimization problem.

In order to derive the *dual form* of the original optimisation problem, *KKT multipliers*, $\alpha_i \geq 0$, where $i = 1, \ldots, n$ are introduced, and we define the *Lagrangian* to be

$$\mathcal{L}(\boldsymbol{\omega}, b, \alpha) = \frac{1}{2}\|\boldsymbol{\omega}\|^2 - \sum_{i=1}^{n} \alpha_i [y_i(\boldsymbol{\omega}^T \cdot \boldsymbol{x}_i - b) - 1]. \tag{2.12}$$

We now seek to minimise $\mathcal{L}(\boldsymbol{\omega}, b, \alpha)$, and set the derivatives of $\mathcal{L}$ with respect to $\boldsymbol{\omega}$ and b to 0. As for the derivative with respect to $\boldsymbol{\omega}$, we obtain

$$\frac{\partial}{\partial \boldsymbol{\omega}}\mathcal{L} = \boldsymbol{\omega} - \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i = 0, \tag{2.13}$$

which implies that $\boldsymbol{\omega}$ can be expressed as a linear combination of the training vectors

$$\boldsymbol{\omega} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i. \tag{2.14}$$

As for the derivative with respect to b, we obtain

$$\frac{\partial}{\partial b}\mathcal{L} = \sum_{i=1}^{n} \alpha_i y_i = 0. \tag{2.15}$$

By substituting Equation 2.14 and Equation 2.15 back to Equation 2.12, we get

$$\mathcal{L}(\boldsymbol{\omega}, b, \alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,g=1}^{n} \alpha_i \alpha_g y_i y_g \boldsymbol{x}_i^T \cdot \boldsymbol{x}_g, \tag{2.16}$$

Recall that we obtained the equation above by minimizing $\mathcal{L}$ with respect to $\boldsymbol{\omega}$ and b. Hence, we have

$$\arg\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,g=1}^{n} \alpha_i \alpha_g y_i y_g \boldsymbol{x}_i^T \cdot \boldsymbol{x}_g, \tag{2.17}$$

subject to $\alpha_i \geq 0$ and Equation 2.15. This form is known as the dual form of the problem. Note that our optimisation problem is now expressed in terms of the inner product of our sample data. This fact will be key when introducing kernel methods. I will give more detail of how kernel methods work in the following section.

Furthermore, note that due to the KKT dual complementarity conditions, we need to satisfy the following two constraints simultaneously

$$\alpha_i \geq 0 \tag{2.18}$$

$$\alpha_i(-y_i(\boldsymbol{\omega}^T \cdot \boldsymbol{x}_i + b) + 1) = 0. \tag{2.19}$$

This implies that the corresponding α_i for all the points for which we have

$$y_i(\boldsymbol{\omega}^T \cdot \boldsymbol{x}_i - b) - 1 > 0 \tag{2.20}$$

is set to zero. We can see that those points do not matter, and only the points which satisfy

$$y_i(\boldsymbol{\omega}^T \cdot \boldsymbol{x}_i - b) - 1 = 0 \tag{2.21}$$

are relevant. These points are exactly our support vectors, and only they influence the decision boundaries. Since we have only few support vectors compared to the size of the full training data, this allows support vector machines to scale very well with large sets of data.

2.2.1.5 Kernel methods

Sometimes, the data are not linearly separable in the original space. One approach to tackle these instances, is to map the original space into a transferred feature space in which separation is easier. Instead of computing the coordinate transformation into feature space, which is in general computational expensive, SVMs use kernel methods, which simply project all pairs of data into the feature space. Suppose we have a mapping function ϕ, mapping the original input space to the feature space. The dual form (Equation 2.15) of our optimization problem then becomes:

$$\arg\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,g=1}^{n} \alpha_i \alpha_g y_i y_g \phi(\boldsymbol{x}_i)^T \cdot \phi(\boldsymbol{x}_g), \tag{2.22}$$

subject to $\alpha_i \geq 0$ and Equation 2.15. Note that the above problem depends on the data only through dot products in feature space. If the kernel function $K(\boldsymbol{x}_i, \boldsymbol{x}_g)$ defined as

$$K(\boldsymbol{x}_i, \boldsymbol{x}_g) = \phi(\boldsymbol{x}_i)^T \cdot \phi(\boldsymbol{x}_g) \tag{2.23}$$

can be computed efficiently, then we can avoid the computationally expensive step by skipping the explicit mapping of the data to a higher dimensional feature space.

To do this, we need to verify that kernel function K is indeed an inner product. Given a function K, instead of trying to find the explicit representation for ϕ, Mercer's theorem [77] gives another way of testing if it is a valid kernel. In a finite feature space, a symmetric matrix of all possible $K(\boldsymbol{x}_i, \boldsymbol{x}_g)$ values can be defined and shown to be positive semi-definite if and only if K is a valid kernel function. There are two desirable properties when choosing a kernel function in an application. Firstly, the kernel should capture the similarity between implicit representations of data in feature space. Secondly, it should require less computation than the explicit calculation of the corresponding feature mapping ϕ.

SVM-Light was used for the work, which supports four kernel functions: the *linear* kernel, the *polynomial* kernel, the *radial basis function* kernel and the *sigmoid tanh* kernel. For each kernel function, there are associated parameters which must be set.

Linear kernel

The linear kernel function is the simplest kernel, which represents a simple scalar product of the two feature vectors:

$$K(\boldsymbol{x}_i, \boldsymbol{x}_g) = \boldsymbol{x}_i^T \cdot \boldsymbol{x}_g.$$

where $\boldsymbol{x}_i$ and $\boldsymbol{x}_g$ are feature vectors. It does not require any user supplied parameters and usually only performs well when the data is close to being linearly separable.

Polynomial kernel

The polynomial kernel represents the similarity of training samples in a feature space over polynomials of the original variables:

$$K(\boldsymbol{x}_i, \boldsymbol{x}_g) = (\gamma \boldsymbol{x}_i^T \cdot \boldsymbol{x}_g + r)^d, \gamma > 0$$

where $\boldsymbol{x}_i$ and $\boldsymbol{x}_g$ are feature vectors and γ, r and d are kernel parameters.

Radial basis function kernel

The radial basis function kernel, also called the Gaussian kernel, is a polynomial kernel of infinite degree. Its features are all possible monomials of input features with no degree restriction:

$$K(\boldsymbol{x}_i, \boldsymbol{x}_g) = \exp(-\gamma ||\boldsymbol{x}_i - \boldsymbol{x}_g||^2), \gamma > 0$$

where $\boldsymbol{x}_i$ and $\boldsymbol{x}_g$ are feature vectors and γ is kernel parameter.

Sigmoid tanh kernel

The sigmoid tanh kernel takes the tanh of a scaled and shifted scalar product, with the general expression

$$K(\boldsymbol{x}_i, \boldsymbol{x}_g) = \tanh(\gamma \boldsymbol{x}_i^T \cdot \boldsymbol{x}_g + r)$$

where $\boldsymbol{x}_i$ and $\boldsymbol{x}_g$ are feature vectors and γ and r are kernel parameters.

Earlier studies suggest that the radial basis function (RBF) kernel is in general a reasonable first choice [57]. Keerthi and Lin [65] showed that if complete model selection using the RBF kernel has been conducted, there is no need to consider linear SVM as the linear kernel is a special case of RBF. In addition, both theoretical and experimental analysis demonstrated that the sigmoid kernel behaves like RBF for certain parameters, but not better than RBF in general [71]. Moreover, an earlier experiments applying machine learning to an automated theorem prover [13] also found the radial basis function (RBF) kernel performed well in finding a relation between the simple algebraic features and the best heuristic choice. Hence the radial basis function (RBF) kernel was selected in the work of this dissertation. The performance of RBF kernel highly depends on the choice of parameters. Compared to Sigmoid tanh kernel and polynomial kernel, the RBF kernel only has a single parameter, which also reduces the complexity of the model selection. I will discuss the parameters of the RBF kernel and parameter optimization method for selecting the optimal parameters in Section 3.5.

2.2.1.6 Soft margin classifier

In Section 2.2.1.2, we discussed linear classifiers by assuming that the data are linearly separable. In Section 2.2.1.5, kernel methods were introduced to help with nonlinear cases. However, even with a transformation of the feature space, some data sets may not be linearly separable. Additionally, the data set itself may contain noisy data, generally referred to as *outliers*, which are far away from any expected position. With the previously described hard margin SVM model, the classifier is very susceptible to outliers since the location of the hyperplane is restricted only by a few support vectors. If some of the outliers are support vectors they will heavily influence the classifier.

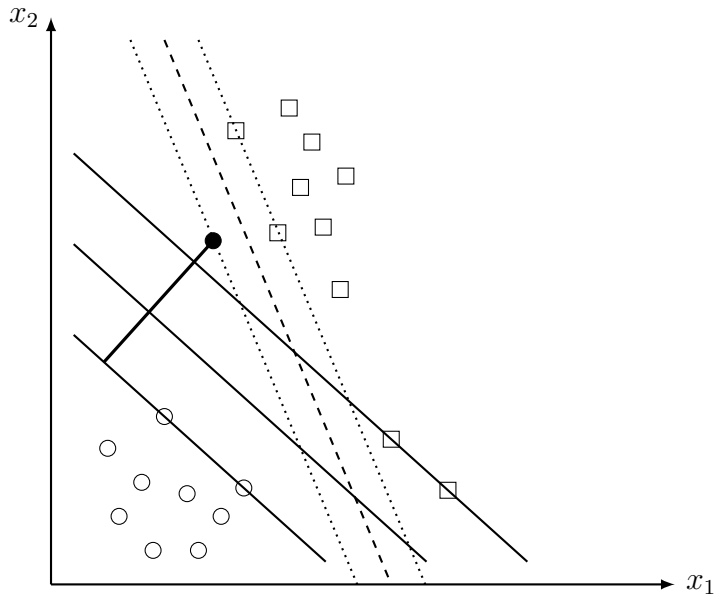


Figure 2.2: Outlier in the dataset

In Figure 2.2, squares are positive examples and circles are negative examples. The solid circle is an outlier. If we ignore the example, we could easily find a reasonable hyperplane (the solid line in the middle) to separate the rest of the points. However, with the outlier, the decision boundaries become the narrow dashed lines shown in the figure, and the margin value is much smaller. Even worse, if the outlier is even further to the upper right, there would be no hyperplane that can separate the dataset.

In order to deal with this situation, we modify the previously described hard margin SVM to allow some sample points to lie within a certain distance or even on the wrong side of the decision boundary. This approach is called the *soft margin* approach. To do this, slack variables, $\xi_i \geq 0$, where $i = 1, ..., n$ are introduced, with one slack variable for each training data point. The previous condition

$$y_i(\boldsymbol{\omega}^T \cdot \boldsymbol{x}_i + b) \geq 1 \tag{2.24}$$

is then changed to

$$y_i(\boldsymbol{\omega}^T \cdot \boldsymbol{x}_i + b) \geq 1 - \xi_i \tag{2.25}$$

Of course, if the slack variable ξ_i is large enough, any hyperplane could satisfy the condition. Ideally the number of points with non-zero values for the slack variables should be minimised. Thus the new optimisation problem becomes:

$$\arg\min_{\gamma,\boldsymbol{\omega},b} \frac{1}{2}\|\boldsymbol{\omega}\|^2 + C\sum_{i=1}^{n} \xi_i \tag{2.26}$$

subject to:

$$y_i(\boldsymbol{\omega}^T \boldsymbol{x}_i + b) \geq 1 - \xi_i. \tag{2.27}$$

where the parameter C represents a trade-off between training error and margin. In SVMLight, the parameter C is split into two to allow for unbalanced sets [78]. The

target function is changed to

$$\arg\min_{\gamma,\boldsymbol{\omega},b} \frac{1}{2}\|\boldsymbol{\omega}\|^2 + C_+ \sum_{i:y_i=1}^{n} \xi_i + C_- \sum_{g:y_g=-1}^{n} \xi_g \tag{2.28}$$

The SVMLight parameter j (equal to 1 by default) is used to set the ratio of C_+ to C_- and typically should be equal to the ratio of the number of negative samples to the number of positive samples in the training set:

$$j = \frac{C_-}{C_+} = \frac{\text{number of negative training examples}}{\text{number of positive training examples}} \tag{2.29}$$

2.2.2 Multi SVMs

Since SVM methods are binary, we have to do more work in the case of multi-class problems: either determine all the decision functions at once or reduce the problem to a set of multiple binary classification problems [8, 104]. One of the most commonly used multiclass classifiers is known as the *one-versus-all* approach. Suppose the dataset is classified into K classes. Then, K binary SVM classifiers are created where each classifier is trained to distinguish one class from the remaining $K-1$ classes. During the testing phase, the test sample is placed in the class giving the largest $f_k(\boldsymbol{x})$ (most positive or least negative) value, where $f_k(\boldsymbol{x})$ is the solution from the kth SVM classifier ($k = 1, \ldots, K$). In my thesis, the one-versus-all approach is used. This *one-versus-all* approach has the advantage that the number of binary classifiers to construct equals the number of classes. However, it also suffers from the problem that the different classifiers contain unbalanced data. Suppose for K classes, that each has an equal number of training samples. During the learning phase, the ratio of training samples of one class to rest of the classes will only be $\frac{1}{K-1}$. This ratio, therefore, shows that training sample sizes will be unbalanced. This problem is addressed by the j parameter in SVMLight, which allows separate weights to be applied to positive and negative slack variables during the optimisation.

2.2.3 Feature selection

Feature selection is the process of selecting a subset of relevant features in the training set and using only this subset as features in the classification task. In itself, it is a broad and important area in machine learning, and I give here the relevant background to understand the tasks for which I use feature selection methods. For further background, we refer to [67, 49, 50].

Feature selection techniques are used for four main purposes.

1. To shorten the learning time.

2. To increase classification accuracy by eliminating noisy features.

3. To enhance generalisation by removing irrelevant features (as many irrelevant features in the dataset can result in over fitting).

4. To provide a better understanding of the underlying process from which the data was generated.

A feature selection algorithm generally consists of a search technique for suggesting feature subsets, along with an evaluation measurement for the feature subsets. The simplest algorithm is to test each possible subset of features and to find the one with the minimum generation error. This is an exhaustive search of the space, and is computationally intractable for most problems. It is therefore necessary to employ approaches to explore only a part of the potential search space. There are two main approaches that deal with feature selection: the *wrapper approach* and the *filter approach* [67].

Wrapper method evaluates subset of features according to their performance (e.g. accuracy) to a given predictive model. Wrapper In the following figures I present a detailed view of the features used in the experiments described in Chapters **??** and **??**. Figures **??** show the distribution of features of the data set used in Chapter **??** in form of histograms, while Figures **??** show the distribution of features of the data in Chapter **??**. In addition, I examined some of the pairwise correlations present between features in both sets. In Figures **??** I present the projection of the data set onto a number of feature pairs. For some feature pairs we can observe a noticeable correlation, which is often simply explained by the nature of the feature (e.g. the proportion of x occuring in polynomials is likely to be correlated with its occurance in monomials). For others, no obvious correlation exists, which adds to justify the inclusion of both features in the set.methods requires to train a new model for each subset, and thus are very computationally expensive. However, they usually provide the best performing feature set for that particular type of model.

Unlike the wrapper approach, the filter approach is independent of the classifier used. The characteristics of the training data are used to select feature subsets (for example, the correlation between features and class, or the redundancy between features). Filter methods are usually faster than wrappers, but they produce a feature set which is not tuned to a specific predictive model. Filter methods are commonly used as a preprocessing step for wrapper methods, allowing a wrapper to be used on larger problems. For example, in applications such as the text processing, the number of features may reach tens of thousands. Directly applying wrapper methods could be very slow and even infeasible for the problem. In the work described in Chapter 5, the number of features is 28, which allows the use of both wrapper and filter approaches and their results are compared.

Chapter 3

Choosing decision procedures and time limits in MetiTarski

The goal of this thesis is to examine how machine learning can be applied to the field of computer algebra. I conducted three machine learning experiments to determine the usefulness of machine learning in computer algebra applications over RCFs. All three share some commonalities in their methodology. In this chapter, I describe the experiments of applying machine learning to the problem-dependent selection of the best decision procedure and selection of the best time limit setting on individual RCF problems in the theorem prover MetiTarski. The experiments mainly served as preliminary experiments to determine whether machine learning was work at all in the given application area. At the same time, we introduce here some aspects of methodology that we reuse for the other experiment, such as the performance measure for classifiers, the selection of kernels and parameter values, and the combination of classifiers for multi-class SVMs.

3.1 Decision produces

During execution, MetiTarski reduces terms with special functions to polynomial inequalities over RCFs and relies on external decision procedures to solve these inequalities. As discussed in Section 2.1.6, different implementations of RCF decision procedures have their own strengths and weaknesses for restricted classes of formulas and there is no single RCF decision procedure which is optimal for all problems. Which decision procedure is the most appropriate one to call is highly dependent on the (usually geometric) properties of the RCF problem being considered, which is in turn largely influenced by the structure of the original MetiTarski problem. However, the relationship between MetiTarski problems and the preferred decision procedure is far from obvious. I want to find out whether machine learning can help with the choice of the appropriate decision procedure for a given problem.

Three RCF decision procedures were tested: Z3 with Strategy 1 (the default option) [83], Mathematica [108] (specified by the -m option) and Qepcad [15] (specified by the -qepcad option). For convenience, in the rest of the chapter, I use Z3 to denote Z3 with Strategy 1. However, it should be clear that it is a non-standard version of Z3.

Technical Note: all computations were performed in Qepcad 1.62, Mathematica 8.0.1 and Z3 4.0 on a 2.4GHz Intel processor. This same machine and softwares were used for the experiments in Section 3.8.

3.2 Evaluation of decision procedures

For each problem, I called MetiTarski with each of the decision procedures. In addition, I recorded the running time for each problem with each decision procedure. A time limit of 60 seconds was set for each proof attempt. The best decision procedure is the one with the fastest runtime. We use a threshold in this comparison, counting a proof as the fastest only if it is faster by a 1% margin than all other proofs found. If more than one decision procedure yielded the minimal runtime, both decision procedures were considered the best. If none of them finished within the time limit, none of the decision procedures were considered the best. This approach is based on Bridge et al.'s work on machine learning for first-order theorem proving [14].

3.3 Problem features

To apply machine learning, we need to identify features of the MetiTarski problem that might be relevant to the correct choice of the RCF decision procedure. A feature is an aspect or measure of the problem that may be expressed numerically. I characterised MetiTarski problems by a vector of real numbers or features. For each problem, each vector of features was associated with label $+1$ or -1, indicating in which of two classes it was placed. Take Z3 as an example: in the learning set for Z3, each problem is labelled $+1$ if Z3 found a proof and was the fastest to do so, or -1 if Z3 failed to find a proof or was not the fastest.

The features used for the current experiment were of two types. One consisted of the indication of various *special functions*: ln, sin, cos, etc. Each feature represented the presence of the specified function in the problem. For example, the feature value related to the function ln was equal to 1 if ln appeared in the given problem, otherwise the value was equal to 0. (Here we get a special case that π is also counted for one feature, since in MetiTarski, instead of having a constant value, π is processed in the same way as other special functions.) The other type of feature was related to the number of variables in the given MetiTarski problem. There are four features, indicating whether the problem involves 0, 1, 2, or more variables. I chose this representation for two reasons. First, it has the same boolean nature as the other feature type. Second, most of the problems have no more than two variables in the problem set. Having more variables in a problem will greatly increase the difficulty of the proof search as CAD is doubly exponential in the number of variables. Decision procedures usually return quickly if the formula has only a few variables. Table 3.1 shows the 22 features that were identified in all problems. Figure 3.1 shows a sample problem in our problem set. The problem would yield the following feature vector:

$$[0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0].$$

Here, each component corresponds to a feature from Table 4.1. and this vector encodes the presence of the special functions exp, sin, and cos, and the number of variables, which is equal to one.

Table 3.1: Description of the features used.

Feature number	Description
1	Presence of the abs function
2	Presence of the arcsin function
3	Presence of the arctan function
4	Presence of the cbrt function
5	Presence of the cos function
6	Presence of the cosh function
7	Presence of the exp function
8	Presence of the ln function
9	Presence of the log function
10	Presence of the max function
11	Presence of the min function
12	Presence of the pi function
13	Presence of the pow function
14	Presence of the sin function
15	Presence of the sinh function
16	Presence of the sqrt function
17	Presence of the tan function
18	Presence of the tanh function
19	Number of variable is equal to zero
20	Number of variable is equal to one
21	Number of variable is equal to two
22	Number of variable is large than two

3.4 Performance measures for classifiers

For each SVM classifier, we need to have a means of measuring its performance. Performance measures are useful in judging the effectiveness of the learning and help to determine the best parameter values to set.

Below, I discuss some standard measures. I will first introduce four basic common measures given in the *confusion matrix* for a two-class classifier (see Table 3.2).

Several standard terms are defined from the confusion matrix: The *accuracy (ACC)* is the proportion of the total number of predictions that were correct:

$$ACC = \frac{TP + TN}{TP + FP + FN + TN}$$

The *recall* is the proportion of positive cases that were correctly identified:

$$recall = \frac{TP}{TP + FN}$$

Figure 3.1: Sample MetiTarski problem: Chua-1-VC1-L-sincos.tptp

```
fof('Chua', conjecture, ! [X] : ((0 <= X & X <= 289)
=> 2.84 - 0.063*exp(-0.019*X) - 1.77*exp(0.00024*X)
*cos(0.0189*X) + 0.689*exp(0.00024*X)*sin(0.0189*X) > 0)).

include('Axioms/general.ax').
include('Axioms/exp-upper.ax').
include('Axioms/exp-lower.ax').
include('Axioms/sin.ax').
include('Axioms/cos.ax').
```

Table 3.2: Confusion matrix for a two class classifier

		Predicted Class	
		Positive	**Negative**
Actual Class	**Positive**	True positives (TP)	False negatives (FN)
	Negative	False positive (FP)	True negatives (TN)

The *precision* is the proportion of the predicted positive cases that were correct:

$$precision = \frac{TP}{TP + FP}$$

The *false positive rate (FPR)* is the proportion of negatives cases that were incorrectly classified as positive:

$$FPR = \frac{FP}{FP + TN}$$

The *true negative rate (TNR)* is defined as the proportion of negatives cases that were classified correctly:

$$TNR = \frac{TN}{TN + FP}$$

The *false negative rate (FNR)* is the proportion of positives cases that were incorrectly classified as negative:

$$FNR = \frac{FN}{TP + FN}$$

From the confusion matrix, the main performance measure is the ACC measure. However, it may not be a reliable performance measure when applied to problems having unbalanced classes. For example, if there were 100 MetiTarski problems in total and in only 5 of them did Z3 fail to make the most efficient proof, the classifier could easily be biased into classifying all the samples as positive labels. The overall accuracy would be 95%, but in practice the classifier would have a 100% recognition rate for the positive class but a 0% recognition rate for the negative class.

Other performance measures are considered good when we encounter unbalanced data set. For example, a good measurement for evaluating the performance of binary classifications is *Matthew's correlation coefficient (MCC)* [5], which takes into account all the four basic measurements:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

where the denominator is set to 1 if any sum term is zero. This measure has the value 1 if perfect prediction is attained, 0 if the classifier is performing as a random classifier, and -1 if the classifier exactly disagrees with the data.

Finally we define another commonly used measure: the F_1-score, which is defined in terms of *precision* and *recall*

$$F_1 = \frac{2 \times precision \times recall}{precision + recall}.$$

The F_1-score reaches its best value at 1 and worst score at 0. We note that F_1-score does not take into account the value of TN.

The F_1-score and Matthew's coefficients are the main measurements I used in the experiment to determine the best parameters during the learning process. Note that the final performance is the joint performance of the set of SVM classifiers (see Section 3.6). To determine the efficacy of the machine learned selection process, the number of problems successfully proved was used and compared to the number of problems proved by always fixing one individual decision procedure.

3.5 Kernel selection and parameter optimization

As stated in Section 2.2.1, SVMs use kernel functions to map the data into higher dimensional spaces where the data may be more easily separated. The application of SVMs to any classification problem requires the choice of an appropriate kernel and its associated parameters, and the accuracy of the learning model is largely dependent on them. The radial basis function (RBF) kernel was selected for this experiment. The performance of RBF kernel highly depends on the choice of parameters. In order to optimize the associated parameter setting, I used a validation set alongside the training set. The basic idea is to choose the kernel and parameters that yield the classifier that maximises some performance measure on the validation set; in this case the accuracy, the Matthews coefficient, or the F_1-score. The actual experiment is then assessed on a test set that was not used in either training or validation.

There are three parameters involved in RBF kernel function. The first one is the weight factor j, which is used to correct the imbalance in the training set and was set to the ratio between the number of negative samples and the number of positive samples. Besides the parameter j, two other parameters are involved in the SVM fitting process. The parameter γ determines how far the influence of a support vector reaches. The behaviour of the RBF kernel is very sensitive to the γ parameter: when γ is too large, the region of influence of the support vector is possibly limited to the support vector itself. However, when γ is too small, the region of influence of any selected support vector would include the whole training set. Thus a small γ will produce low bias and high variance results, while a large γ will give higher bias and low variance results. The parameter C

governs the trade-off between margin and training error. It defines how much penalty the SVM optimizer receive for each misclassified training example. When C is set to a large value, the optimizer will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly, whereas a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points.

The choice of optimal values for γ and C is not trivial. The parameter selection process needs to be done so that the classifier can accurately predict testing data. As discussed above, a common setting for the experiment is to separate the data set into three parts, with approximately half of the problems placed in a learning set, a quarter put in a validation set used for kernel selection and parameter optimization, and the final quarter retained in a test set used for judging the effectiveness of the learning.

To choose γ and C, a grid-search optimisation procedure [57] was used with the training and validation set, involving a search over a range of (γ, C) values to find the pair which would maximize the score of the corresponding performance (accuracy, Matthews coefficient or F_1 score). I tested a wide range of values of γ (varied between $2^{-15}, 2^{-14}, 2^{-13}, \ldots, 2^3$) and C (varied between $2^{-5}, 2^{-4}, 2^{-3}, \ldots, 2^{15}$) in the grid search process (the ranges are suggested by [57]). Following the completion of the grid-search, the values for γ and C giving optimal performance scores were selected.

3.6 Combining classifiers for choosing decision procedures

SVMs were originally designed for binary classification. However, many real-world problems have more than two classes. For example, in this experiment, we have three classes to represent three decision procedures (Z3, QEPCAD and MATHEMATICA). I use *one-versus-all* to represent these classes. Membership in the positive class implies that the decision procedure represented by this classifier is the most efficient one among the three for the given problem. Membership in the negative class implies that the decision procedure represented by this classifier did not terminate within the given time limit or is not the most efficient one. If each predictive classifier obtained by the samples in the learning set was perfect, then for any problem from the test set only one classifier would place it in the positive class and all the other classifiers would place it in the negative class. Then the best decision procedure would be the one for which the problem was classified in the positive class. However, in practice, more than one classifier will return a positive result for some problems, while for others, no classifiers may return a positive.

Assuming that the requirement is to select only one decision procedure as the best choice, we want to find a way to select one even in these ambiguous cases. Our choice here relies on the margin value. The margin for a single sample is a measure of the distance of the sample to the decision line. In SVMLIGHT, the output value from a classifier is not simply a class label ("+1" or "−1"), but the margin value. The classifiers with optimal parameter settings were applied to the test set to output the margin values for each classifier. A large margin represents high confidence in a correct prediction. More specifically, a large positive margin represents high confidence for the problem being classified in a positive class, while a large negative margin represents high confidence for the problem being classified in a negative class. Thus the relative magnitudes of the

classifiers were considered in the experiment. The classifier with most positive (or least negative) margin was selected to indicate the best decision procedure for the selection.

3.7 Results

The experiment was done on 825 MetiTarski problems in a variant of the Thousands of Problems for Theorem Provers (TPTP) format [98]. They were taken from the entire MetiTarski problem set at the time the work was done. The only filtering applied was to omit the existential problems, of which there are 39, because none of the three decision procedures works for them. The data was randomly split into three subsets, with approximately half of the problems placed in a learning set (418 problems), a quarter put in a validation set (213 problems) used for kernel selection and parameter optimization, and the final quarter retained in a test set (194 problems) used for judging the effectiveness of the learning. The data was not *stratified* (the ratio of number of positive samples and negative samples differs in each data set) at the time when the experiment was conducted. Though the ratio of the samples were not strictly enforced, it was found the actual difference of the ratio is within 10% percentage. While this may affect the exact makeup of the learning and test populations, it does not materially affect the validity of the results, given the small variation. Since this was a preliminary experiment, it was not re-run with corrected software. However, I address this issue in my other two experiments. The learning set and validation set were combined and used for learning after parameter optimization process, the learned model was then tested on the separate test set.

The total number of problems proved out of 194 testing problems was used to measure the efficacy of the machine learned selection process. The key question was whether or not the overall selection process does better than assigning a problem to any fixed individual decision procedure. Thus the learned selection was compared with using each of the decision procedures on their own. The selection results are given in Table 3.3.

Table 3.3: Number of problems proved

Decision procedures	Number of problems	Percentage of test set
Machine Learning	163	84%
Z3	160	82%
QEPCAD	153	79%
Mathematica	158	81%

Using machine learning to select the best decision procedure yields better results than any fixed individual decision procedure. Though the improvement is only marginal, it is clear that the machine learned algorithm works in the given context. Note that there is an upper limit for our benchmark set which corresponds to the case when the best decision procedure is always selected for each problem. Such a perfect algorithm would prove 172 out of 194 test problems (there are 22 problems in the test set, which cannot be proved by any of the three decision procedures within time limit). We see there is still room for improving the selection process.

3.8 Time limits on RCF decision procedure calls

In MetiTarski's proof search, the RCF tests typically dominate the overall processor time, while the time spent in resolution is much smaller. Only unsatisfiable RCF subproblems contribute to a MetiTarski proof. If the RCF time limit is too long, some problems may not be solved because the RCF decision procedure wastes too much time trying to prove the satisfiability of irrelevant RCF subproblems. By limiting the amount of time on individual RCF problems, MetiTarski can minimise the time wasted on difficult RCF problems and focus on easier ones, potentially moving the proof forwards. However, if the RCF time limit is too short, important clause simplification steps may be missed when RCF calls time out. The next experiment was to apply machine learning to the problem-dependent selection of the best time limit setting for RCF decision procedure calls in MetiTarski.

This experiment is similar to the previous one. Two RCF decision procedures, Z3 [34] and Mathematica [108] were used. Machine learning was applied to select which decision procedure to use and which time limit to set on RCF calls for a given MetiTarski problem. In each individual run, MetiTarski called Z3 or Mathematica with various time limits on RCF calls: 0.1s, 1s, 10s, or 100s. A time limit of 100 CPU seconds was set for each proof attempt. For each MetiTarski problem, the best setting to use was determined by running Z3 or Mathematica with various RCF time limits and choosing the one which gave the fastest overall runtime. Each MetiTarski problem was characterised by a vector of real numbers or features. The features used for this experiment were the same as in Table 3.1. Each vector of features was associated with label $+1$ or -1, indicating in which of two classes it was placed. Taking the setting of Z3 with 1s RCF time limit as an example, a corresponding learning set was derived with each problem labelled $+1$ if Z3 with 1s RCF time limit found a proof and was the fastest to do so, or -1 if it failed to find a proof in that time limit, or was not the fastest.

The experiment was done on the same 826 MetiTarski problems (one new problem was added at the time the experiment was conducted), except a new split was imposed on the data set. The data was randomly split into a learning set (414 problems), a validation set (204 problems) and a test set (208 problems). The total number of problems proved out of 208 testing problems was used to measure the efficacy of the machine learned selection process. The learned selection was compared with fixed RCF time settings. The selection results are given in Table 3.4. We can see that the machine learned algorithm for selection performs better on our benchmark set than any fixed individual settings used in isolation.

3.9 Summary

Machine learning was applied to the problem-dependent selection of the most efficient decision procedure and the selection of the best time limit setting for RCF decision procedure calls in the theorem prover MetiTarski. The machine learned selection yielded better results than any individual fixed option (both for the case of decision procedure selection and time limit setting). Though the improvement is only marginal, the machine learned algorithm for selection performs better on our benchmark set. The results are promising for continuing work in the area. Some of the methodology used here is also used in the following experiments. However, there were some aspects that were improved

Table 3.4: Selection results for time limit setting

Time limit setting	Number of problems	Percentage of test set
Machine Learning	176	85%
Z3 (0.1s)	164	79%
Z3 (1s)	171	82%
Z3 (10s)	168	81%
Z3 (100s)	168	81%
Mathematica (0.1s)	143	69%
Mathematica (1s)	162	78%
Mathematica (10s)	172	83%
Mathematica (100s)	170	82%

in the later experiments. As this was a first preliminary experiment for exploration of the field, I chose not to rerun it in the improved setting.

One improvement was to use a 5-fold cross validation process instead of holding out samples for training and validation sets. This approach avoids holding out some data for validation and thus makes full use of the training set for the optimization of parameters. Another improvement would be to normalize the features to zero mean and unit variance. And also we noticed the improvement is marginal, it would be good to conduct more repetitions of runnings with randomly partitioned data, which allows for better measuring the significance of the machine learning results.

Exploring more feature types relevant to the classification process is also essential for improving the efficacy of the machine learned selection process. Future work could be done by following this line. Possible features could be the number of atomic formulas, the number of symbols and so on. Also, a more sophisticated feature selection process could be applied given many features. Analysing the results of the feature selection process also gives some insight as to why some decision procedures or heuristics perform better than others in certain problem cases, which can help with developing new decision procedure or heuristics.

Chapter 4

Choosing a heuristic for CAD variable ordering

In the computation of a CAD, the order in which the variables are projected during the projection phase plays a significant role: some problems are even infeasible in one ordering, while easily solvable in another. There are various heuristics for selecting a suitable variable ordering given an instance, and no single one is suitable for all problems. The choice of the best heuristic depends on the problem considered. However, this relationship between problems and heuristics is far from obvious. To tackle this problem, I applied machine learning for the task of selecting a variable ordering for both CAD itself and quantifier elimination using CAD, utilising the `nlsat` dataset [79] of fully existentially quantified problems.

I have already covered the relevant background on CAD and machine learning in Chapter 2. In the rest of this chapter, I will describe the methodology of the experiment, comparing three heuristics and analyse the results. At the end, I will give a summary and ideas for future work.

4.1 CAD implementation

For this experiment, we focus on a single CAD implementation, namely QEPCAD [16]. QEPCAD was chosen as it is a competitive implementation of both CAD and quantifier elimination. QEPCAD was used with its default settings, which implement McCallum's projection operator [74] and partial CAD [27].

4.2 Heuristics

In the experiment, three existing heuristics for picking a CAD variable ordering were used:

Brown: This heuristic chooses a variable ordering according to the following criteria, starting with the first and breaking ties with successive ones:

(1) Eliminate a variable first if it has lower overall degree in the input.

(2) Eliminate a variable first if it has lower (maximum) total degree in those terms in the input in which it occurs.

(3) Eliminate a variable first if there is a smaller number of terms in the input which contain the variable.

It is named after Brown, who suggested it in [17].

sotd: This heuristic constructs the full set of projection polynomials for each permitted ordering and selects the ordering whose corresponding set has the lowest sum of total degrees for each of the monomials in each of the polynomials. It is labelled `sotd` for *sum of total degree* and was suggested by Dolzmann, Seidell and Sturm [36], whose study found it to be a good heuristic for both CAD and quantifier elimination by CAD.

ndrr: This heuristic constructs the full set of projection polynomials for each ordering and selects the ordering whose set has the lowest number of distinct real roots of the univariate polynomials within. It is labelled `ndrr` for *number of distinct real roots* and was suggested by Bradford et al. [11]. `Ndrr` was shown to assist with examples where `sotd` failed.

All three heuristics may identify more than one variable ordering as a suitable choice. In this case, we selected the alphabetically first one.

4.3 Data

Problems were taken from the nlsat dataset [79], which I chose over more traditional CAD problem sets (such as Wilson et al. [107]) as the latter have an insufficient number of problems to suitably apply machine learning. In addition, I chose to restrict the data set to instances with only three variables. This has two reasons: first, since we have only a small number of variables, it is feasible to test all possible variable orderings. Secondly, we avoid the possibility that QEPCAD will produce errors or warnings related to well-orientedness with the McCallum projection [74]. Out of the set, 7001 three-variable CAD problems were extracted for the experiment.

Two experiments were undertaken, one applying machine learning to CAD itself, and another to quantifier elimination using CAD. These two experiments are separate, since for quantified problems QEPCAD can use partial CAD techniques to stop the lifting process early if the outcome is already determined, while for unquantified ones the full process is completed; the two outputs can be quite different.

The problems from the `nlsat` dataset are all fully existential (satisfiability or SAT problems). A second set of problems for the quantifier-free experiment was obtained by simply removing all quantifiers. An example of the QEPCAD input for a SAT problem is given in Figure 4.1 with the corresponding input for the unquantified problem in Figure 4.2. The first line declares three variables in the input problem, the second line indicates the number of free variables in the problem. For example, there are zero free variables in the quantified case and three free variables in the quantifier-free case. The next lines show the commands that were used to calculate the cell counts, which are relevant for the evaluation of the different heuristics. Note that in the quantified case, QEPCAD can collapse stacks when sufficient truth values for the constituent cells have been discovered to determine a truth value for the base cell. Hence, since our problems are all fully existential, the output for all quantified problems is always a single cell: true or false. In

these cases we are not interested in the number of cells in the output, but rather the total number of cells constructed during the process. Therefore, the commands in Figures 4.1 and 4.2 differ: for the quantified problems, QEPCAD uses the `d-stat` command following construction to obtain the number of cells constructed in the partial CAD; while in the quantifier-free case, `d-fpc-stat` is used to compute the number of cells produced in the CAD of $\mathbb{R}^3$.

For quantified problems there are better alternatives to building a CAD (see for example the work of Jovanovic and de Moura [63]). However, the decision to use only SAT problems was based on the availability of data. An advantage to this choice is that a fully existential or fully universal quantification allows for all six possible variable orderings. Future work may include expanding the experiments to consider mixed quantifiers.

Figure 4.1: Sample QEPCAD input for a quantified problem.

```
(x0,x1,x2)
0
(E x0)(E x1)(E x2)[[((x0 x0) + ((x1 x1) + (x2 x2))) = 1]].
go
go
go
d-stat
go
finish
```

Figure 4.2: Sample QEPCAD input for a quantifier-free problem.

```
(x0,x1,x2)
3
[[((x0 x0) + ((x1 x1) + (x2 x2))) = 1]].
go
go
d-proj-factors
d-proj-polynomials
go
d-fpc-stat
go
```

4.4 Evaluating the heuristics

Since each problem has three variables and all the quantifiers are of the same kind, all six possible variable orderings are admissible. For each ordering, we had QEPCAD build a CAD and then counted the number of cells. The best ordering was defined as the one resulting in the smallest cell count. If more than one ordering gave a minimal count, both orderings were considered best. The decision to focus on cell counts (rather than say computation time) was made so that the experiment could validate the use of machine

learning to CAD theory, rather than just the QEPCAD implementation. That is to say, the cell count is a structural property of the resulting CAD, rather than a property of the specific CAD implementation. Furthermore, it is usually the case that cell counts and timings are strongly correlated.

The heuristics (`Brown`, `sotd` and `ndrr`) were implemented in MAPLE by England [40] and for each problem the orderings suggested by the heuristics were recorded and compared to the cell counts produced by QEPCAD. Note that none of three heuristics takes into account the quantifier structure of the problem, but rather work on only properties of the polynomials. As discussed above, some heuristics are more expensive than others. However, since none of the costs were prohibitive for our data set the cost of an heuristic is not considered here.

Finally, the machine learning task was to predict which of the three heuristics will give an *optimal* variable ordering for a given problem, where *optimal* means that it produced the lowest cell count in the resulting CAD.

4.5 Problem features

To apply machine learning, we need to identify features of the CAD problem that may be relevant to the correct choice of heuristic. Table 4.1 shows the 11 features that we identified, where (x_0, x_1, x_2) are the three variable labels used in all our problems. The features were chosen as easily computable properties which might affect the performances of the heuristics.

Table 4.1: Description of the features used. The proportion of a variable occurring in polynomials is the number of polynomials containing the variable divided by total number of polynomials. The proportion of a variable occurring in monomials is the number of terms containing the variable divided by total number of terms in polynomials.

Feature number	Description
1	Number of polynomials.
2	Maximum total degree of polynomials.
3	Maximum degree of x_0 among all polynomials.
4	Maximum degree of x_1 among all polynomials.
5	Maximum degree of x_2 among all polynomials.
6	Proportion of x_0 occurring in polynomials.
7	Proportion of x_1 occurring in polynomials.
8	Proportion of x_2 occurring in polynomials.
9	Proportion of x_0 occurring in monomials.
10	Proportion of x_1 occurring in monomials.
11	Proportion of x_2 occurring in monomials.

Each feature vector in the training set was associated with a label, $+1$ or -1, indicating in which of two classes it was placed. To take `Brown` heuristic as an example, a corresponding training set was derived with each problem labelled $+1$ if `Brown` heuristic suggested a variable ordering with the lowest number of cells, or -1 otherwise.

The features could all be easily calculated from the problem input using MAPLE. For example, if the input formula is defined as the set of polynomials

$$\{-6x_0^2 - x_2^3 - 1, \quad x_0^4 x_2 + 9x_1, \quad x_0 + x_0^2 - x_2 x_0 - 5\}$$

then the problem will have the feature vector

$$\left[3, 5, 4, 1, 3, 1, \frac{1}{3}, 1, \frac{5}{9}, \frac{1}{9}, \frac{1}{3}\right].$$

After the feature generation process, the training data (feature vectors) were normalized so that each feature had zero mean and unit variance across the set. The same normalization was then also applied to the validation and test sets.

4.6 Parameter optimization

We used SVM-LIGHT with a RBF kernel to do the classification for this experiment. As discussed in Section 3.5, given a training set, we can easily compute the value of parameter j by looking at the ratio of the number of negative samples to the number of positive samples. However, it is less trivial to find the optimal values of γ and C. In order to determine good values for these parameters, we employed a five-fold cross validation [66].

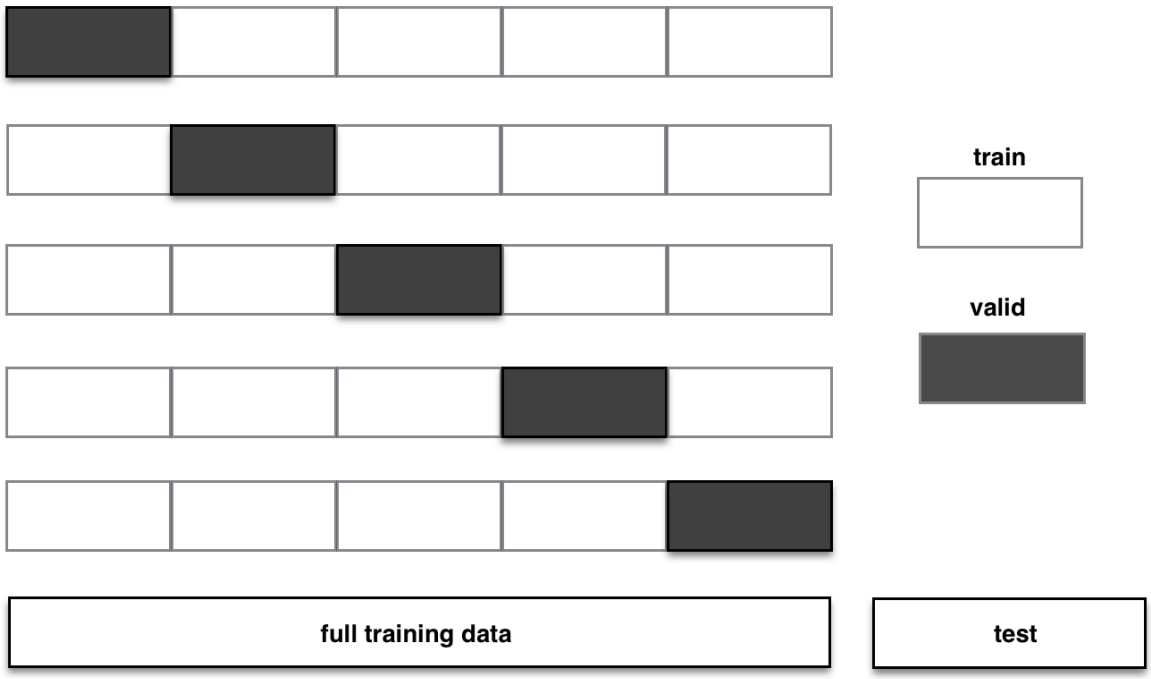
The overall data was partitioned into two sets, with approximately 75% of the problems placed into a training set (5280 problems) and 25% of them retained in a test set (1721 problems). Both sets were *stratified* to maintain relative class proportions (i.e. in each partition, the ratio between positive and negative samples is the same). For the five-fold cross validation we further partitioned the training set into five subsets of equal size that all contain the same number of positive (+1) and negative (-1) samples. We then perform five runs, where in each run, one subset is used as the validation set, while the remaining four subsets are combined and used as the training set. As in Section 3.5, a grid-search optimisation procedure was used on each run to determine the best parameter setting for that run. The grid-search optimisation procedure involves a search over a range of (γ, C) values, where γ varied between $2^{-15}, 2^{-14}, 2^{-13}, \ldots, 2^3$, and C varied between $2^{-5}, 2^{-4}, 2^{-3}, \ldots, 2^{15}$. The score of Matthews coefficient for each (γ, C) pair from all five runs was then averaged and the pair with the highest average performance score was chosen as the parameter setting for the classifier on the full training data, and the learned model was then tested on the separate test set. Figure 4.3 illustrates the five-fold cross validation process.

4.7 Results

We used the number of problems for which a selected variable ordering is optimal to measure the efficacy of each heuristic and compared the efficacy of the heuristic selected by machine learning with the efficacy of each individual heuristic.

Table 4.2 breaks down the results into a set of mutually exclusive outcomes that describe all possibilities. The column headed 'Machine Learning' indicates whether the machine learning selected heuristic was the best one. The next three columns indicate which of the heuristics performed best. All 13 possibilities are listed above. Note that at

Figure 4.3: 5-fold cross validation



least one of the fixed heuristics must have a 'Y' since, by definition, the optimal ordering is obtained by at least one heuristic. For each of these cases we list the number of problems for which this case occurred for both the quantifier-free and quantified experiments.

For many problems more than one heuristic selects the optimal variable ordering and the probability of a randomly selected heuristic giving the optimal ordering depends on how many optimal heuristics we have. For example, a random selection would be successful 1/3 of the time if one heuristic gives the optimal ordering or 2/3 of the time if two heuristics do so.

In Table 4.2, case 1 is where machine learning cannot make any difference as all heuristics are equally optimal. We compare the remaining cases pairwise. For each pair, the behaviour of the fixed heuristics are identical and the difference is whether or not machine learning picked a winning heuristic (one of the ones with a Y). We can see that in all cases but one the machine learning algorithm selects an optimal heuristic more often than not (in Cases 8 and 9 machine learning selects optimally for 50% of Quantifier-Free examples and 47% of quantified examples). For each pair we can compare its selection with a random selection. For example, for the pair of cases 2 and 3, `sotd` and `ndrr` are successful heuristics and `Brown` is not. A random selection would be successful 2/3 of the time. For the quantifier-free examples, the machine learned selection is successful $146/(146+39)$ or approximately 79% of the time.

We repeated this calculation for the quantified case and the other pairs, as shown in Table 4.3. In each case the values have been compared to the chance of success when picking a random heuristic. There are two distinct sets in Table 4.3: those where only one heuristic was optimal and those where two are. We see that the machine learning selection is better than random choice in every case in both experiments.

Table 4.2: Categorising the problems into a set of mutually exclusive cases characterised by which heuristics were successful.

Case	Machine Learning	`sotd`	`ndrr`	`Brown`	quantifier-free	Quantified
1	Y	Y	Y	Y	399	573
2	Y	Y	Y	N	146	96
3	N	Y	Y	N	39	24
4	Y	Y	N	Y	208	232
5	N	Y	N	Y	35	43
6	Y	N	Y	Y	64	57
7	N	N	Y	Y	7	11
8	Y	Y	N	N	106	66
9	N	Y	N	N	106	75
10	Y	N	Y	N	159	101
11	N	N	Y	N	58	89
12	Y	N	N	Y	230	208
13	N	N	N	Y	164	146

Table 4.3: Proportion of examples where machine learning picks a successful heuristic.

`sotd`	`ndrr`	`Brown`	Quantifier-free	Quantified
Y	Y	N	79% (>67%)	80% (>67%)
Y	N	Y	86% (>67%)	84% (>67%)
N	Y	Y	90% (>67%)	84% (>67%)
Y	N	N	50% (>33%)	47% (>33%)
N	Y	N	73% (>33%)	53% (>33%)
N	N	Y	58% (>33%)	59% (>33%)

By summing those cases in Table 4.2 where machine learning selects the optimal heuristic, as well as the cases where each individual heuristic performs best, we get Table 4.4. This compares, for both the quantifier-free and quantified problem sets, the learned selection with each of the heuristics on their own.

Of the three heuristics, `Brown` seems to be the best, albeit by a small margin. Its performance is a little surprising, both because the `Brown` heuristic is less well known (having never been formally published) and because it requires little computation (taking only simple measurements on the input).

Table 4.4: Total number of problems for which each heuristic picks the best ordering.

	Machine Learning	`sotd`	`ndrr`	`Brown`
quantifier-free	1312	1039	872	1107
Quantified	1333	1109	951	1270

For the quantifier-free problems there were 399 problems where all three heuristics picked the optimal variable ordering, 499 where two did and 823 where one did. Hence for the problem set the chances of picking a successful heuristic at random is

$$\frac{100}{1721}\left(399 + 499 \times \tfrac{2}{3} + 823 \times \tfrac{1}{3}\right) \simeq 58\%$$

which compares to $100 \times 1312/1721 \simeq 76\%$ for machine learning. For the quantified problems the figures are 64% and 77%. Hence, machine learning performs much better than a random choice in both cases. Further, if we were to use only the heuristic that performed the best on this data, the `Brown` heuristic, then we would pick a successful ordering for approximately 64% of the quantifier-free problems and 74% of the quantified problems. We see that a machine learned choice is also superior to simply using any one heuristic.

4.8 A comparison of the three heuristics

In the process of applying machine learning to pick a heuristic for selecting a variable ordering, a large amount of data was generated. For each of the 7001 three-variable examples in the `nlsat` database a CAD was constructed for all six variable orderings, and all three heuristics were used to predict a variable ordering. Previous work on CAD heuristics has involved small data sets [36] (used 48 examples to obtain their conclusions), so such a large data set may lead to fresh insight. From the experimental results, the conclusion of which heuristic performed best varies depending on the criteria used. In this section, I give a comparison of the three heuristics and highlight their performance under different criteria.

Table 4.5 shows the number of problems and their relative occurrence in the problem set where each heuristic was the most competitive of the three. From this table, we see that `Brown` heuristic is most likely to make the best choice, both when quantified and when quantifier-free.

Whilst selecting an optimal variable ordering is important, it is also relevant to consider the actual savings in cell counts. When a heuristic selects a non-optimal variable ordering it may differ from the optimal choice by as little as 2 cells or as many as thousands of cells. Table 4.6 summarises this behaviour by computing the average cell count for each problem over the six variable orderings, then computing the percentage saved (with a negative percentage indicating an increase in cell count) for the variable ordering each heuristic selects. The mean and median of the savings of each heuristic are given in Table 4.6.

Next, we investigated how much of a cell count saving is offered by each heuristic, and made the following calculations for each problem:

(1) The average cell count of the six orderings;

(2) The difference between the cell count for each heuristic's pick and the problem average;

(3) The value of (2) as a percentage of (1).

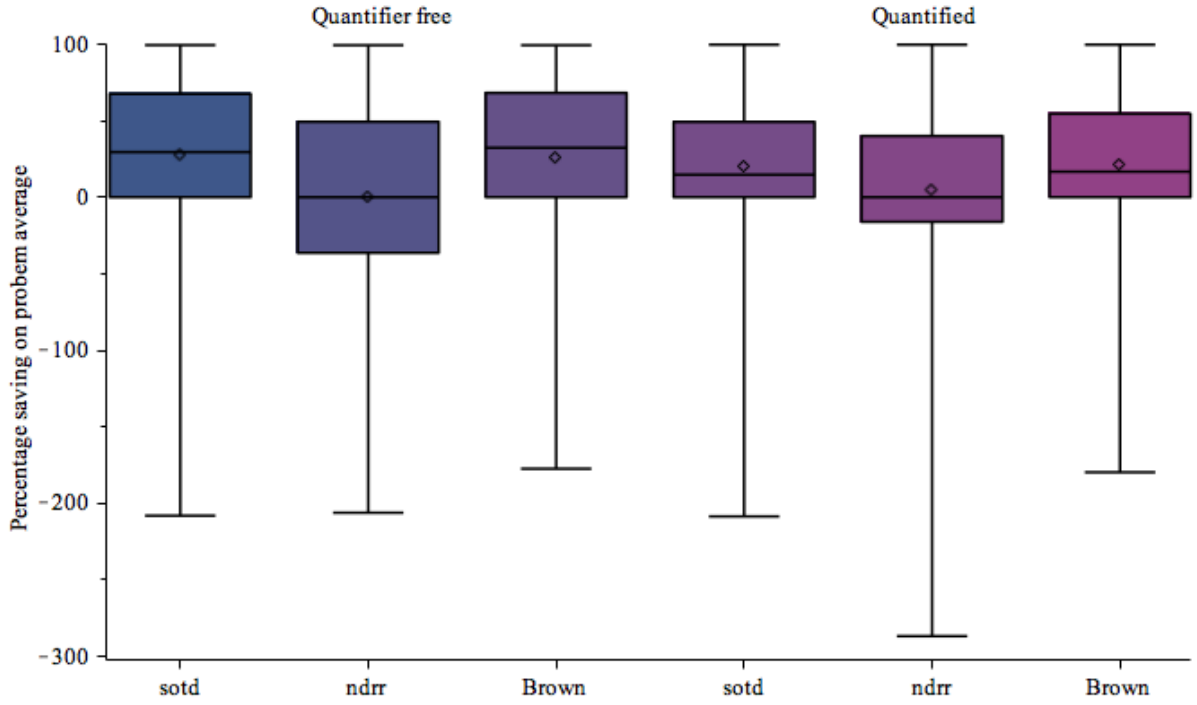
These calculations were made for all problems in which no variable ordering timed out (5262 of the quantifier-free problems and 5332 of the quantified problems). The data is

Table 4.5: Number of problems and their relative occurrence in the problem set where each heuristic was the most competitive.

	`sotd`	`ndrr`	`Brown`
quantifier-free	4221 (60.29%)	3620 (51.71%)	4523 (64.61%)
Quantified	4603 (65.75%)	4000 (57.13%)	5166 (73.79%)

shown in form of boxplots in Figure 4.4. The boxes indicate the second and third quartiles, separated by the median. The vertical range indicates the range of the data set, and the mean is shown as a circle. The mean and median values are also given in Table 4.6 (and marked in Figure 4.4 with circles and lines respectively). Outliers are discounted, which are points further than $\frac{3}{2}$ times the interquartile range away from the upper and lower quartiles.

Figure 4.4: Percentage of cell count saving offered by each heuristic (mean and median). The range of the data is indicated (discounting outlier values).



While `Brown` heuristic makes the best choice most frequently, for quantifier-free problems the average saving of using `sotd` is actually larger. However, for quantified problems `Brown` heuristic delivered more savings. `Ndrr` performs the worst on average, but there are classes of problems where it makes a better choice than the others. For example, consider the problems where at least one ordering timed out. Table 4.7 describes how often each heuristic avoids a time out. We see that for quantified problems `ndrr` does the best.

Table 4.6: How much of a cell count saving is offered by each heuristic (mean and median).

	Mean average			Median value		
	`sotd`	`ndrr`	`Brown`	`sotd`	`ndrr`	`Brown`
quantifier-free	27.32%	-0.20%	25.27%	29.47%	0.00%	32.28%
Quantified	19.47%	4.15%	21.03%	14.68%	0.00%	16.67%

Table 4.7: How many times each heuristic avoids a time out.

	`sotd`	`ndrr`	`Brown`
quantifier-free	559	537	594
Quantified	512	530	478

4.9 Summary

The experimental results confirmed our hypothesis, that no one heuristic is superior for all problems and the correct choice will depend on the problem. Each of the three heuristics tested had a substantial set of problems for which they were superior to the others and so the problem was a suitable application for machine learning.

Using machine learning to select the best CAD heuristic yielded better results than choosing one heuristic at random, or just using any of the individual heuristics in isolation, indicating there is a relation between simple algebraic features of the problem and the best heuristic choice. This could lead to the development of a new individual heuristic in the future.

The experiments involved testing heuristics on 1721 CAD problems, certainly the largest such experiment that I am aware of. For comparison, the best known previous study on such heuristics [36] tested with 48 problems. From the experimental results, the conclusion of which heuristic is the best varies depending on the criteria. If machine learning is not available then `Brown` heuristic is the most competitive for our example set, and this is despite it involving less computation than the others.

Chapter 5

Predicting the usefulness of Gröbner basis preconditioning

Similar to how CAD computation is useful for solving systems of polynomial (in)equations and inequalities, Gröbner basis calculation is one of the main practical tools for solving systems of polynomial equations. Furthermore, a study by Wilson [106] shows an interesting connection between both algorithms. In particular, applying a Gröbner basis calculation to systems of polynomial (in)equalities to precondition the input problem before invoking CAD may often reduce the number of cells generated during the CAD construction. However, Gröbner basis preconditioning is not always beneficial. As there is no fixed rule to decide if Gröbner basis preconditioning is beneficial or not, we investigate whether machine learning can help with the prediction.

In the rest of this chapter, I present the methodology of the experimental work undertaken. I applied machine learning to a single classification problem, predicting whether Gröbner basis preconditioning is beneficial or not for a given set of problems. In addition, a series of feature selection experiments was carried out to determine which measured features are significant. The learning results were analysed and compared using different feature subsets. Finally I will give a summary and ideas for future work.

5.1 Gröbner basis preconditioning for CAD

Our goal is to apply machine learning to the problem of predicting whether Gröbner basis preconditioning is beneficial to CAD problems or not. More specifically, we want to examine this in the context of conjunctions of polynomial (in)equalities. Suppose we want to decide the validity of the formula

$$e_1 = 0 \wedge \cdots \wedge e_k = 0 \wedge B(f_1, \ldots, f_l),$$

where B is a boolean combination of inequations ($\neq$) and inequalities ($>, <$) on some polynomials f_j where $j \in 1, \cdots, l$, and $e_1 = 0, \cdots, e_k = 0$ is a set of polynomial equations. Applying a Gröbner basis preconditioning to the input does the following: rather than computing a CAD for the system of equations E (i.e. $e_1 = 0 \wedge e_2 = 0 \wedge \cdots$) and inequalities B directly, we first compute a (purely lexicographical) Gröbner basis on E, and output a set of polynomials GB which is equivalent to E in the sense that it generates the same ideal. The CAD step is now performed on the Gröbner basis GB (instead of E), together with B.

Note that Gröbner basis preconditioning is computationally very cheap compared to CAD construction. However, it may be the case that the CAD calculation on the preconditioned input is less efficient than on the original input; hence it is not always beneficial to perform the CAD on the preconditioned input. The question therefore is less whether we want to *compute* the Gröbner basis of the input, but rather whether we want to *use* it. This distinction allows us to always compute the Gröbner basis of the input and use some of its algebraic properties as features for our learning algorithm.

5.2 Data

I conducted an initial experiment using the same problem set (the `nlsat` dataset [79]) as in Chapter 4. For Gröbner basis preconditioning to be useful, a problem needs to contain a conjunction of at least two equalities. The Gröbner basis of a single polynomial is that polynomial itself, so preconditioning is possible, but not useful in this case. From the data set, 493 three-variable problems and 403 four-variable problems were extracted that meet this criterion. Gröbner basis preconditioning was applied to each problem. Cell counts with and without using Gröbner basis preconditioning were collected separately using Maple. It turns out that Gröbner basis preconditioning is always beneficial for the extracted three-variable and four-variable problems. This is an interesting finding for the `nlsat` dataset, however it means that the `nlsat` dataset cannot be used for a meaningful machine learning experiment.

I also considered other traditional CAD problem sets (such as Wilson et al. [107]), however none of them have a sufficient number of problems for machine learning as far as we know. Hence, we decided to generate problems for the experiment. The problem generation process was designed to generate unbiased data sets for learning, while being computationally feasible for the experiment.

In total, 1200 problems were generated in Maple using the *randpoly* (random polynomial generator) command. The code in Figure 5.1 was used to generate the problems for the experiment. For each problem, two sets of polynomials (E and B) were generated with each of them containing three polynomials. The set E represents the set of conjoined polynomial equations, while B represents the set of polynomial inequations and inequalities. The set of polynomials was generated by fixing the number of variables to three and the number of terms to two for each polynomial. Total degrees varied between 2 and 4 (as illustrated in Figure 5.1, 400 problems are generated for each variation) and the coefficients of the polynomials vary between -20 and 20.

5.3 Evaluating the heuristics

Originally, 1200 three-variable problems were generated as previously described. For each problem, the CAD cell counts both with and without applying Gröbner basis preconditioning were recorded and compared. A time limit of 300 CPU seconds was set for each problem, which resulted in 1062 problems that finished the run within this time limit. These 1062 problems constitute the dataset. The usefulness of Gröbner basis preconditioning was determined by whether it reduced the cell count or not, compared with a direct CAD.

Technical Note: all computations were performed in Maple 17 on a 2.4GHz Intel

Figure 5.1: Sample MAPLE command for random polynomial generation. The value of `num` varies between 2 and 4.

```
E := [randpoly([x,y,z], terms = 2,
    degree = num, coeffs = rand(-20 .. 20)),
    randpoly([x,y,z], terms = 2,
    degree = num, coeffs = rand(-20 .. 20)),
    randpoly([x,y,z], terms = 2,
    degree = num, coeffs = rand(-20 .. 20))
    ];
B := [randpoly([x,y,z], terms = 2,
    degree = num,coeffs = rand(-20 .. 20)),
    randpoly([x,y,z], terms = 2,
    degree = num,coeffs = rand(-20 .. 20)),
    randpoly([x,y,z], terms = 2,
    degree = num,coeffs = rand(-20 .. 20))
    ];
```

processor. The CAD method used is the one described in Chen et al. [24], where first $\mathbb{C}^n$ is decomposed cylindrically in the variable ordering, and then the decomposition is refined to a CAD of $\mathbb{R}^n$. The two phases are analogous but not equivalent to projection and lifting (see [23] for details). A purely lexicographical order with $x \prec y \prec z$ was used as a monomial order for the Gröbner base (denoted plex(z, y, x)).

5.4 Problem features

In order to apply machine learning, we need to identify relevant features of the problems. Table 5.1 shows the 28 features that were identified, where (x, y, z) are the three variable labels used in all our problems. They were chosen as easily computable features of the problems that relate to the cell count of its CAD. The features used for the current experiment mainly fall into two sets. The first set of features were generated from polynomials of the original problem, the other set of features were obtained from polynomials after applying Gröbner basis preconditioning. In addition, Wilson [106] proposed the following metric:

$$\texttt{TNoI}(F) = \sum_{f \in F} \texttt{NoI}(f),$$

where $\texttt{NoI}(f)$ is the number of indeterminates present in a polynomial f, and F denotes the set of polynomials in the problem. The measure `TNoI` itself showed a promising correlation to whether the preconditioning is beneficial or not. Moreover, the logarithm (base 2) of the ratio of `TNoI` (equivalently the difference of the logarithms) has an even stronger correlation to changes in the cell counts. Hence the difference of the logarithms of `TNoI` was also included in the feature set. In addition, we also consider the logarithm of the ratio of the maximum total degrees (`tds`) and the sum of total degrees (`stds`). The definition of the `tds` and `stds` measures is

$$\texttt{tds}(F) = \max_{f \in F} \texttt{tds}(f),$$

$$\texttt{stds}(F) = \sum_{f \in F} \texttt{tds}(f).$$

Note it should be clear that `stds` measure here differs from the `sotd` heuristic described in Section 4.2. The `stds` measure here only calculates the sum of the total degrees of the input polynomials, and no projection is involved. However, the `sotd` heuristic constructs the full set of projection polynomials for each permitted ordering and selects the ordering whose corresponding set has the lowest sum of total degrees for each of the monomials in each of the polynomials. The `stds` measure is computationally much cheaper than the `sotd` heuristic.

In addition to training a classifier using all the features provided in Table 5.1, I trained classifiers using two subsets of the all features in order to understand whether one set appeared more useful than the other. One feature subset contains 12 features about the set of polynomials from the original problem before applying Gröbner basis preconditioning (Feature Numbers 1 to 12 from Table 5.1), while the other subset contains 13 features about the set of polynomials after applying Gröbner basis preconditioning (Feature Numbers 13 to 25 from Table 5.1). The number of polynomials for the input problems was always six, so this is not included in the first feature set. However, the number of polynomials after Gröbner basis preconditioning varies and hence is included in the second feature set. For convenience, in the rest of the chapter, I use *all features* to denote all 28 features in Table 5.1, *before features* for features obtained from the input polynomials before applying Gröbner basis preconditioning, and *after features* for features obtained from the polynomials after applying Gröbner basis preconditioning.

I applied the feature generation process to create training sets with three feature sets (*all features*, *before features*, *after features*) separately. In spite of the use of different feature sets, each feature vector in the training set was associated with a label (the corresponding tuples in the three feature subsets have the same label). A training set was derived, with each problem labelled $+1$ if Gröbner basis preconditioning is beneficial for CAD construction, or -1 otherwise. The features were calculated from the problem input using Maple. For example, suppose that the input formula is defined using the set of polynomials

$$E := \{-12yz - 3z, \quad 17x^2 - 6, \quad -2yz + 5x\}$$
$$B := \{-2yz - 9y, \quad -15x^2 - 19y, \quad 6xz + 3\}$$

Computing a Gröbner basis for E, we obtain a new set of polynomials GB

$$GB := \{17x^2 - 6, \quad 4y + 1, \quad z + 10x\}.$$

Then the problem will have the *all features* vector

$$\left[12, 12, 2, 2, 1, 1, \frac{2}{3}, \frac{2}{3}, \frac{2}{3}, \frac{1}{3}, \frac{5}{12}, \frac{5}{12}, 6, 10, 10, 2, 2, 1, 1, \frac{2}{3}, \frac{1}{2}, \frac{1}{2}, \frac{1}{3}, \frac{1}{3}, \frac{1}{4}, 0.263, 2.585, 0\right],$$

where each component corresponds to a feature in Table 5.1; the *before features* vector will be

$$\left[12, 12, 2, 2, 1, 1, \frac{2}{3}, \frac{2}{3}, \frac{2}{3}, \frac{1}{3}, \frac{5}{12}, \frac{5}{12}\right],$$

Table 5.1: Description of the features used

Feature number	Description
1	`TNoI` before GB.
2	`stds` before GB.
3	`tds` of polynomials before GB.
4	Maximum degree of x among all polynomials before GB.
5	Maximum degree of y among all polynomials before GB.
6	Maximum degree of z among all polynomials before GB.
7	Proportion of x occurring in polynomials before GB.
8	Proportion of y occurring in polynomials before GB.
9	Proportion of z occurring in polynomials before GB.
10	Proportion of x occurring in monomials before GB.
11	Proportion of y occurring in monomials before GB.
12	Proportion of z occurring in monomials before GB.
13	Number of polynomials after GB.
14	`TNoI` after GB.
15	`stds` after GB.
16	`tds` of polynomials after GB.
17	Maximum degree of x among all polynomials after GB.
18	Maximum degree of y among all polynomials after GB.
19	Maximum degree of z among all polynomials after GB.
20	Proportion of x occurring in polynomials after GB.
21	Proportion of y occurring in polynomials after GB.
22	Proportion of z occurring in polynomials after GB.
23	Proportion of x occurring in monomials after GB.
24	Proportion of y occurring in monomials after GB.
25	Proportion of z occurring in monomials after GB.
26	lg(`TNoI` before) - lg(`TNoI` after)
27	lg(`stds` before) - lg(`stds` after)
28	lg(`tds` before) - lg(`tds` after)

with components corresponding to Feature Numbers 1 to 12 in Table 5.1; and the *after features* vector will be

$$\left[6, 10, 10, 2, 2, 1, 1, \frac{2}{3}, \frac{1}{2}, \frac{1}{2}, \frac{1}{3}, \frac{1}{3}, \frac{1}{4}\right],$$

with component corresponding to Feature Numbers 13 to 25 in Table 5.1.

After the feature generation process, the training data (feature vectors) were standardised so that each feature had zero mean and unit variance across the training set. The same standardisation was then applied to features in the test set, using the mean and standard derivation from the training set.

5.5 Cross-validation and grid-search

I conducted the experiment on the 1062 conjoined polynomial (in)equalities. The data was partitioned into 80% training (849 problems) and 20% test (213 problems), stratified

to maintain relative class proportions in both training and test partitions. We recall that there are two parameters involved when using a RBF kernel: a penalty parameter C and γ for RBF kernel itself. We used a grid-search optimisation procedure along with a five-fold stratified cross validation (see Section 4.6) to find optimal parameter values for C and γ.

I repeated this procedure for the *all features* set, the *before features* set and the *after features* set.

5.6 Results for three feature sets

I compared the machine learning outcomes, between results obtained with *all features* used, with just the *before features*, and with just the *after features*. The classification accuracy was used to measure the efficacy of the machine learning selection process. The test set contained 159 positive samples and 54 negative samples (there are 159 problems where applying Gröbner basis preconditioning is beneficial). In approximately 75% of the cases Gröbner basis preconditioning was beneficial for CAD construction for the given problem; this was used as a baseline for measuring the efficacy of the classifiers. The selection results are given in Table 5.2. We can see that only using *before features* has no improvement compared to the baseline case; while both *all features* and *after features*, demonstrated better results than baseline. The selection result of using *all features* is inferior to only using *after features*. This seems to indicate that the features that matter mainly concern the algebraic properties of the polynomials after preconditioning. However, we cannot conclude this directly: early research has shown that a variable that is completely useless by itself can provide a significant performance improvement when taken with others [49]. To be absolutely confident about which features were significant and which were superfluous, further feature selection experiments were carried out. The results show that the optimal feature subsets contains features from both *before features* and *after features*. We will see more details in the following sections.

Table 5.2: Predicted accuracy

Heuristics	Number of problems	Percentage of test set
Always applying GB	159	75%
All features	162	76%
Before features	159	75%
After features	167	78%

5.7 Feature selection

Given the fact that *after features* is a subset of *all features*, there is an indication that not all the features contribute to the machine learning process, only a small number of features are needed for learning to be effective. Moreover, the reduced feature set is often beneficial for better understanding the underlying concept. Consequently some *feature selection* methods were applied. I have already discussed various feature selection

techniques in the machine learning background chapter (see Section 2.2.3). In this experiment, both filter and wrapper methods were applied for feature selection, and the results were compared. The feature selection experiments were conducted in WEKA (Waikato Environment for Knowledge Analysis) [51], which is a popular machine learning library written in Java. WEKA supports several standard machine learning tasks, for example data preprocessing, clustering, classification, regression and feature selection. Each data point is also represented as a fixed number of features. The inputs for this experiment are samples of 29 features, where the first 28 are the real-valued features from Table 5.1, and the final one is a "nominal" feature denoting its class. I will describe two feature selection methods (the filter method and the wrapper method) in the following two sections.

5.7.1 The filter method

A popular filter method, the correlation based feature selection method, was applied (see Hall [52]). Unlike other popular filter methods (see [50]), correlation based feature selection measures the rank of feature subsets instead of individual features. The feature subset which contains features highly correlated with the class, but uncorrelated with each other is preferred. The heuristic below is used to measure the quality of a feature subset, and takes into account feature-class correlation as well as feature-feature correlation.

$$G_s = \frac{k\overline{r_{ci}}}{\sqrt{k + k(k-1)\overline{r_{ii'}}}} \tag{5.1}$$

Above, k is the number of features in the subset, $\overline{r_{ci}}$ denotes the average feature-class correlation of feature i, and $\overline{r_{ii'}}$ is the average feature-feature correlation between feature i and i'. Here, the numerator of Equation 5.1 indicates how much relevance there is between the class and a set of features, while the denominator measures the redundancy among the features. The higher this measure, the better the feature subset.

In order to apply this heuristic to estimate the merit of a feature subset, it is necessary to compute the feature-class correlations and feature-feature correlations. With the exception of the class attribute, all 28 features are continuous. In order to have a common measure for computing the correlations in Equation 5.1, we first discretize the numeric features using the method of Fayyad and Irani [41]. After that, a correlation measure based on the information-theoretical concept of *entropy* is used, which is a measure of the uncertainty of a random variable. The entropy of a variable X is defined as

$$H(X) = -\sum_i p(x_i) log_2(p(x_i)). \tag{5.2}$$

The entropy of X after observing values of another variable Y is defined as

$$H(X|Y) = -\sum_j p(y_j) \sum_i p(x_i|y_j) log_2(p(x_i|y_j)). \tag{5.3}$$

where $p(x_i)$ is the prior probabilities for all values of X, and $p(x_i|y_i)$ is the posterior probabilities of X given the values of Y. The *information gain (IG)* [88] measures the amount by which the entropy of X decreases by additional information about X provided by Y, and it is given by

$$IG(X, Y) = H(X) - H(X|Y). \tag{5.4}$$

The *symmetrical uncertainty (SU)* (a modified information gain measure) is then used to measure the correlation between two discrete variables (X and Y) [87]:

$$SU(X,Y) = 2.0 \times \left[\frac{H(X) - H(X|Y)}{H(X) + H(Y)}\right] \tag{5.5}$$

Treating each feature as well as the class as random variables, we can apply this as our correlation measure. More specifically, we simply use $SU(c,i)$ to measure the correlation between a feature i and a class c, and $SU(i,i^{'})$ to measure the correlation between features i and i'. These values are then substituted as $\overline{r_{ci}}$ and $\overline{r_{ii'}}$ in Equation 5.1.

We recall that the aim of this correlation based filter method is to find the optimal subset of features which maximises the metric given in Equation 5.1. Although the size of the feature set in this experiment was small, consisting of only 28 features, the number of possible subsets is still very large. There are

$$2^{28} - 1 \simeq 2.7 \times 10^8$$

subsets, which is too many for exhaustive search. Instead a greedy stepwise forward selection search strategy was used for searching the space of feature subsets, which works by adding the current best feature at each round. The search begins with the empty set, and in each step the score (as in Equation 5.1) is computed for every single feature addition, and the feature with the best score improvement is added. If at some step none of the remaining features provide an improvement, the algorithm stops, and the current feature set is returned. The best feature subset found with this method is shown in Table 5.3.

Table 5.3: Feature selection results suggested by the filter method, ordered by importance

Feature number	Description
14	`TNoI` after GB.
13	Number of polynomials after GB.
2	`stds` before GB.
26	lg(`TNoI` before) - lg(`TNoI` after)
21	Proportion of y occurring in polynomials after GB.
15	`stds` after GB.
23	Proportion of x occurring in monomials after GB.
19	Maximum degree of z among all polynomials after GB.
25	Proportion of z occurring in monomials after GB.
27	lg(`stds` before) - lg(`stds` after)

5.7.2 The wrapper method

The wrapper feature selection method evaluates attributes by using accuracy estimates provided by the actual target learning algorithm. Evaluation of each feature set was

conducted with a learning scheme (as with earlier experiments, a support vector machine with radial basis function kernel function was used). The SVM algorithm is run on the dataset, with the same data partitions as described in Section 5.5. Similarly, a five-fold cross validation was carried out. The feature subset with the highest average accuracy was chosen as the final set on which to run the SVM algorithm.

In more detail, in each training/validation fold, starting with an empty set of features, each feature was added and a model was fitted to the training data set and the classifier was then tested on the validation set. This was done on all the features, resulting in a score for each where the score reflects the accuracy of the classifier. The final score for each feature was its average over the five folds. Having obtained a score for all features in the manner above, the feature with the highest score was then added in the feature set. Then, the same greedy procedure as described for the filter method (see Section 5.7.1), was applied to obtain the best feature subset.

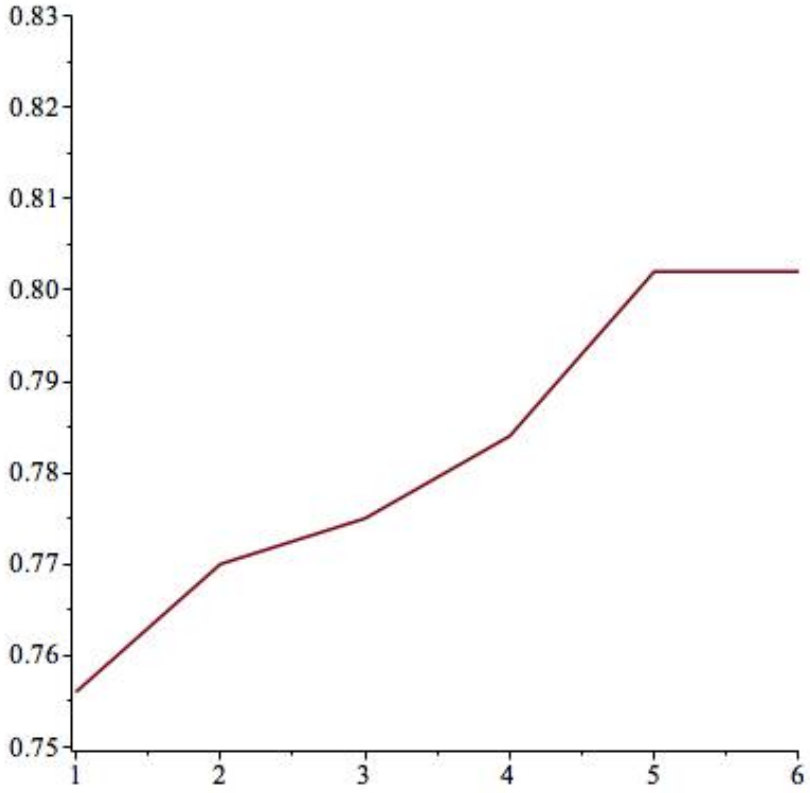
Considering the large number of cases, the parameters (C, γ) were selected from an optimised sub range instead of performing the full grid search range as in Section 5.5. Ideally one would optimise over a large parameter selection range, but a reduced range suffices to demonstrate the performance of a reduced feature set. In the previous experiments with *all features*, *before features* and *after features*, I found that C taken from $2^5, 2^6, 2^7, 2^8, 2^9, 2^{10}$ and γ taken from $2^{-5}, 2^{-6}, 2^{-7}, 2^{-8}, 2^{-9}, 2^{-10}$ generally provided good classifier performance (the best parameter settings from *all features*, *before features* and *after features* all fell into this range). The 36 pairs of (C, γ) values were tested and an optimal feature subset with the highest accuracy was found for each. The feature subset that gave the highest accuracy was then selected as the final feature set. The best feature subset found is shown in Table 5.4, where we can see that most of the features related to variable z (feature numbers 9, 12 and 13). This makes sense since the projection order we used was $x \prec y \prec z$. The variable z is projected first and hence affects the complexity the most.

Table 5.4: Feature selection results suggested by the wrapper method, ordered by importance

Feature number	Description
14	`TNoI` after GB.
9	Proportion of z occurring in polynomials before GB.
22	Proportion of z occurring in polynomials after GB.
4	Maximum degree of x among all polynomials before GB.
12	Proportion of z occurring in monomials before GB.

Furthermore I examined the performance on even further reduced feature sets, obtained by the feature ranking given by the wrapper method. The overall prediction accuracies are shown in Figure 5.2. For instance, the predictor obtained from only using a single feature (the best ranked feature was TNoI after GB for both filter and wrapper methods) scored an accuracy score of 0.756 in that run, with the performance steadily increasing with the size of the feature set until the fifth feature. Taking any sixth feature into the set did not improve the performance noticeably, and hence resulted in the cutoff

Figure 5.2: Performance of a sample run with different sizes of feature sets



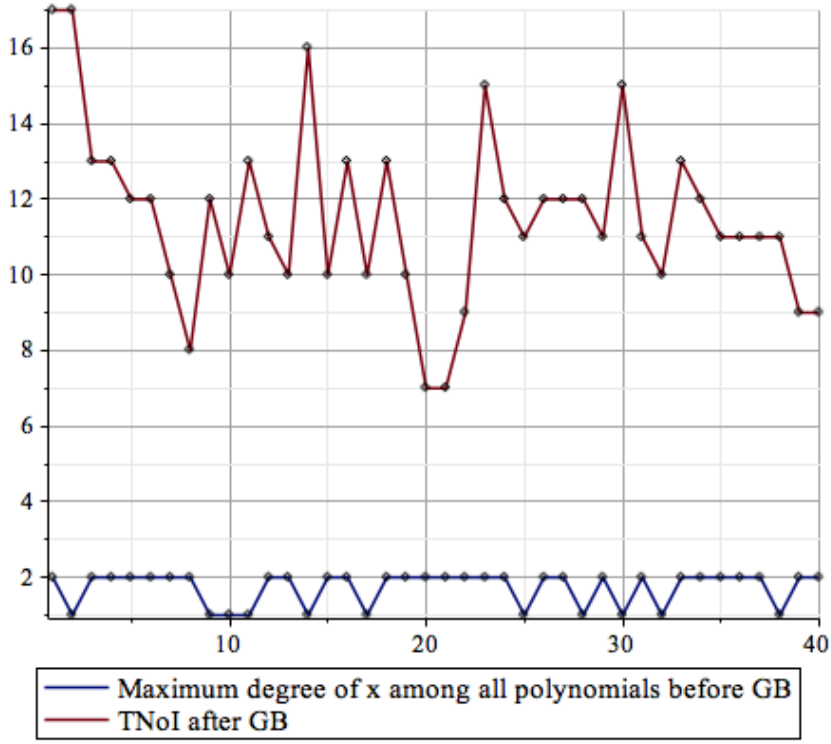
chosen by the wrapper method.

In addition, given the now relatively low dimensionality of the feature space as obtained by the wrapper method, I performed an error analysis on the misclassified data points. Figure 5.3 shows 40 misclassified data points and their features 4 and 14, while Figure 5.4 shows the remaining features 9, 12, 22 of the same samples. While most of the features do not seem to exhibit any prominent pattern, it is interesting that feature 4 of all misclassified samples is either 1 or 2. Compared to the whole data set, which consisted to roughly a third of samples with a feature 4 value of 3 or 4, this seems to indicate that the algorithm performs better on instances with a higher maximum degree of x among all polynomials before GB.

5.7.3 Results with reduced features

Having obtained now the reduced feature sets, I applied the same experiment again to compare the performance of the learning when using the smaller feature sets. The data set was partitioned into 80% training and 20% test set, and were stratified to maintain relative class proportions in both training and test partitions. Again, a five-fold cross validation and a finer grid-search optimisation procedure over the range of (C, γ) pairs was conducted where γ varies between $2^{-15}, 2^{-14}, 2^{-13}, \ldots, 2^{3}$ and C varies between $2^{-5}, 2^{-4}, 2^{-3}, \ldots, 2^{15}$ (as described in Section 4.6) to determine the parameter selection, and the selected features were used. The classifier with maximum averaged Matthew coefficient was selected and the resulting classifier was then evaluated on the test. The testing

Figure 5.3: Feature 4 and 14 of misclassified data

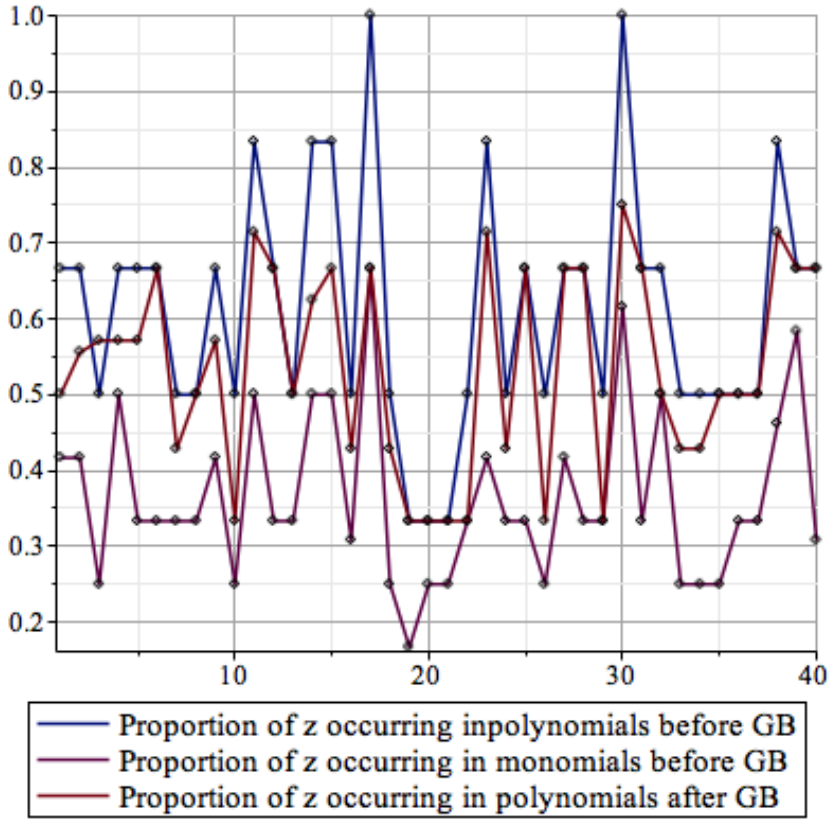


data was also reduced to contain only the features selected. The classification accuracy was used to measure the performance of the classifier. In order to better estimate the generalisation performance of classifiers with reduced feature sets, the data was permuted and partitioned into 80% training and 20% test again and the whole process was repeated for 50 times. For each run, each training set was standardised to have zero mean and unit variance, with the same offset and scaling applied subsequently to the corresponding test partition. Figure 5.5 shows boxplots of the list of accuracy generated by 50 runs of the five-fold cross validation. Both reduced feature sets generated similar results and show a large improvement on the base case where Gröbner basis preconditioning is always used before CAD construction. The average overall prediction accuracy of filter subset and wrapper subset is 79% and 78% respectively; all the 50 runs of wrapper subset performed above base line, while the top three quantiles of the results of both sets achieve higher than 77% percentage accuracy.

5.8 Summary

Distribution and Correlation of Features We investigated the application of machine learning to the problem of predicting whether Gröbner basis preconditioning is beneficial to CAD problems or not. The experiments involved testing on 1062 randomly generated problems. I conducted the initial experiment on three feature sets (*all features*, *before features* and *after features*) to predict whether Gröbner basis preconditioning is beneficial or not for CAD construction. Three feature sets were tested in order to understand whether one set appeared more useful than the others. Using the machine learned selec-

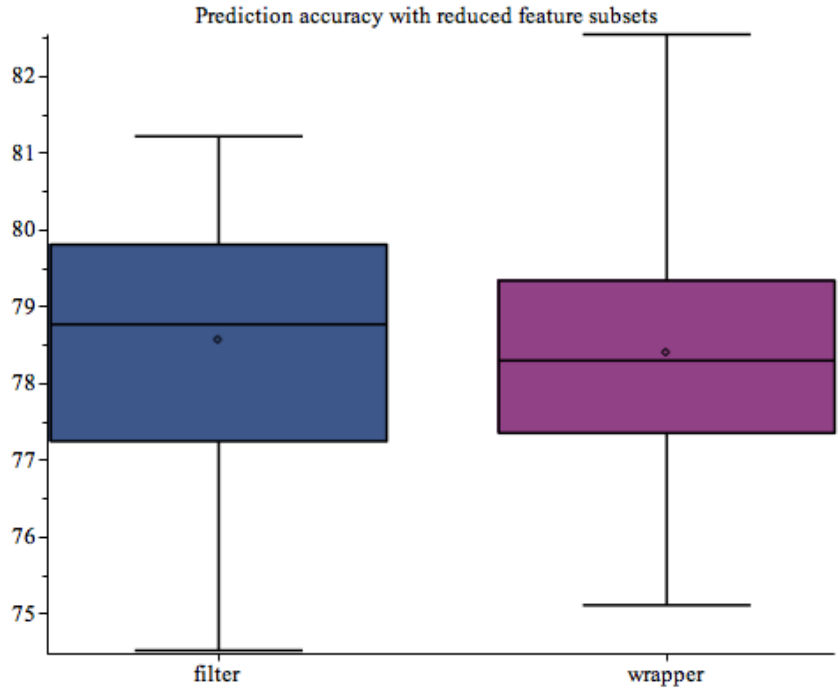
Figure 5.4: Feature 9,12 and 22 of misclassified data



tion (with *all features* and *after features*) yielded better results than always using Gröbner basis preconditioning by default. The results from using only *before features* ties with the baseline case. I also did a series of feature selection experiments; the results showed that the optimal feature subsets contains features from both *before features* and *after features*. The results also demonstrated that not all features contribute to the machine learning process; having fewer features may even improve learning efficiency. Though a large number of features were examined, it turned out that only less than half of all features are required for effective learning in this experiment. Different feature selection methods gave rise to different feature subsets, while both of them showing an improvement over the baseline case. The prediction of which features were needed could not be reasonably made beforehand, and so the feature selection method is important and it may indicate something useful to be learned about computer algebra algorithms.

There are many ways in which this work could be extended to further explore the benefit machine learning can offer in the formulation of problems for CAD. My experiments involved testing on randomly generated problems due to the limited availability of data, and thus the data was quite uniform. An obvious extension to the work in this section is to experiment on a wider data set to see if these results, both the benefit of machine learning and the superiority of Gröbner basis preconditioning, hold more generally. Furthermore in the current experiment, the variable ordering for CAD and Gröbner basis preconditioning was fixed. Ideally, we could also investigate the efficiency of machine learning on both, Gröbner basis preconditioning and the variable ordering simultaneously. These problems

Figure 5.5: Boxplots of 50 runs of the 5-fold cross validation with both the feature sets suggested by filter and wrapper methods



are interrelated: the choice of variable ordering could affect the choice of Gröbner basis preconditioning or not and likewise, performing Gröbner basis preconditioning could affect what the best variable ordering is. Hence, the best variable ordering without Gröbner basis preconditioning can be different to the best one after preconditioning, and the best overall may or may not involve preconditioning. A key extension for future work would be to use machine learning to predict both variable ordering and the usefulness of Gröbner preconditioning together, and investigate if a better performance can be obtained.

Chapter 6

Related work

The term *algorithm selection* was first introduced by Rice [91] to formalize the problem: *Which algorithm is likely to perform best for my problem?* Computer algebra systems often have the issue of algorithm selection since there is rarely one single best algorithm for the entire problem space. Computer algebra systems often use *meta-algorithms* to make these choices, and their decisions are based on numerical parameters [21]. For example, MATHEMATICA uses a meta-algorithm for solving systems over cylindrical cells described by cylindrical algebraic formulae. This allows cells produced from a CAD to be used easily in further computation [97].

In this dissertation, another approach was taken to address the algorithm selection problem, instead of the traditional meta-algorithms approach. Namely, we employed machine learning for the task of selecting the best decision or heuristic in three instances of computer algebra applications over RCFs. Although the approach taken here for the particular case of computer algebra is novel, there is existing work in similar fields. The related work presented in the rest of the section covers these closely related topics, namely machine learning in theorem proving, in automated/interactive theorem provers, and in Boolean satisfiability (SAT) problems and quantified Boolean formula (QBF) solvers. Machine learning was applied to demonstrate that the performance of cutting edge algorithms as well as actual solutions can be accurately predicted based on cheaply computable features.

6.1 Machine learning for first-order theorem proving

In general, the proof finding process in theorem proving involves a very large search space. Research has been done to improve proof automation by applying machine learning techniques to learn from proof search heuristics.

E is an automatic theorem prover for first order logic with equality developed by Schulz [94]. In E, there is an automatic mode, which picks heuristics for selecting clauses, literals, the term ordering, and other parameters using binary or ternary valued features. The features include properties like "Are the axioms (non-negative clauses) in the problem unit, Horn, or non-Horn clauses?", "Do the goals contain variables or are they ground?". The space of all problems is partitioned into different classes based on these features. Each of the resulting classes is assigned to a separate search heuristic, which is selected using prior experimental results. In general the automatic mode shows a better result than the first heuristic picked even by an expert. However, since the classes are predefined by the

developer, there is also a risk of over-specialization.

Recent work by Bridge et al. [14, 13] took a different approach to tackling the heuristic selection problem in E. The approach taken was to use existing heuristics and to automatically learn to select a good heuristic using simple features of the conjecture and the associated axioms. Five top-performance heuristics (most likely to be selected by the auto mode of E) were selected for the experiment. Heuristic selection based on simple features of the conjecture and the associated axioms was shown to do better than any single heuristic, and did much better than random heuristic selection. Bridge conducted feature selection experiments; the results showed only a few features were required for effective learning. Compared to the automatic mode in E, this approach allows real-valued features to be used for learning and assumes fewer predefined connections between features and heuristics. The approach described in this dissertation is based on this work, while different application areas are investigated. The initial motivation for exploring machine learning in computer algebra systems largely follows from the success of this research.

6.2 Machine learning for axiom selection

Most automated theorem provers fail to discharge proof obligations generated by interactive theorem provers if there are a large number of background facts involved. Irrelevant clauses in resolution problems increase the search space, making proofs hard to find within a reasonable time limit. As a result, it is essential to find ways to automatically select relevant axioms. Sledgehammer is a subsystem of Isabelle/HOL [80], which discharges interactive proof goals with assistance from automatic theorem provers. Recent research done by Kühlwein et al. [70] applies machine learning to Sledgehammer [86]. Mash [70] implements a weighted sparse naive Bayes algorithm. When Sledgehammer is invoked, it exports new facts and their proofs to the machine learner and queries it to obtain relevant facts. Overall, this work goes some way to showing that there is a place for machine learning in theorem proving, and that useful results can be found.

Earlier research has applied machine learning for axiom selection for systems based on set theory and higher-order logic. Urban's Mizar Proof Advisor (MPA) is able to select suitable axioms from the huge `Mizar` library for an arbitrary problem within the `Mizar` Problems for Theorem Proving (MPTP) [101]. MPTP is a system for translating the `Mizar Mathematical Library (MML)` [92] into untyped first-order format suitable for automated theorem provers and for generating corresponding theorem-proving problems. MPA was used to improve the performance of the MPTP system by applying machine learning to the previous proof experience extracted from `MML`, and then suggesting a limited number of premises that are most likely to be useful for proving an arbitrary formula. A feature-based machine learning framework was used, where features are symbols presented in formulas. The training system runs on the existing server mode implemented in SNoW [22], using a naive Bayes algorithm. The output for the learner is `MML` theorems ordered by their expected utility (chance of being useful in the proof).

Following the success of MPA, Urban developed the MaLARea System (machine learning for automated reasoning) [102] applying similar ideas. The SNoW system with theDistribution and Correlation of Features naive Bayesian learning mode was also used here. Information about all successful proofs found so far is collected and used for training. Each training example contains all the symbols of a solved conjecture as features and the names of the axioms needed for its proof as the target output. The trained classifier is

then used to prune axioms for future runs. A list of axiom names was ranked according to their expected usefulness, and this ranking was further used for future proofs (the top n high rank axioms from the classifier were selected if the next run will limit the number of axioms to n).

6.3 Machine learning for interactive theorem proving

Komendantskaya and Heras's ML4PG project [68] attempts to show that it is possible to apply machine learning to interactive theorem provers. The ML4PG system provides a link between a theorem prover (Coq) [6] and machine learning tools (MATLAB [90] and Weka [51]). It allows users to gather patterns of the form proof tree, goal structures or proof steps. The gathered data is passed to the machine learning tool, which returns clusters of lemmas that show similarities to the non-trivial proof. The results of clustering give hints to users to further inspect the proofs of the clustered lemmas to see if there is a proof pattern that could help advance the problematic proof. In their conclusions, it would appear that clustering based on goal structures provides the best results.

Kaliszyk et al. [64] took a different approach by learning proof dependencies from formalizations done in the Coq system. A feature-based machine learning framework was used, where the features are the set of defined constants that appear in theorems and definitions. Three machine learning methods (Naive Bayes [9], K-Nearest Neighbour (KNN) [28] and a modified version of the MePo filter [76]) and their combinations were compared on a dataset of 5021 toplevel Coq proofs coming from the `CoRN` repository [31]. Each of the learning algorithms orders the available premises by their likelihood of usefulness in proving the goal. Combining the three learning methods resulted in the best performance, which suggested on average 75% of the needed proof dependencies among the first 100 predictions. This experiment shows that learning the relevant parts of the library necessary for proving a new Coq conjecture is possible.

6.4 Machine learning for SAT solvers

SAT solvers are programs used to automatically decide whether a propositional logic formula is satisfiable or not. They have been used in many formal verification and program analysis applications. Many SAT solvers exist [47], and different solvers perform best on different set of problems. Machine learning can be applied to select the best solver for the given problems.

Xu et al. developed a system called SATzilla2007 [110], which builds runtime prediction models using linear regression techniques based on features extracted from the Boolean satisfiability problems and the algorithm's past performance. This approach is often referred to as the *empirical hardness* approach. They computed the runtime for each selected candidate solver on each problem, and identified distinct features suggested by domain experts (most of the features are derived from those suggested by Nudelman et al. [81]) for building supervised learning model. The ridge regression model was used to build a model that optimizes a given performance (such as mean runtime, percentage of instances solved, or score in a competition) for each solver. The solver with best predicted performance was selected.

Different from SATzilla2007, their recent version SATzilla2012 [111] is based on

cost-sensitive classification models. For each new instance, a cost-sensitive classification model [100] is applied for every pair of solvers in the portfolio, predicting which solver performs better on a given instance based on the instance features. Finally the solver with highest vote (or the one with the second-highest number of votes if the first one failed) is run. They also pre-determine whether the feature extraction process is too expensive or not using a classification model. A backup solver is used if the feature extraction process is predicted to take too long or the predicted solver could not complete its run.

QBFs are a generalization of the SAT problem in which the variables are allowed to be universally or existentially quantified. Samulowitz and Memisevic [93] explore the use of multinomial logistic regression [53] for QBFs. They developed 10 variable branching heuristics, where machine learning can be used to predict run-times and to choose the optimal heuristic from them. The training set was obtained by running each heuristic for each problem instance: the winning heuristic (that with the fastest runtime) for each problem instance is recorded. They selected 78 features based on the basic properties appropriate for SAT problems, but in addition with those specific to QBFs (e.g. number of universal quantifiers, quantifier alternations).

6.5 Summary

For the work described in this dissertation, a different application area was studied. Based on the promising results obtained by the existing research on applying machine learning techniques to theorem proving (automatic/interactive theorem provers, SAT/QBF solvers), I investigated if machine learning is also applicable in a similar, yet different area: computer algebra systems over RCFs. In computer algebra systems, the fact that the choice of which algorithm to use is dependent on the particular problem being tackled is a good motivation for machine learning since there is hope that relevant features can be extracted. I give further discussion of the key results and future work of this thesis in the next chapter.

Chapter 7

Conclusion

The problem of algorithm selection in computer algebra system is very important and can dramatically affect the feasibility of a solution. But the right choice is not obvious even for an expert in the field. The aim of this thesis was to see if machine learning is applicable to the algorithm selection in computer algebra systems. With promising results from three instances of computer algebra applications over RCFs: (i) choosing decision procedures and time limits in MetiTarski; (ii) choosing a heuristic for CAD variable ordering; (iii) predicting the usefulness of Gröbner basis preconditioning, we can give a positive answer. The remainder of this chapter summarises the key results and suggests how the work can be extended in future.

7.1 Key results

We summarise the key results described in this thesis.

Chapter 3: Choosing decision procedures and time limits in MetiTarski

Machine learning was applied to the problem-dependent selection of the most efficient decision procedure and the selection of the best time limit setting for RCF decision procedure calls in MetiTarski. The experiments were done on 825 MetiTarski problems in a variant of the Thousands of Problems for Theorem Provers (TPTP) format [98]. In the experiment of selecting the best decision procedure, three RCF decision procedures were tested: Z3 with Strategy 1, Mathematica and QEPCAD. For each problem, the best decision procedure to use was determined by running each of the decision procedures on the problem and choosing the one with the fastest runtime. The experimental results confirmed our thesis that no one decision procedure is superior for all problems and the correct choice will depend on the problem. The machine-learned selection process in this application performed better than any fixed decision procedure. As a benchmark, choosing the best decision procedure proved 163 out of 194 problems, showing that machine learned selection achieved an 84% optimal choice.

A further experiment was conducted to select between Z3 and Mathematica and to set the best time limit on RCF calls for a given MetiTarski problem. In each individual run, MetiTarski called Z3 or Mathematica with various time limits on RCF calls: 0.1s, 1s, 10s, or 100s. The machine learned algorithm for selection

performed better on our benchmark set than any of the individual fixed settings used in isolation.

Although a small data set was used and some marginal improvement was obtained, it is clear that the machine learned algorithm for selection can be useful, and this motivated our investigation of its application to the field.

Chapter 4: Heuristic selection of CAD variable ordering
The choice of variable ordering for CAD is very important and can dramatically affect the feasibility of a problem. We have investigated the use of machine learning for making the choice of which heuristic to use when selecting a variable ordering for CAD, and quantifier elimination by CAD. Using machine learning to select the best CAD heuristic yielded better results than random choice: selecting an optimal heuristic for 76% of quantifier-free problems and 77% quantified problems (compared with 58% and 64% for a random choice and 64% and 74% for the best performing heuristic (`Brown`)), indicating that there is a relationship between the simple algebraic features and the best heuristic choice. This relationship could be further explored to help develop of new individual heuristics in the future. From the experimental results, the conclusion of which heuristic is the best varies depending on which criteria used to judge. We highlight the strong performance of `Brown` heuristic, surprising both because it requires the least computation and since it is not formally published. (To the best of our knowledge it is mentioned only in notes to a tutorial at ISSAC 2004 [17]).

Chapter 5: Predicting the usefulness of Gröbner basis preconditioning
Applying a Gröbner basis calculation to the systems of polynomial (in)equalities to precondition the input problem before invoking CAD may often reduce the number of cells generated during CAD construction. However, Gröbner basis preconditioning is not always beneficial. I investigated the application of machine learning to the problem of predicting whether Gröbner basis preconditioning is beneficial to CAD problems or not. Three feature sets (*all features*, *before features* and *after features*) were tested in order to understand whether one set appeared more useful than the other. Using machine learned selection (with *all features* and *after features*) yielded better results than always using Gröbner basis preconditioning by default.

A series of feature selection experiments was also conducted. The results show that not all features contribute to the machine learning process; less than half of the features were required for effective learning in this experiment. Using a reduced feature set yielded better results: the feature subset suggested by the filter method successfully predicted average 79% of the problems and using the feature subset suggested by the wrapper method successfully predicted average 78% of the problems from 50 runs of the 5-fold cross validation (compared to 75% when always using Gröbner basis preconditioning by default). Results from feature selection experiments show that the optimal feature subset contains features from both *before features* and *after features*. The properties related to the first projected variable affects heavily to the complexity of the rest of the algorithm.

7.2 Future work

I summarise the possible extensions of this work and ideas for future investigation.

- **Chapter 3: Choosing decision procedures and time limits in MetiTarski**

    1. The current feature set only contains two types of features: one related to the various *special functions* and the other related to the number of variables in the problem. A possible future extension could be exploring more feature types relevant to the machine learned selection process. Possible features could be the number of atomic formulas, the number of symbols and so on. Also, a more sophisticated feature selection process could be applied as described in Chapter 5.
    2. I note there are other variations of RCF strategies which can be called by MetiTarski. For example, interval constraint propagation (icp) [33], QEPCAD with icp, Mathematica with icp, Z3 with icp. Specialised variations of RCF decision procedures have their own strengths and weaknesses for restricted classes of formulas. It would be interesting to test if machine learning can assist with the selection among more strategies.

- **Chapter 4: Heuristic selection of CAD variable ordering**

    1. Although a large data set of real world problems was used, we note that in some ways the data was quite uniform. A key area of future work is experimentation on a wider data set to see if these results, both the benefit of machine learning and the superiority of `Brown` heuristic, hold more generally. An initial extension would be to relax the parameters used to select problems from the `nlsat` dataset, for example by allowing problems with more variables. Another restriction with this dataset is that all problems have one block of existential quantifiers. Possible ways to generalise the data include randomly applying quantifiers to the the existing problems, or randomly generating whole problems as in Chapter 5.
    2. We do not suggest SVMs as the only suitable machine learning method for this experiment, but overall an SVM with a radial basis function kernel worked well here. It would be interesting to see if other machine learning methods could offer similar or even better selections. Further improvements may also come from more work on feature selection. The features used here were all derived from the polynomials in the input. One possible extension would be to consider also the types of relations present and how they are connected logically (likely to be particularly beneficial if problems with more variables or more varied quantifiers are allowed).
    3. Other heuristics can also be tested, for example the greedy `sotd` heuristic [36] which chooses an ordering one variable at a time based on the `sotd` of new projection polynomials or combined heuristics, (where we narrow the selection with one and then break the tie with another).

Finally, we note that there are other CAD implementations. In addition to QEPCAD there is `ProjectionCAD` [39], MATHEMATICA [97], `RegularChains` [23] and `SyNRAC` [60] in MAPLE and `Redlog` [37] in REDUCE. Each implementation has its own intricacies and often different underlying theory so it would be interesting to test if machine learning can assist with these as it does with QEPCAD.

- **Chapter 5: Predicting the usefulness of Gröbner basis preconditioning**

1. Randomly generated problems were used in the experiment due to the limited availability of data sets, thus the data was quite uniform. It would be interesting to experiment with a wider data set to see if these results, both the benefit of machine learning and the superiority of Gröbner basis preconditioning, hold more generally.
2. In the work described in this thesis, Gröbner basis preconditioning replaces equalities with their Gröbner basis (with respect to the same fixed variable ordering as the CAD is constructed with). Ideally, we could also investigate Gröbner basis preconditioning together with variable ordering. A key extension for future work will be to see if machine learning can offer benefit in the prediction of both variable ordering and the usefulness of Gröbner preconditioning together in the formulation of problems for CAD.

7.3 Final remarks

My aim in this thesis was to see whether machine learning is applicable to the algorithm selection problem arising in computer algebra systems, in particular on problems in RCFs. I performed three substantial experiments and each one yielded a positive outcome. This confirms my thesis that machine learning can be beneficial for computer algebra. I did not select computer algebra problems for my experiments according to their likelihood of being amenable to machine learning: indeed, RCFs are not unique amongst computer algebra problems in requiring the user to make an algorithm selection. I hope that the positive results I have demonstrated will encourage further application of machine learning in the field of symbolic computation.

Bibliography

[1] Behzad Akbarpour and Lawrence C Paulson. Extending a resolution prover for inequalities on elementary functions. In *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 47–61. Springer, 2007.

[2] Behzad Akbarpour and Lawrence C Paulson. Metitarski: An automatic theorem prover for real-valued special functions. *Journal of Automated Reasoning*, 3(44):175–205, 2010.

[3] MA Anusuya and Shriniwas K Katti. Speech recognition by machine, a review. *arXiv preprint arXiv:1001.2267*, 2010.

[4] D. Arnon, G.E. Collins, and S. McCallum. Cylindrical algebraic decomposition I: The basic algorithm. *SIAM Journal of Computing*, 13:865–877, 1984.

[5] Pierre Baldi, Søren Brunak, Yves Chauvin, Claus AF Andersen, and Henrik Nielsen. Assessing the accuracy of prediction algorithms for classification: an overview. *Bioinformatics*, 16(5):412–424, 2000.

[6] Bruno Barras, Samuel Boutin, Cristina Cornes, Judicaël Courant, Jean-Christophe Filliatre, Eduardo Gimenez, Hugo Herbelin, Gerard Huet, Cesar Munoz, Chetan Murthy, et al. The Coq proof assistant reference manual: Version 6.1. 1997.

[7] S. Basu. Algorithms in real algebraic geometry: A survey. Available from: `www.math.purdue.edu/∼sbasu/raag_survey2011_final.pdf`, 2011.

[8] Kristin Bennett and OL Mangasarian. Combining support vector and mathematical programming methods for induction. *Advances in Kernel Methods—SV Learning*, pages 307–326, 1999.

[9] Christopher M Bishop et al. *Pattern Recognition and Machine Learning*, volume 1. springer, 2006.

[10] Jacek Bochnak, Michel Coste, and Marie-Françoise Roy. *Real algebraic geometry*. Springer, 1998.

[11] R. Bradford, J.H. Davenport, M. England, and D. Wilson. Optimising problem formulations for cylindrical algebraic decomposition. In J. Carette, D. Aspinall, C. Lange, P. Sojka, and W. Windsteiger, editors, *Intelligent Computer Mathematics*, volume 7961 of *Lecture Notes in Computer Science*, pages 19–34. Springer, 2013.

[12] Russell Bradford, James H Davenport, Matthew England, and David Wilson. Optimising problem formulation for cylindrical algebraic decomposition. In *Intelligent Computer Mathematics*, pages 19–34. Springer, 2013.

[13] James P. Bridge. Machine learning and automated theorem proving. Technical Report UCAM-CL-TR-792, University of Cambridge, Computer Laboratory, November 2010.

[14] James P. Bridge, Sean B. Holden, and Lawrence C. Paulson. Machine learning for first-order theorem proving - learning to select a good heuristic. *J. Autom. Reasoning*, 53(2):141–172, 2014.

[15] Christopher W Brown. QEPCAD B: a program for computing with semi-algebraic sets using CADs. *ACM SIGSAM Bulletin*, 37(4):97–108, 2003.

[16] C.W. Brown. QEPCAD B: A program for computing with semi-algebraic sets using CADs. *ACM SIGSAM Bulletin*, 37(4):97–108, 2003.

[17] C.W. Brown. Companion to the Tutorial: Cylindrical algebraic decomposition, presented at ISSAC '04. Available from: `www.usna.edu/Users/cs/wcbrown/research/ISSAC04/handout.pdf`, 2004.

[18] C.W. Brown and J.H. Davenport. The complexity of quantifier elimination and cylindrical algebraic decomposition. In *International Symposium on Symbolic and Algebraic Computation*, ISSAC '07, pages 54–60. ACM, 2007.

[19] C.W. Brown, M. El Kahoui, D. Novotni, and A. Weber. Algorithmic methods for investigating equilibria in epidemic modelling. *Journal of Symbolic Computation*, 41:1157–1173, 2006.

[20] Bruno Buchberger. Bruno buchbergers PhD thesis 1965: An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal. *Journal of symbolic computation*, 41(3):475–511, 2006.

[21] Jacques Carette. Understanding expression simplification. In *International Symposium on Symbolic and Algebraic Computation*, pages 72–79. ACM, 2004.

[22] Andrew J Carlson, Chad M Cumby, Jeff L Rosen, and Dan Roth. Snow user guide. 1999.

[23] C. Chen, M. Moreno Maza, B. Xia, and L. Yang. Computing cylindrical algebraic decomposition via triangular decomposition. In *International Symposium on Symbolic and Algebraic Computation*, ISSAC '09, pages 95–102. ACM, 2009.

[24] Changbo Chen, Marc Moreno Maza, Bican Xia, and Lu Yang. Computing cylindrical algebraic decomposition via triangular decomposition. In *International Symposium on Symbolic and algebraic computation*, pages 95–102. ACM, 2009.

[25] Matthew Clegg, Jeffery Edmonds, and Russell Impagliazzo. Using the Gröbner basis algorithm to find proofs of unsatisfiability. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 174–183. ACM, 1996.

[26] G.E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Proceedings of the 2nd GI Conference on Automata Theory and Formal Languages*, pages 134–183. Springer, 1975.

[27] G.E. Collins and H. Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *Journal of Symbolic Computation*, 12:299–328, 1991.

[28] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21–27, 1967.

[29] David A Cox, John Little, and DONAL OSHEA. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Springer, 2007.

[30] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.

[31] Luís Cruz-Filipe, Herman Geuvers, and Freek Wiedijk. C—CoRN, the constructive Coq repository at Nijmegen. In *Mathematical Knowledge Management*, pages 88–103. Springer, 2004.

[32] J.H. Davenport, R. Bradford, M. England, and D. Wilson. Program verification in the presence of complex numbers, functions with branch cuts etc. In *14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, SYNASC '12, pages 83–88. IEEE, 2012.

[33] Ernest Davis. Constraint propagation with interval labels. *Artificial intelligence*, 32(3):281–331, 1987.

[34] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[35] Leonardo De Moura and Grant Olney Passmore. Computation in real closed infinitesimal and transcendental extensions of the rationals. In *Automated Deduction–CADE-24*, pages 178–192. Springer Berlin Heidelberg, 2013.

[36] A. Dolzmann, A. Seidl, and T. Sturm. Efficient projection orders for CAD. In *Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation*, ISSAC '04, pages 111–118. ACM, 2004.

[37] A. Dolzmann and T. Sturm. REDLOG: Computer algebra meets computer logic. *SIGSAM Bulletin*, 31(2):2–9, 1997.

[38] Carl Engelman. Mathlab: a program for on-line machine assistance in symbolic computations. In *Proceedings of the November 30–December 1, 1965, fall joint computer conference, part II: computers: their impact on society*, pages 117–126. ACM, 1965.

[39] M. England. An implementation of CAD in Maple utilising problem formulation, equational constraints and truth-table invariance. Department of Computer Science Technical Report series 2013-04, University of Bath. Available at `http://opus.bath.ac.uk/35636/`, 2013.

[40] Matthew England, David Wilson, Russell Bradford, and James H Davenport. Using the regular chains library to build cylindrical algebraic decompositions by projecting and lifting. In *Mathematical Software—ICMS 2014*, pages 458–465. Springer, 2014.

[41] Usama M. Fayyad and Keki B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *IJCAI*, pages 1022–1029, 1993.

[42] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15:3133–3181, 2014.

[43] R. Fletcher. *Practical Methods of Optimization; (2Nd Ed.)*. Wiley-Interscience, New York, NY, USA, 1987.

[44] I.A. Fotiou, P.A. Parrilo, and M. Morari. Nonlinear parametric optimization using cylindrical algebraic decomposition. In *Decision and Control, 2005 European Control Conference. CDC-ECC '05.*, pages 3735–3740, 2005.

[45] Richard J Gaylord, Paul R Wellin, Bill Titus, Susan R McKay, Wolfgang Christian, et al. Computer simulations with mathematica: explorations in complex physical and biological systems. *Computers in Physics*, 10(4):349–350, 1996.

[46] Jerry Glynn. *Exploring Math from Algebra to Calculus with Derive: A Mathematical Assistant for Your Personal Computer*. Math Ware, 1992.

[47] Carla P. Gomes, Henry Kautz, Ashish Sabharwal, and Bart Selman. Satisfiability solvers, 2008.

[48] Erich Grädel. *Finite Model Theory and Its Applications*, volume 2. Springer, 2007.

[49] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *The Journal of Machine Learning Research*, 3:1157–1182, 2003.

[50] M Hall. Correlation-based feature selection for discrete and numeric class machine learning. In *Intentional Conference on Machine Learning*. CA. Morgan Kaufmann Publishers, 2000.

[51] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009.

[52] Mark A. Hall and Geoffrey Holmes. Benchmarking attribute selection techniques for discrete class data mining. *IEEE Transactions On Knowledge And Data Engineering*, 15:1437–1447, 2003.

[53] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[54] Simon Haykin and Neural Network. A comprehensive foundation. *Neural Networks*, 2(2004), 2004.

[55] Anthony C Hearn et al. *REDUCE 2 user's manual*. Department of Computer Science, Stanford University, 1970.

[56] André Heck and Wolfram Koepf. *Introduction to MAPLE*, volume 1993. Springer, 1993.

[57] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. A practical guide to support vector classification. Technical report, Department of Computer Science, National Taiwan University, 2003.

[58] Zongyan Huang, Matthew England, David Wilson, James H Davenport, and Lawrence C Paulson. A comparison of three heuristics to choose the variable ordering for cylindrical algebraic decomposition. *ACM Communications in Computer Algebra*, 48(3/4):121–123, 2015.

[59] Zongyan Huang, Matthew England, David J Wilson, James H Davenport, Lawrence C Paulson, and James P Bridge. Applying machine learning to the problem of choosing a heuristic to select the variable ordering for cylindrical algebraic decomposition. In *Intelligent Computer Mathematics — International Conference, CICM 2014, Coimbra, Portugal, July 7-11, 2014. Proceedings*, pages 92–107, 2014.

[60] H. Iwane, H. Yanami, H. Anai, and K. Yokoyama. An effective implementation of a symbolic-numeric cylindrical algebraic decomposition for quantifier elimination. In *Conference on Symbolic Numeric Computation*, SNC '09, pages 55–64, 2009.

[61] Hidenao Iwane, Hitoshi Yanami, and Hirokazu Anai. An effective implementation of a symbolic-numeric cylindrical algebraic decomposition for optimization problems. In *Proceedings of the 2011 International Workshop on Symbolic-Numeric Computation*, SNC '11, pages 168–177, New York, NY, USA, 2011. ACM.

[62] T. Joachims. Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, chapter 11, pages 169–184. MIT Press, Cambridge, MA, 1999.

[63] Dejan Jovanović and Leonardo De Moura. Solving non-linear arithmetic. In *Automated Reasoning*, pages 339–354. Springer, 2012.

[64] Cezary Kaliszyk, Lionel Mamane, and Josef Urban. Machine learning of Coq proof guidance: First experiments. In *6th International Symposium on Symbolic Computation in Software Science, SCSS 2014, Gammarth, La Marsa, Tunisia, December 7-8, 2014*, pages 27–34, 2014.

[65] S. Sathiya Keerthi and Chih-Jen Lin. Asymptotic behaviors of support vector machines with gaussian kernel. *Neural Comput.*, 15(7):1667–1689, July 2003.

[66] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, pages 1137–1143, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[67] Ron Kohavi and George H John. Wrappers for feature subset selection. *Artificial intelligence*, 97(1):273–324, 1997.

[68] Ekaterina Komendantskaya, Jónathan Heras, and Gudmund Grov. Machine learning in proof general: interfacing interfaces. *arXiv preprint arXiv:1212.3618*, 2012.

[69] Igor Kononenko. Machine learning for medical diagnosis: history, state of the art and perspective. *Artificial Intelligence in medicine*, 23(1):89–109, 2001.

[70] Daniel Kühlwein, Jasmin Christian Blanchette, Cezary Kaliszyk, and Josef Urban. Mash: Machine learning for sledgehammer. In *Interactive Theorem Proving*, pages 35–50. Springer, 2013.

[71] Hsuan-Tien Lin and Chih-Jen Lin. A study on sigmoid kernels for SVM and the training of non-PSD kernels by SMO-type methods. Technical report, Department of Computer Science, National Taiwan University, 2003.

[72] Richard Liska, Ladislav Drska, Jiri Limpouch, Milan Sinor, Michael Wester, and Franz Winkler. Computer algebra, algorithms, systems and applications. 1999.

[73] Donald W. Loveland. *Automated Theorem Proving : A Logical Basis*. Fundamental studies in computer science. North-Holland Pub. Co. New York, Amsterdam, New York, 1978.

[74] S. McCallum. An improved projection operation for cylindrical algebraic decomposition. In B. Caviness and J. Johnson, editors, *Quantifier Elimination and Cylindrical Algebraic Decomposition*, Texts & Monographs in Symbolic Computation, pages 242–268. Springer, 1998.

[75] S. McCallum. On projection in CAD-based quantifier elimination with equational constraint. In *International Symposium on Symbolic and Algebraic Computation*, ISSAC '99, pages 145–149. ACM, 1999.

[76] Jia Meng and Lawrence C Paulson. Lightweight relevance filtering for machine-generated resolution problems. *Journal of Applied Logic*, 7(1):41–57, 2009.

[77] J. Mercer. Functions of positive and negative type, and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 209(441-458):415–446, 1909.

[78] Katharina Morik, Peter Brockhausen, and Thorsten Joachims. Combining statistical learning with a knowledge-based approach: a case study in intensive care monitoring. Technical report, Technical Report, SFB 475: Komplexitätsreduktion in Multivariaten Datenstrukturen, Universität Dortmund, 1999.

[79] New York University. The benchmarks used in solving nonlinear arithmetic. Online at `http://cs.nyu.edu/ dejan/nonlinear/`, 2012.

[80] Tobias Nipkow, Lawrence C Paulson, and Markus Wenzel. *Isabelle/HOL: a proof assistant for higher-order logic*, volume 2283. Springer, 2002.

[81] Eugene Nudelman, Kevin Leyton-Brown, HolgerH. Hoos, Alex Devkar, and Yoav Shoham. Understanding random sat: Beyond the clauses-to-variables ratio. In Mark Wallace, editor, *Principles and Practice of Constraint Programming CP 2004*,

volume 3258 of *Lecture Notes in Computer Science*, pages 438–452. Springer Berlin Heidelberg, 2004.

[82] Edgar Osuna, Robert Freund, and Federico Girosi. Training support vector machines: an application to face detection. In *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, pages 130–136. IEEE, 1997.

[83] Grant Passmore, Lawrence C Paulson, and Leonardo de Moura. Real algebraic strategies for Metitarski proofs. *Intelligent Computer Mathematics*, pages 358–370, 2012.

[84] Grant Olney Passmore and Paul B Jackson. Combined decision techniques for the existential theory of the reals. In *Intelligent Computer Mathematics*, pages 122–137. Springer, 2009.

[85] GrantOlney Passmore and PaulB. Jackson. Abstract partial cylindrical algebraic decomposition i: The lifting phase. In S.Barry Cooper, Anuj Dawar, and Benedikt Lwe, editors, *How the World Computes*, volume 7318 of *Lecture Notes in Computer Science*, pages 560–570. Springer Berlin Heidelberg, 2012.

[86] Lawrence C. Paulson and Jasmin Christian Blanchette. Three years of experience with sledgehammer, a practical link between automatic and interactive theorem provers. In Geoff Sutcliffe, Stephan Schulz, and Eugenia Ternovska, editors, *IWIL 2010*, volume 2 of *EPiC Series*, pages 1–11. EasyChair, 2012.

[87] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C (2Nd Ed.): The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 1992.

[88] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[89] Richard H Rand. *Computer algebra in applied mathematics: An introduction to MACSYMA*. Pitman Boston, 1984.

[90] MATLAB Release. The Mathworks. *Inc., Natick, Massachusetts, United States*, 2012.

[91] John R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.

[92] Piotr Rudnicki. An overview of the Mizar project. In *Workshop on Types for Proofs and Programs*, pages 311–330, 1992.

[93] Horst Samulowitz and Roland Memisevic. Learning to solve QBF. In *Proceedings of the 22Nd National Conference on Artificial Intelligence - Volume 1*, AAAI'07, pages 255–260. AAAI Press, 2007.

[94] S. Schulz. E – A Brainiac Theorem Prover. *Journal of AI Communications*, 15(2/3):111–126, 2002.

[95] Inna K Shingareva and Carlos Lizárraga-Celaya. *Maple and Mathematica: a problem solving approach for mathematics*. Springer, 2009.

[96] H Strubbe. Calculations with SCHOONSCHIP. Technical Report CERN-DD-73-16, CERN, Geneva, Jun 1973. Part of this text will be presented at the Third Colloquium on Advanced Computing Methods in Theoretical Physics held in Marseille on 25-29 June 1973.

[97] A. Strzeboński. Solving polynomial systems over semialgebraic sets represented by cylindrical algebraic formulas. In *International Symposium on Symbolic and Algebraic Computation*, ISSAC '12, pages 335–342. ACM, 2012.

[98] Geoff Sutcliffe. The TPTP problem library and associated infrastructure. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

[99] A. Tarski. A decision method for elementary algebra and geometry. In B.F. Caviness and J.R. Johnson, editors, *Quantifier Elimination and Cylindrical Algebraic Decomposition*, Texts and Monographs in Symbolic Computation, pages 24–84. Springer, 1998.

[100] K. M. Ting. An instance-weighting method to induce cost-sensitive trees. *IEEE Trans. on Knowl. and Data Eng.*, 14(3):659–665, May 2002.

[101] Josef Urban. MPTP–motivation, implementation, first experiments. *Journal of Automated Reasoning*, 33(3-4):319–339, 2004.

[102] Josef Urban. Malarea: a metasystem for automated reasoning in large theories. In *Proceedings of the CADE-21 Workshop on Empirically Successful Automated Reasoning in Large Theories, Bremen, Germany, 17th July 2007*, 2007.

[103] Hal R Varian. *Computational economics and finance: modeling and analysis with Mathematica*, volume 2. Springer, 1996.

[104] Jason Weston and Chris Watkins. Support vector machines for multi-class pattern recognition. In *7th European Symposium on Artificial Neural Networks, Bruges, Belgium, April 21-23, 1999, Proceedings*, pages 219–224, 1999.

[105] D. Wilson, J.H. Davenport, M. England, and R. Bradford. A "piano movers" problem reformulated. In *15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, SYNASC '13. IEEE, 2013.

[106] David J Wilson, Russell J Bradford, and James H Davenport. Speeding up cylindrical algebraic decomposition by gröbner bases. In *Intelligent Computer Mathematics*, pages 280–294. Springer, 2012.

[107] D.J. Wilson, R.J. Bradford, and J.H. Davenport. A repository for CAD examples. *ACM Communications in Computer Algebra*, 46(3):67–69, 2012.

[108] Stephen Wolfram. *The MATHEMATICA® book, version 4*. Cambridge university press, 1999.

[109] Chris Wooff and David Hodgkinson. *muMath: a microcomputer algebra system*. Academic Press Professional, Inc., 1987.

[110] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Satzilla: Portfolio-based algorithm selection for sat. *J. Artif. Int. Res.*, 32(1):565–606, June 2008.

[111] Lin Xu, Frank Hutter, Jonathan Shen, Holger H Hoos, and Kevin Leyton-Brown. Satzilla2012: improved algorithm selection based on cost-sensitive classification models. *Balint et al.(Balint et al., 2012a)*, pages 57–58, 2012.

Appendices

Appendix A

Distribution and correlation of features

In the following figures I present a detailed view of the features used in the experiments described in Chapters 4 and 5. Figures A.1 to A.11 show the distribution of features of the data set used in Chapter 4 in form of histograms, while Figures A.12 to A.39 show the distribution of features of the data in Chapter 5. In addition, I examined some of the pairwise correlations present between features in both sets. In Figures A.40 to A.52, I present the projection of the data set onto a number of feature pairs. For some feature pairs we can observe a noticeable correlation, which is often simply explained by the nature of the feature (e.g. the proportion of x_0 occurring in polynomials is likely to be correlated with its occurrence in monomials). For others, no obvious correlation exists, which adds to justify the inclusion of both features in the set.

Figure A.1: Number of polynomials.

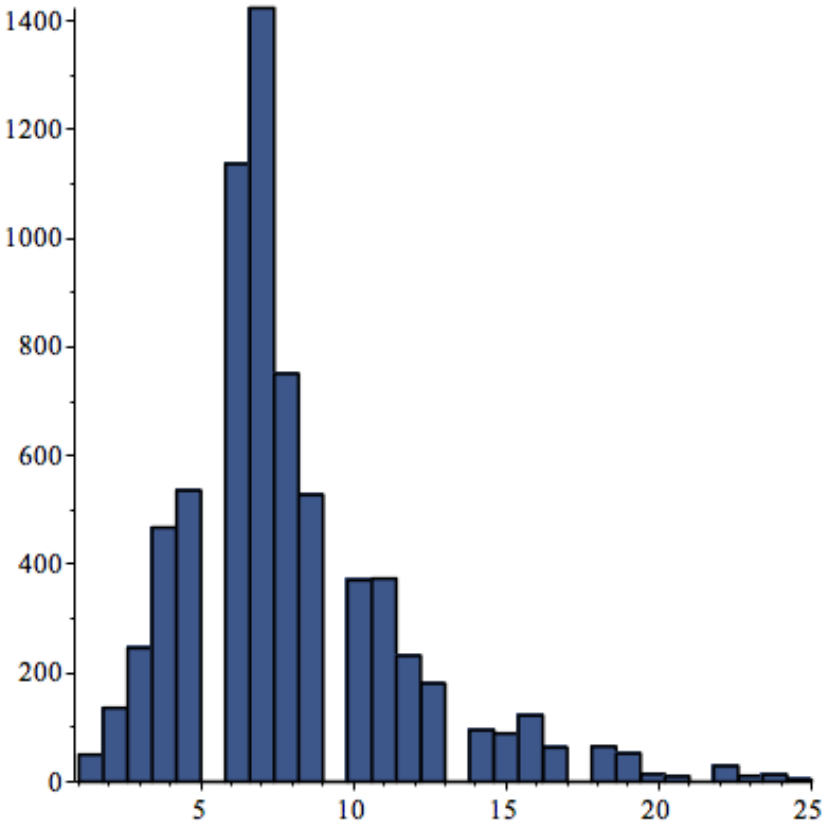


Figure A.2: Maximum total degree of polynomials.

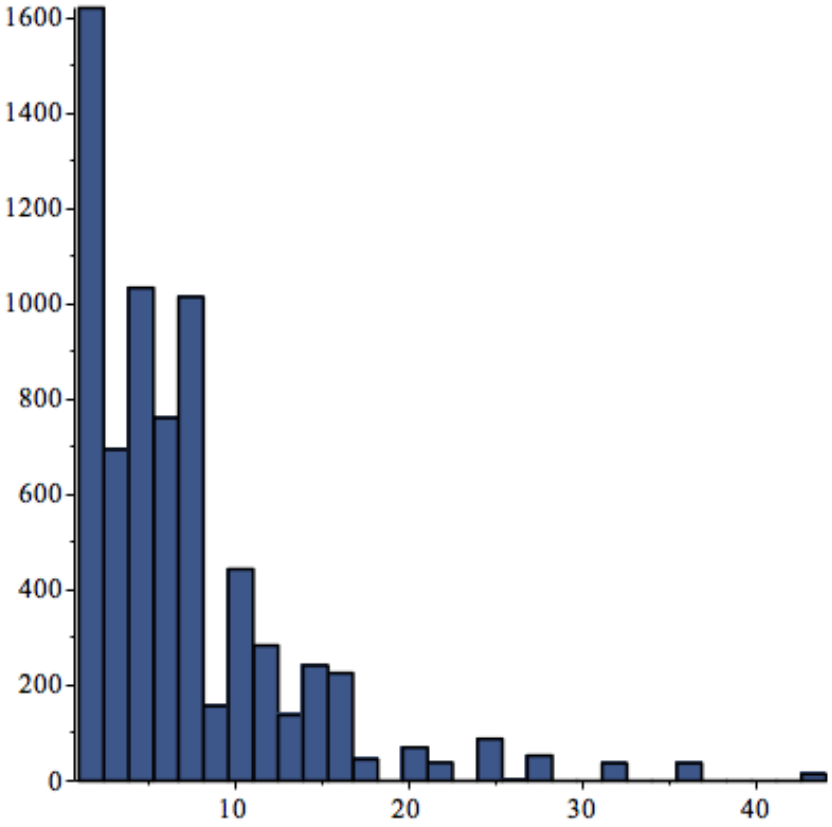


Figure A.3: Maximum degree of x_0 among all polynomials.

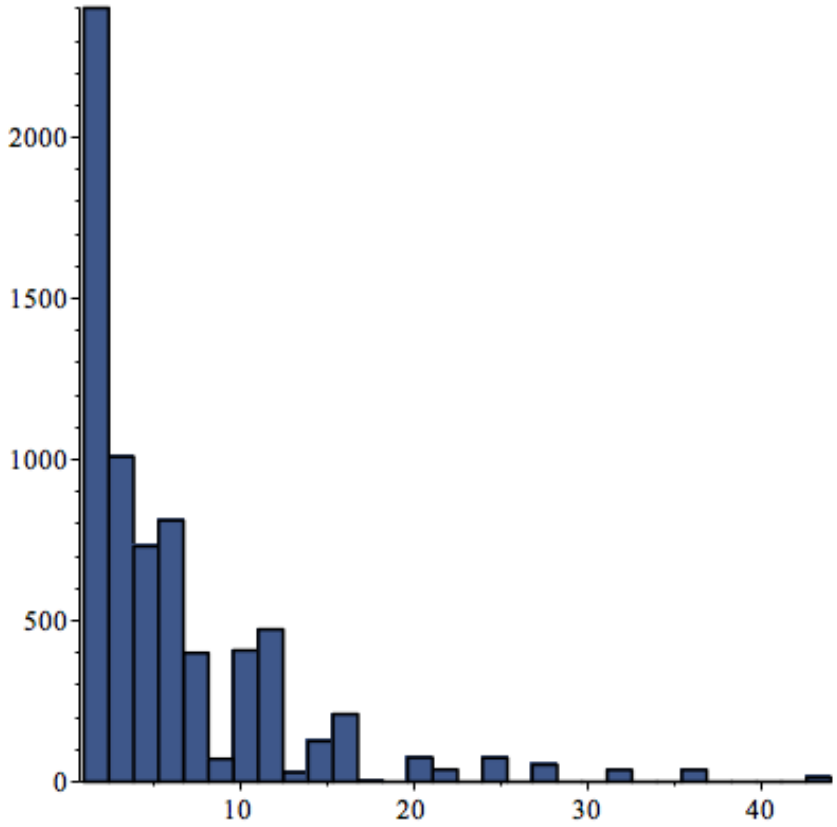


Figure A.4: Maximum degree of x_1 among all polynomials.

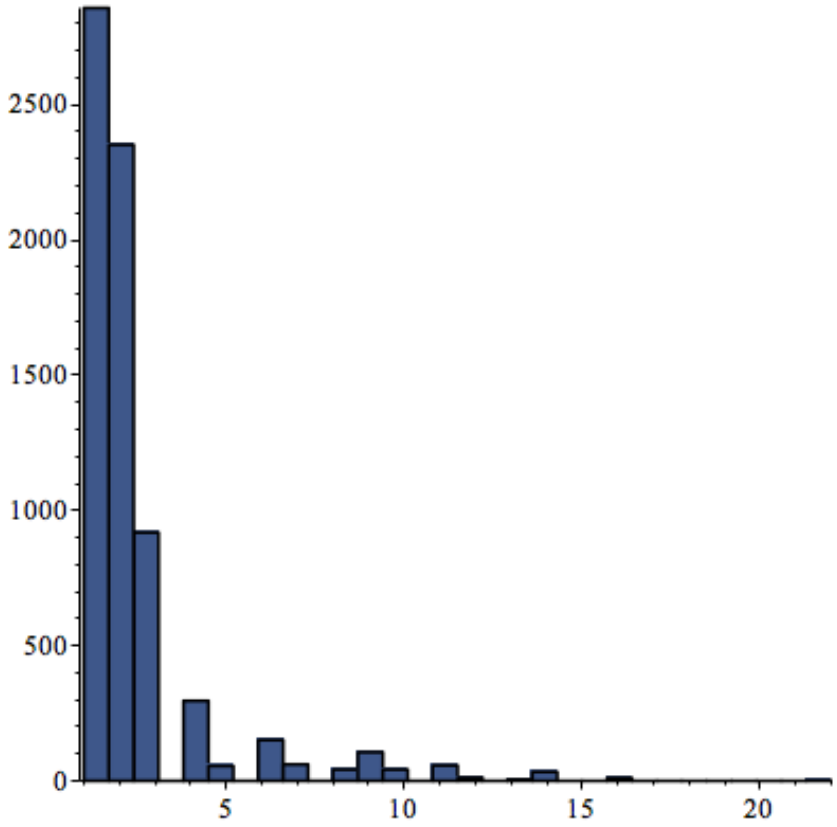


Figure A.5: Maximum degree of x_2 among all polynomials.

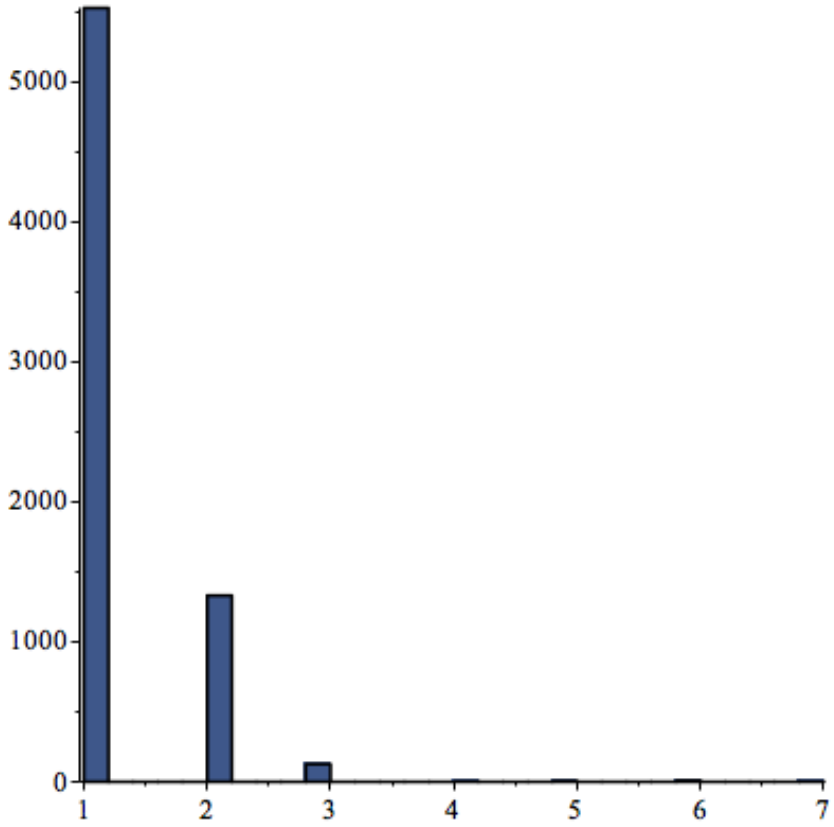


Figure A.6: Proportion of x_0 occurring in polynomials.

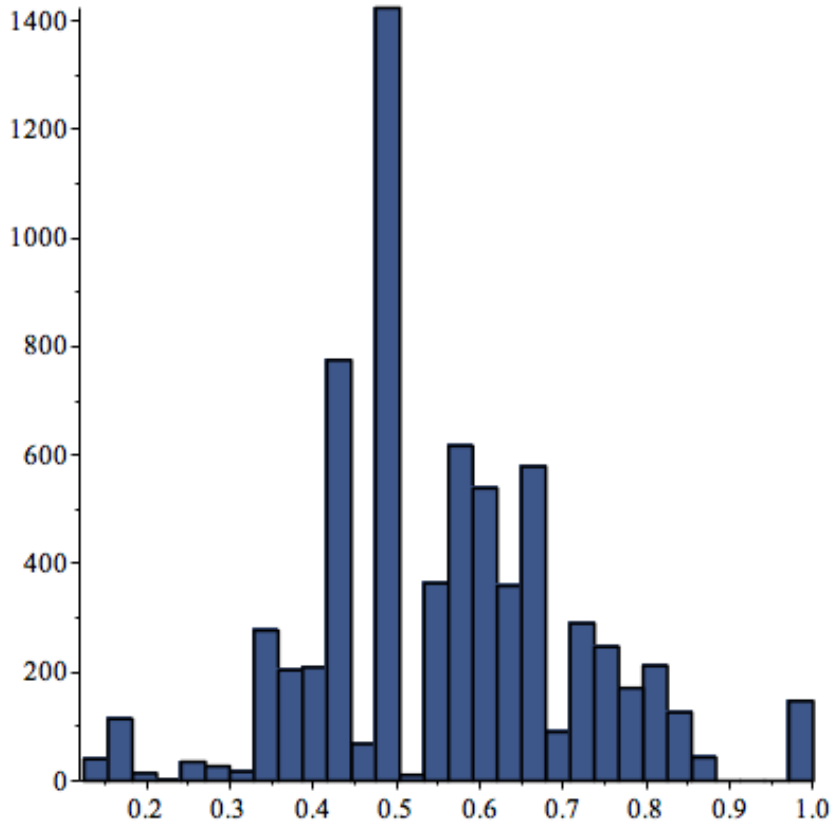


Figure A.7: Proportion of x_1 occurring in polynomials.

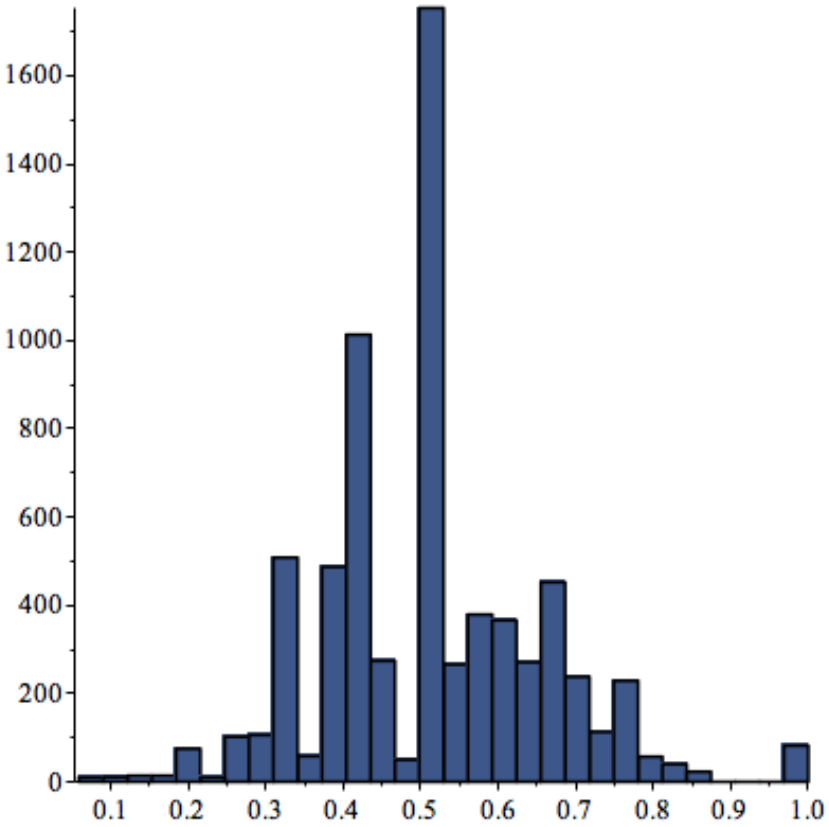


Figure A.8: Proportion of x_2 occurring in polynomials.

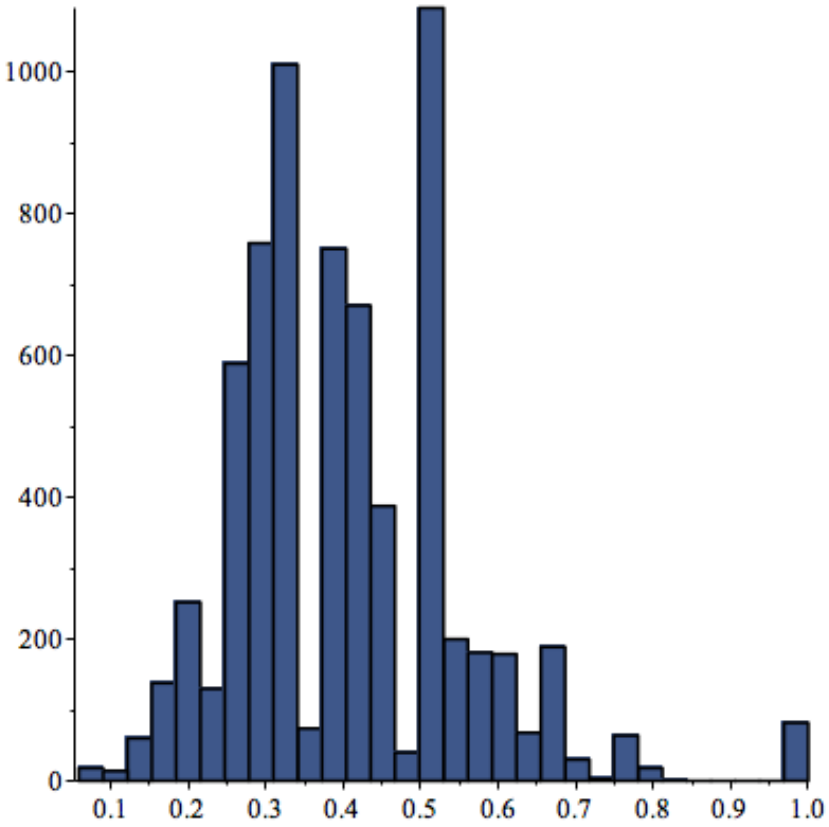


Figure A.9: Proportion of x_0 occurring in monomials.

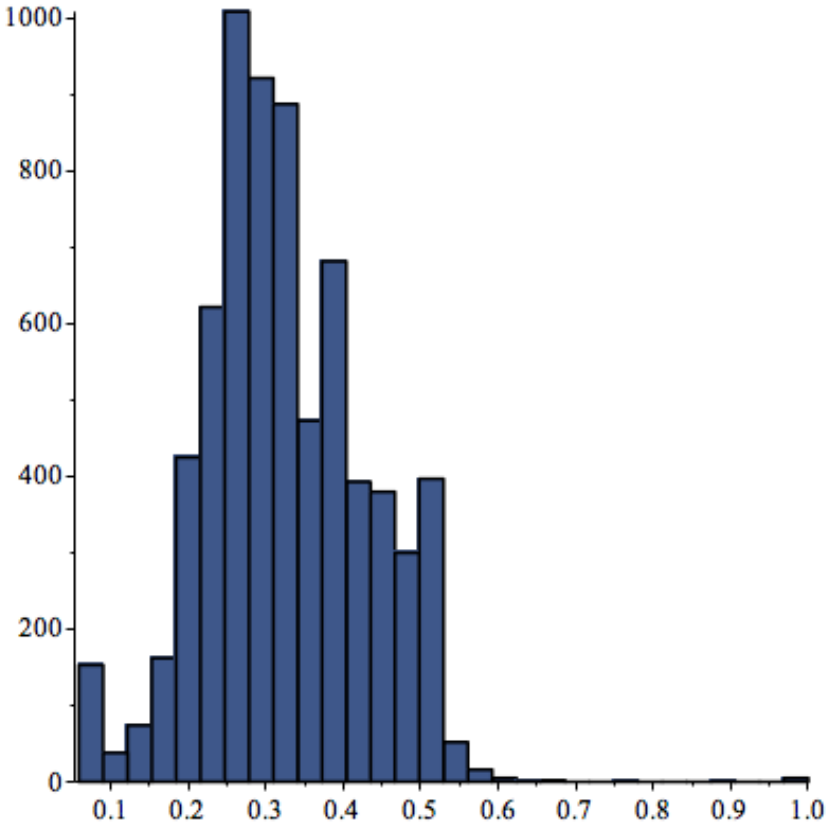


Figure A.10: Proportion of x_1 occurring in monomials.

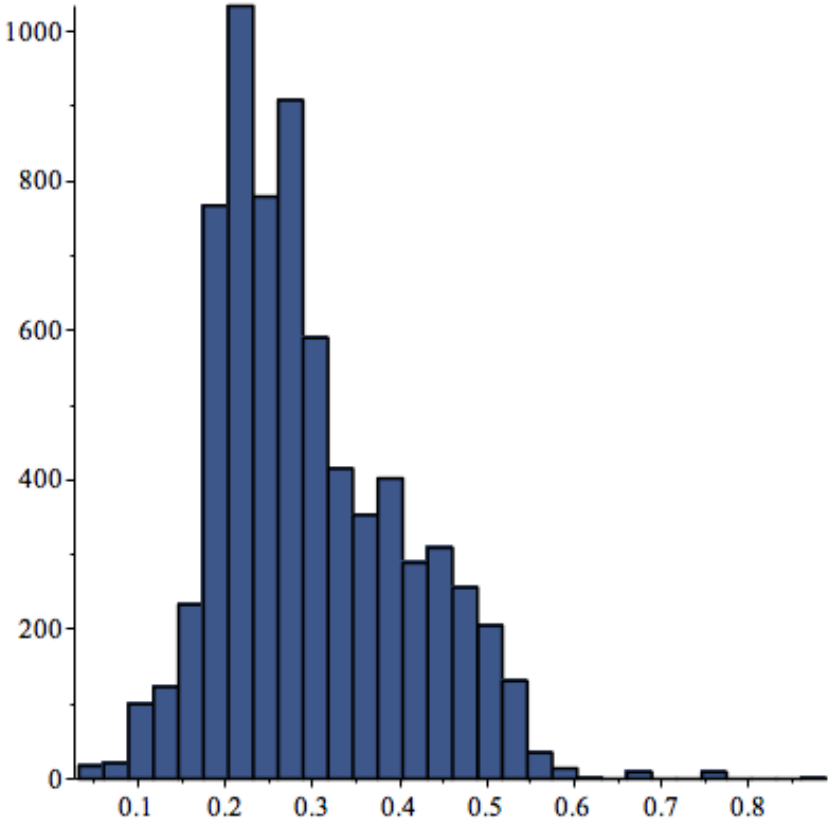


Figure A.11: Proportion of x_2 occurring in monomials.

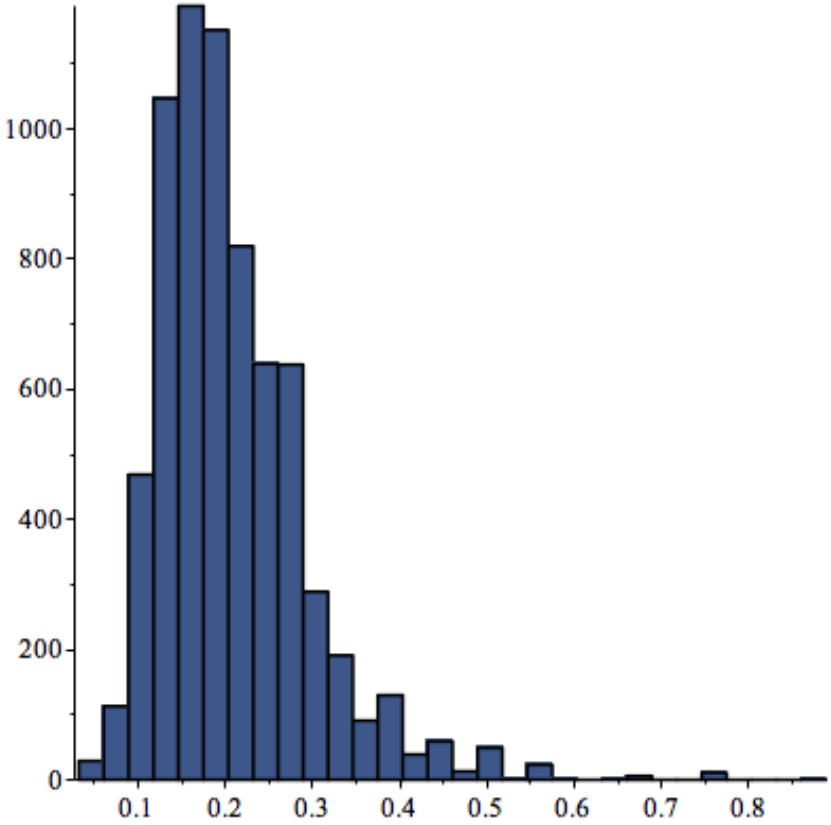


Figure A.12: `TNoI` before GB.

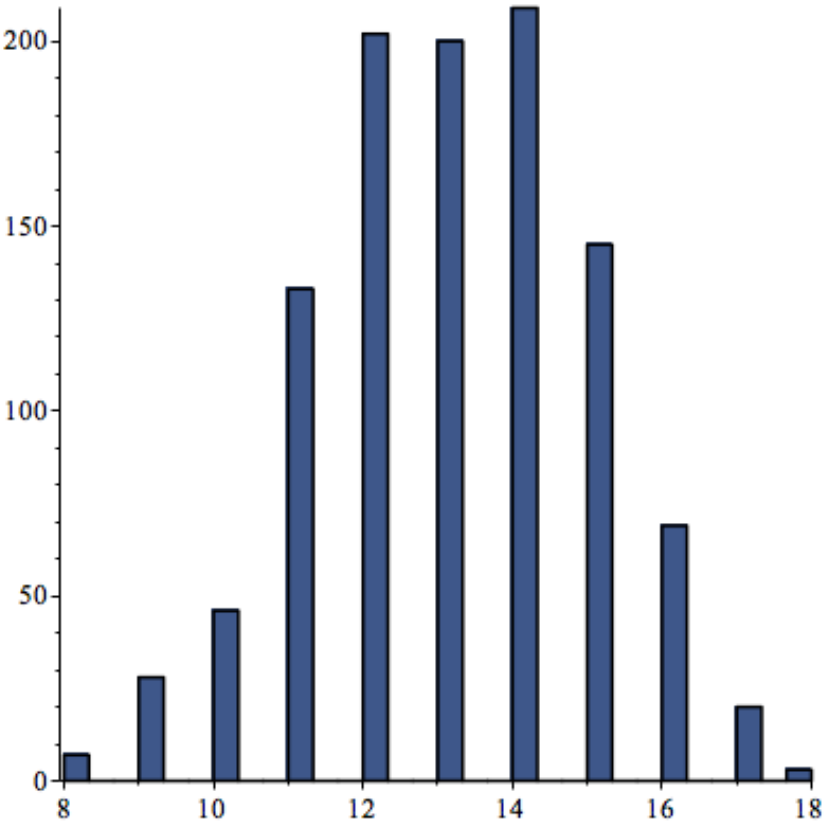


Figure A.13: `stds` before GB.

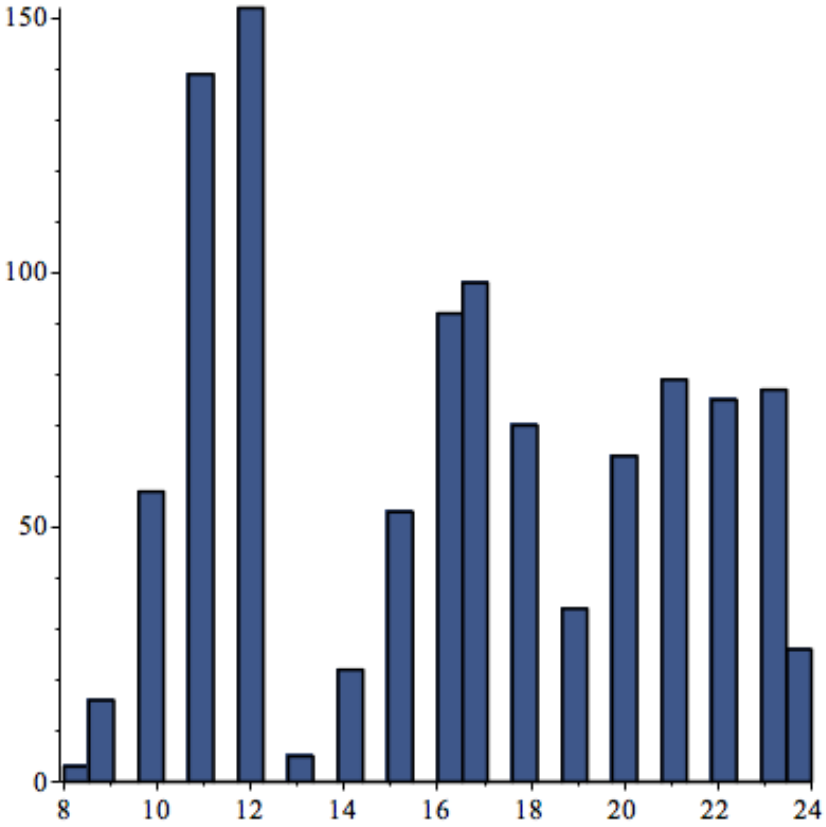


Figure A.14: `tds` of polynomials before GB.

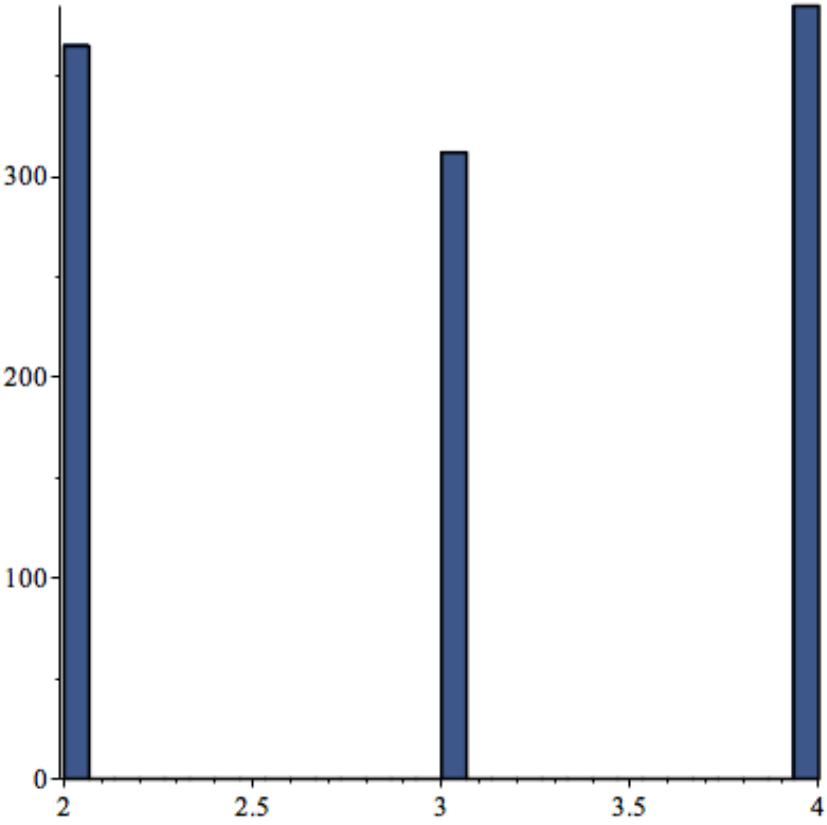


Figure A.15: Maximum degree of x among all polynomials before GB.

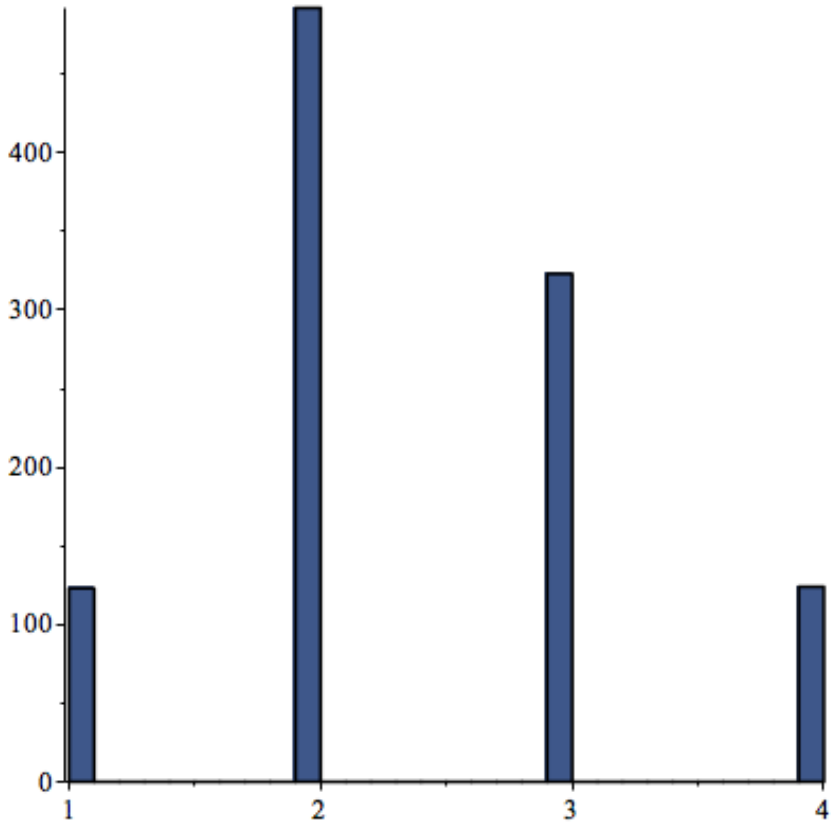


Figure A.16: Maximum degree of y among all polynomials before GB.

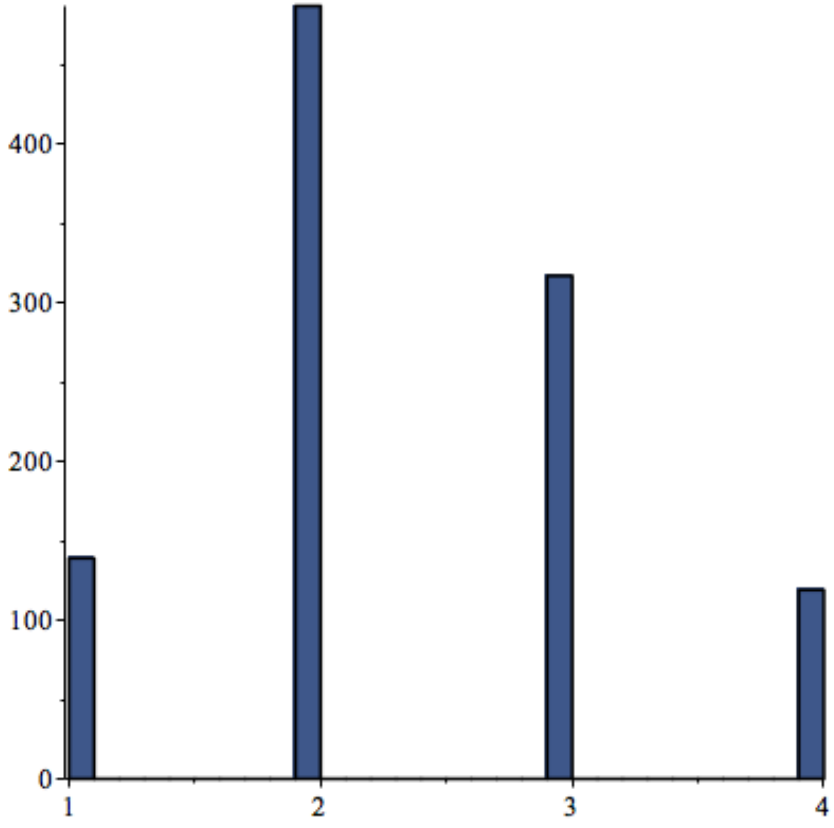


Figure A.17: Maximum degree of z among all polynomials before GB.

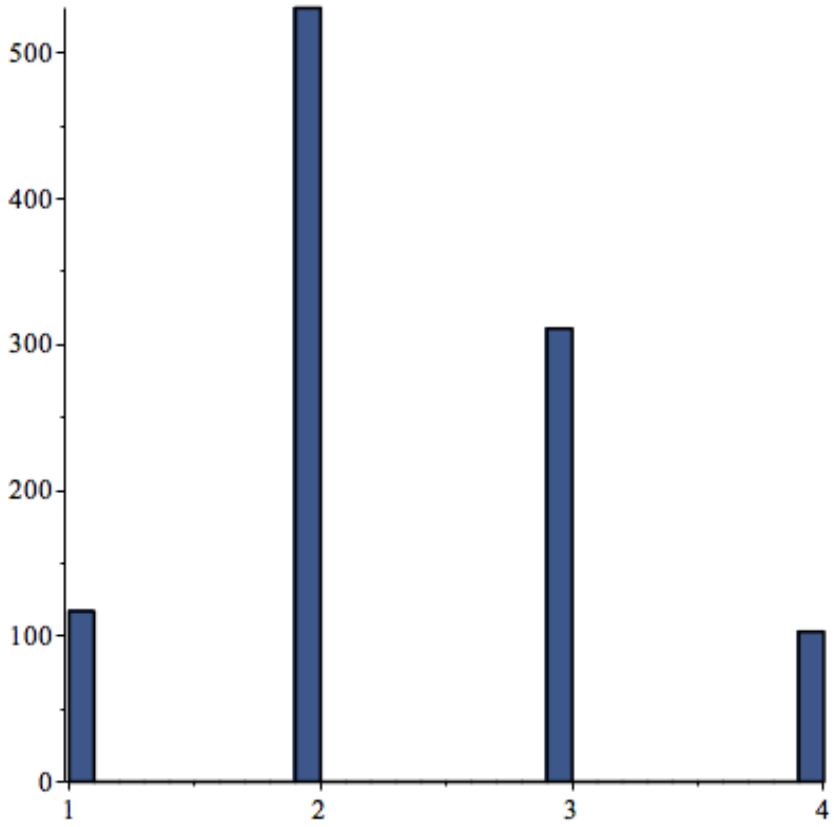


Figure A.18: Proportion of x occurring in polynomials before GB.

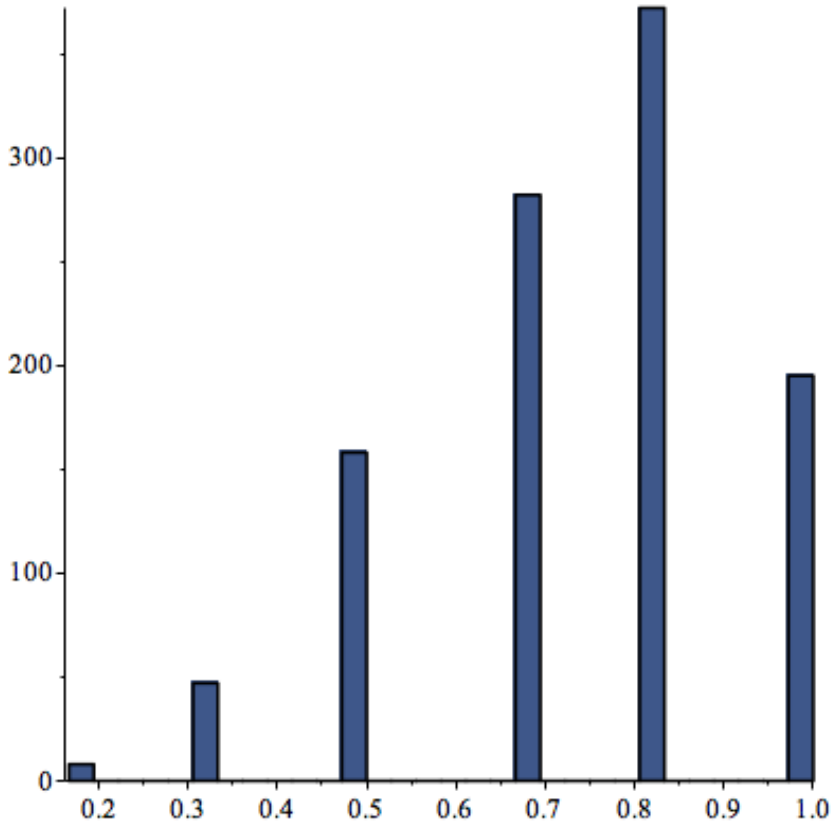


Figure A.19: Proportion of y occurring in polynomials before GB.

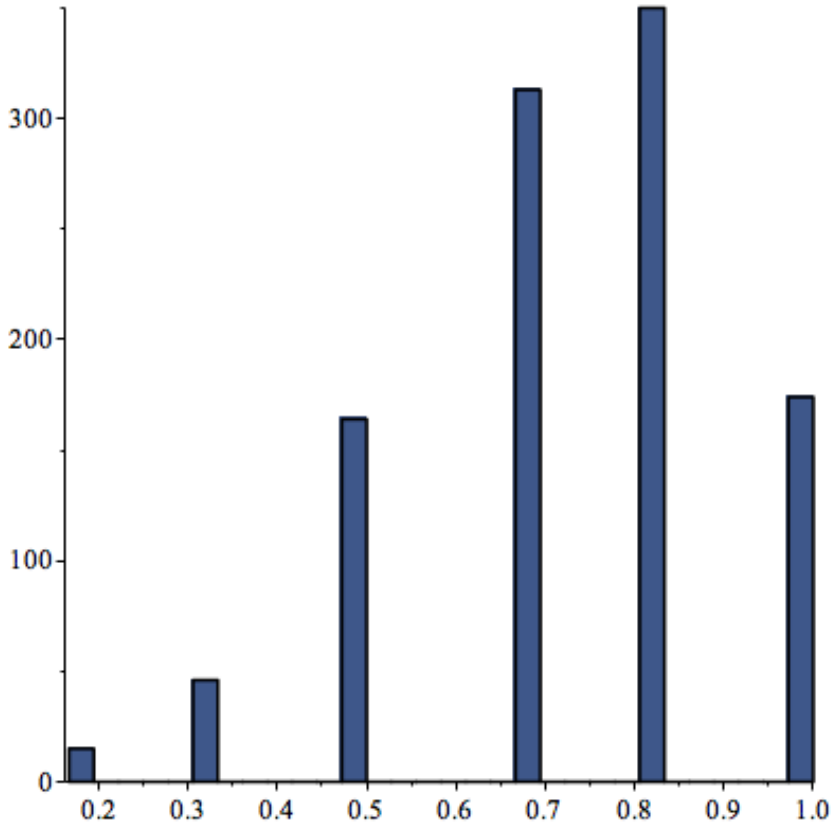


Figure A.20: Proportion of z occurring in polynomials before GB.

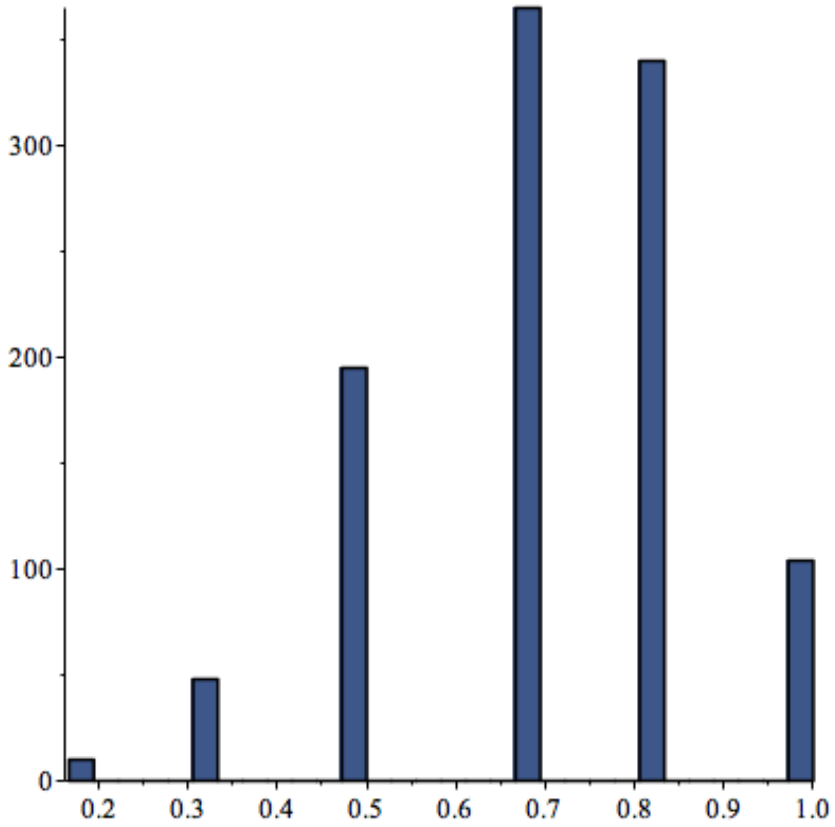


Figure A.21: Proportion of x occurring in monomials before GB.

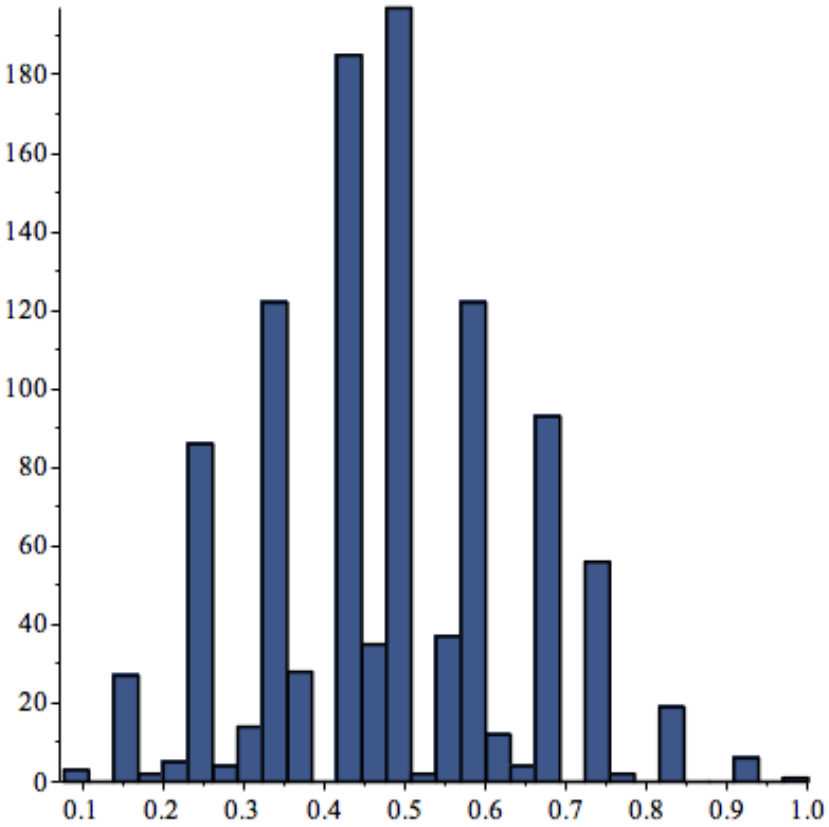


Figure A.22: Proportion of y occurring in monomials before GB.

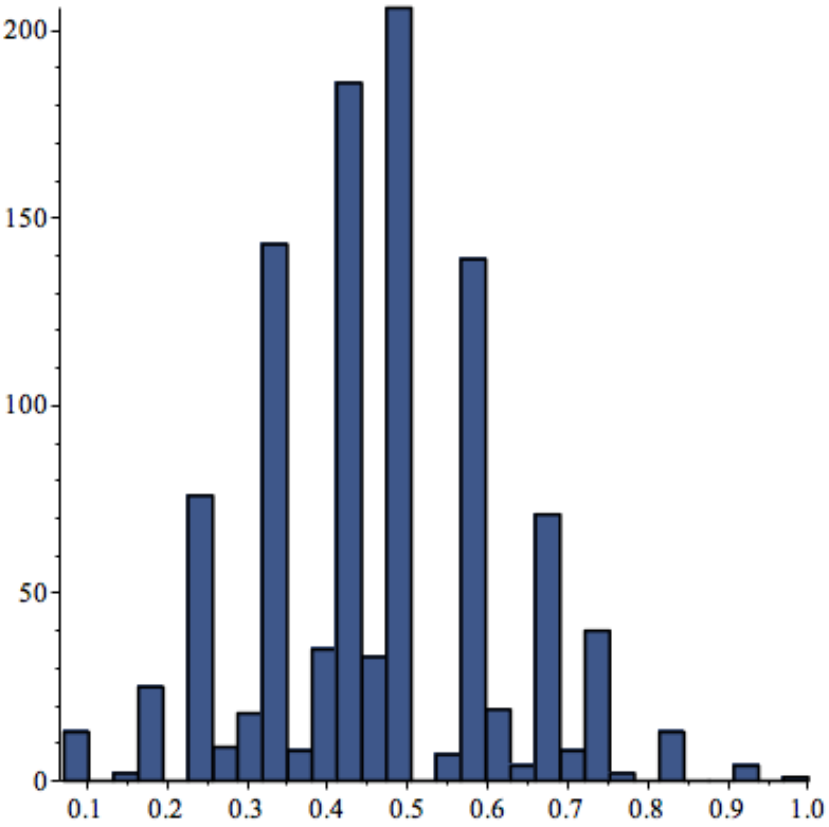


Figure A.23: Proportion of z occurring in monomials before GB.

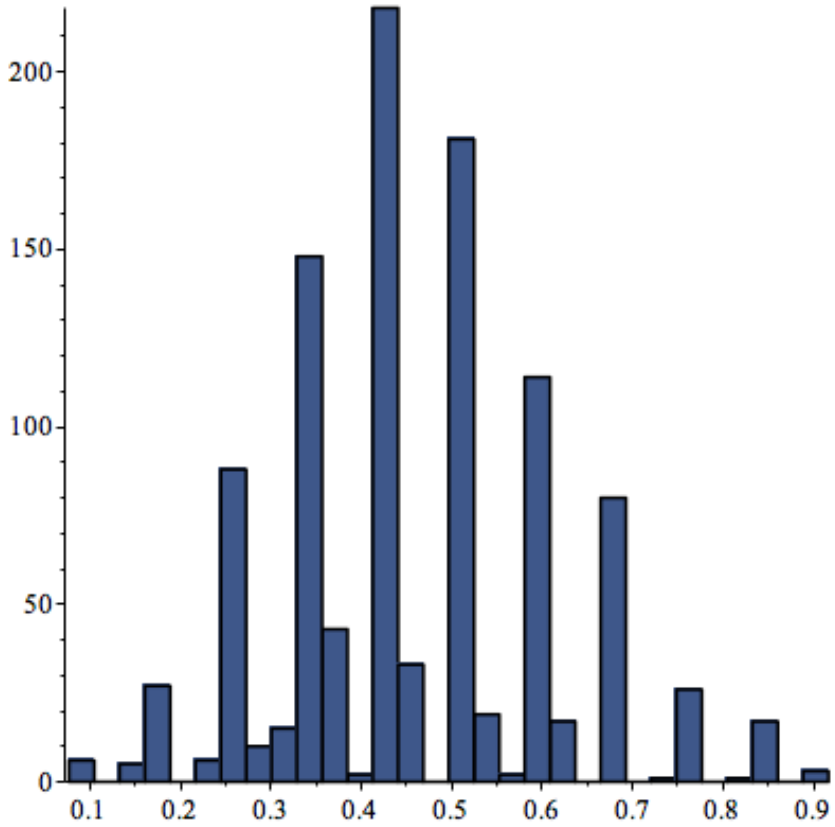


Figure A.24: Number of polynomials after GB.

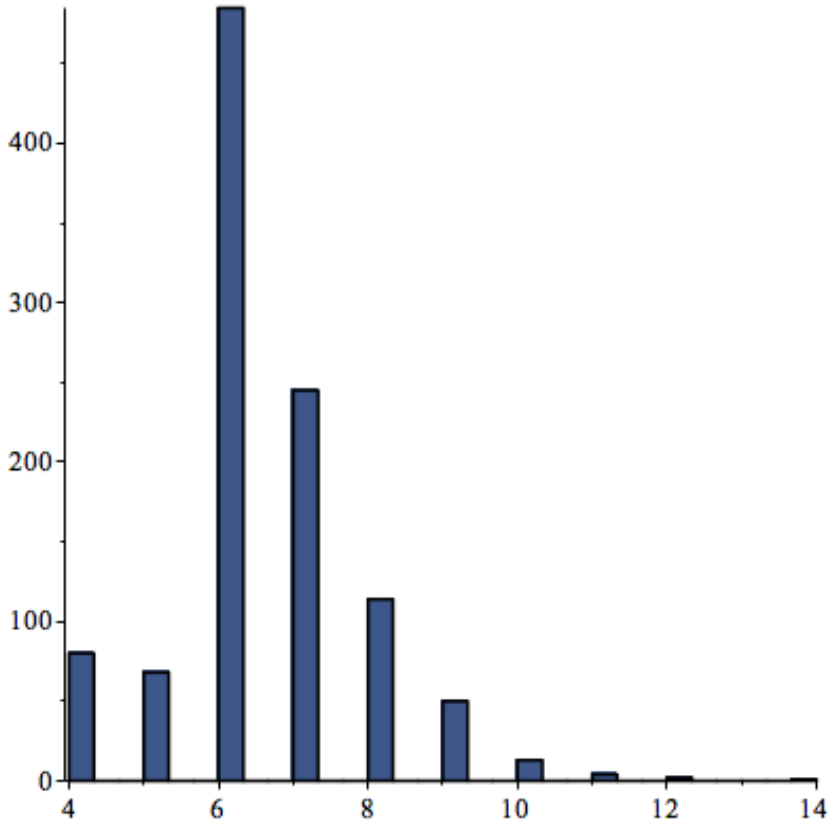


Figure A.25: `TNoI` after GB.

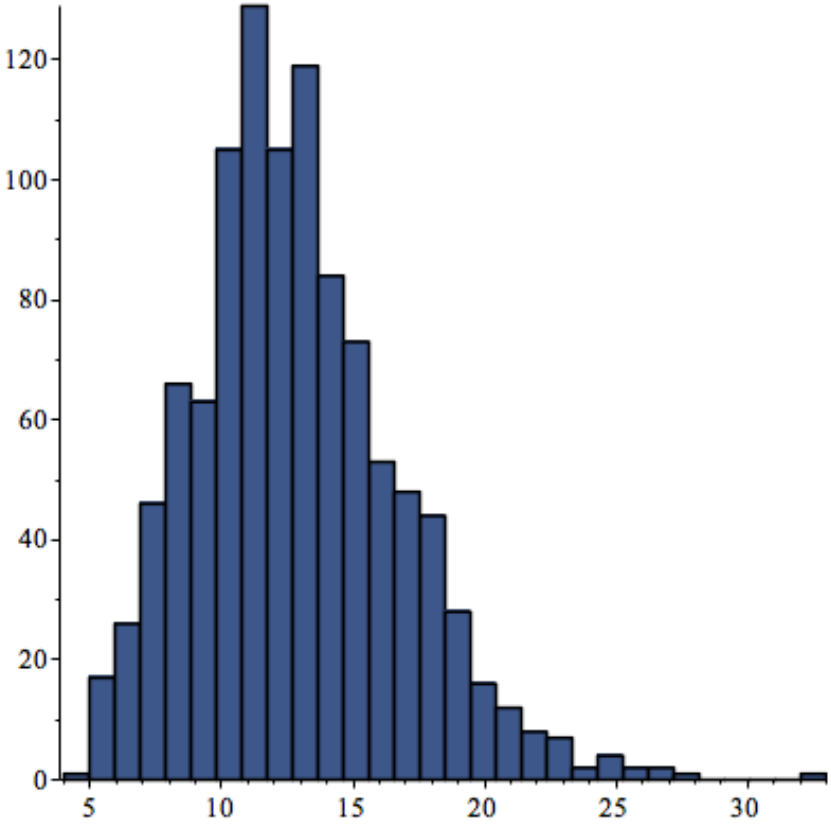


Figure A.26: `stds` after GB.

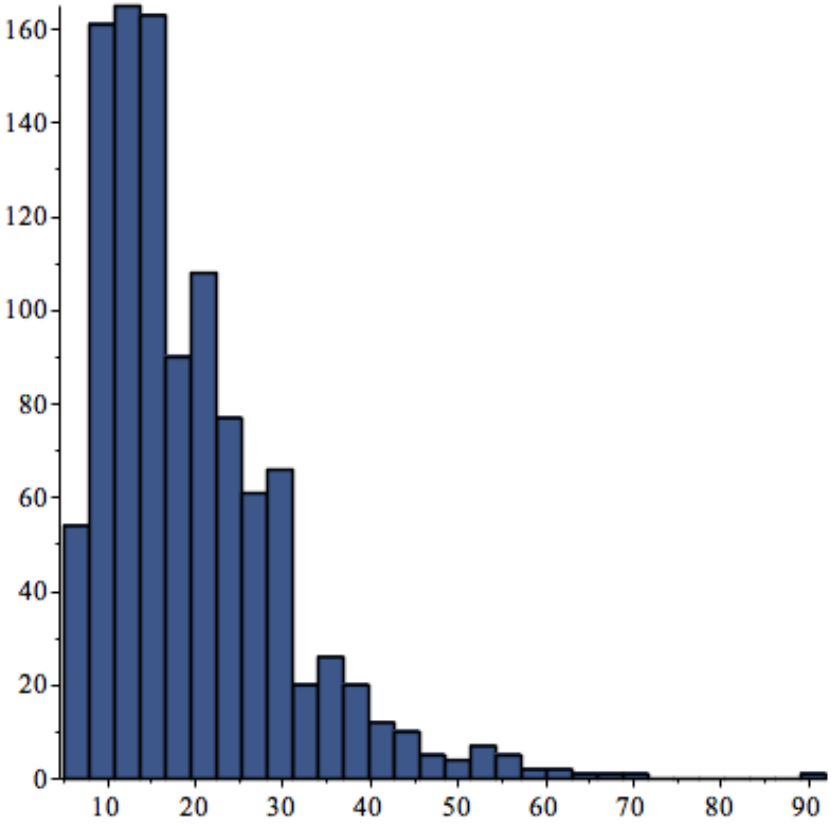


Figure A.27: `tds` of polynomials after GB.

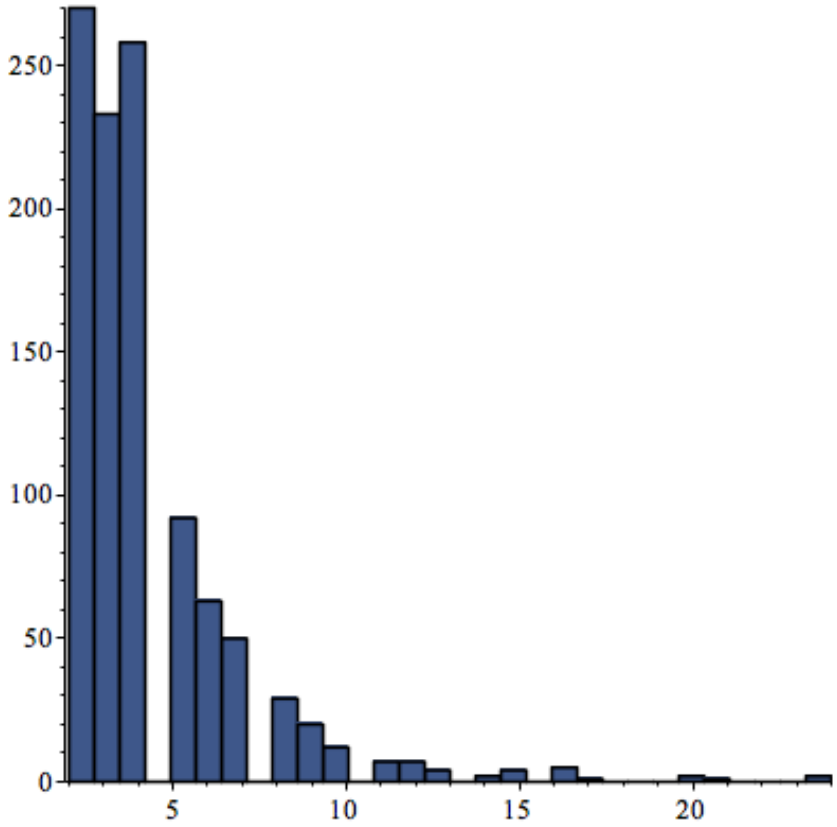


Figure A.28: Maximum degree of x among all polynomials after GB.

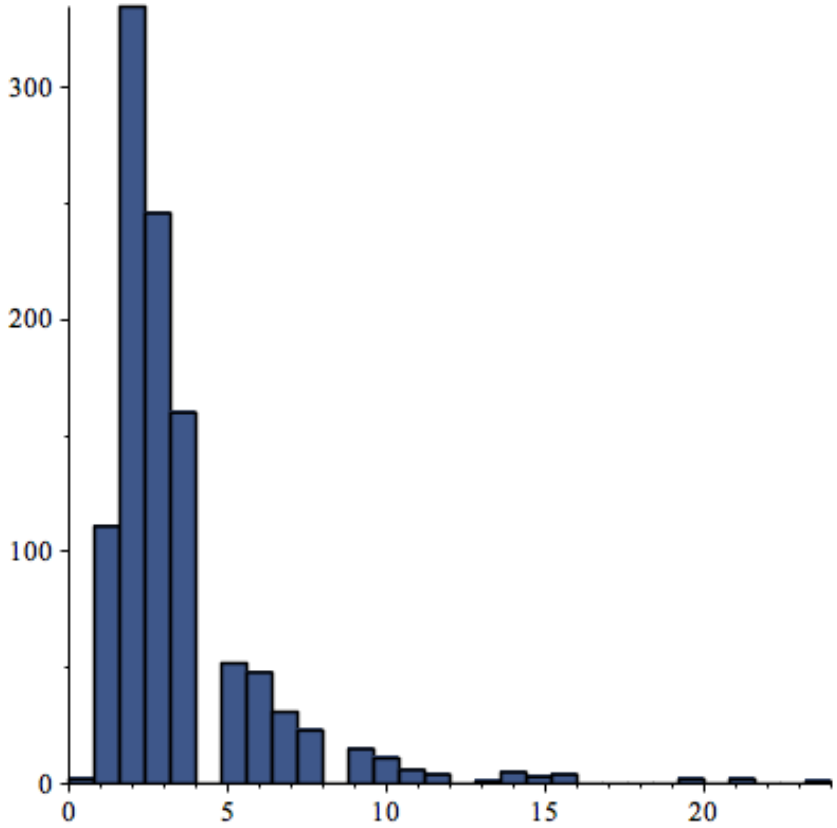


Figure A.29: Maximum degree of y among all polynomials after GB.

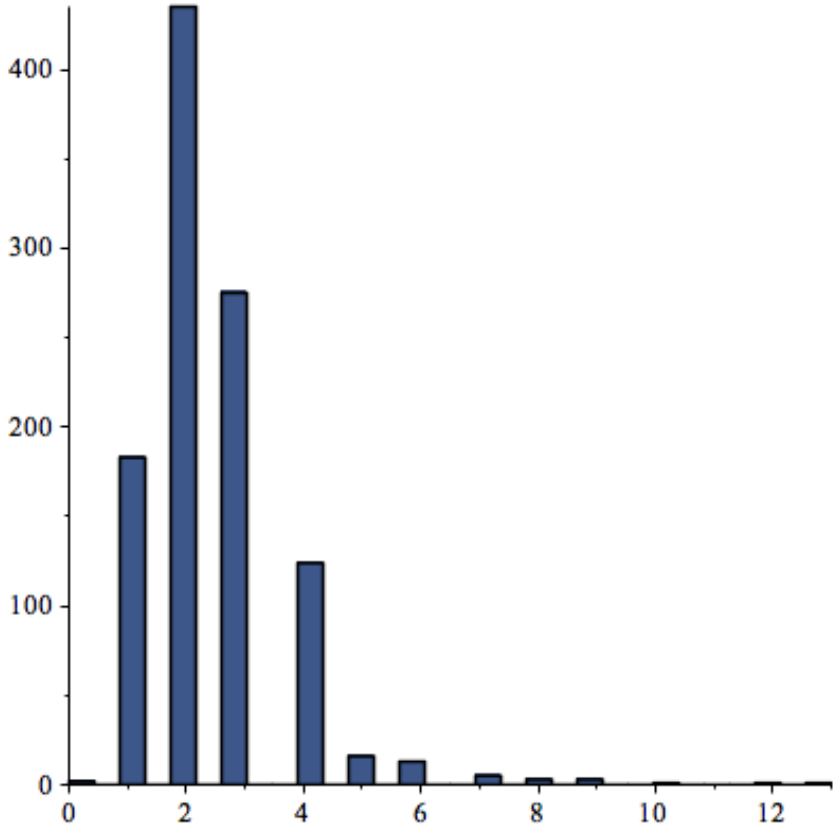


Figure A.30: Maximum degree of z among all polynomials after GB.

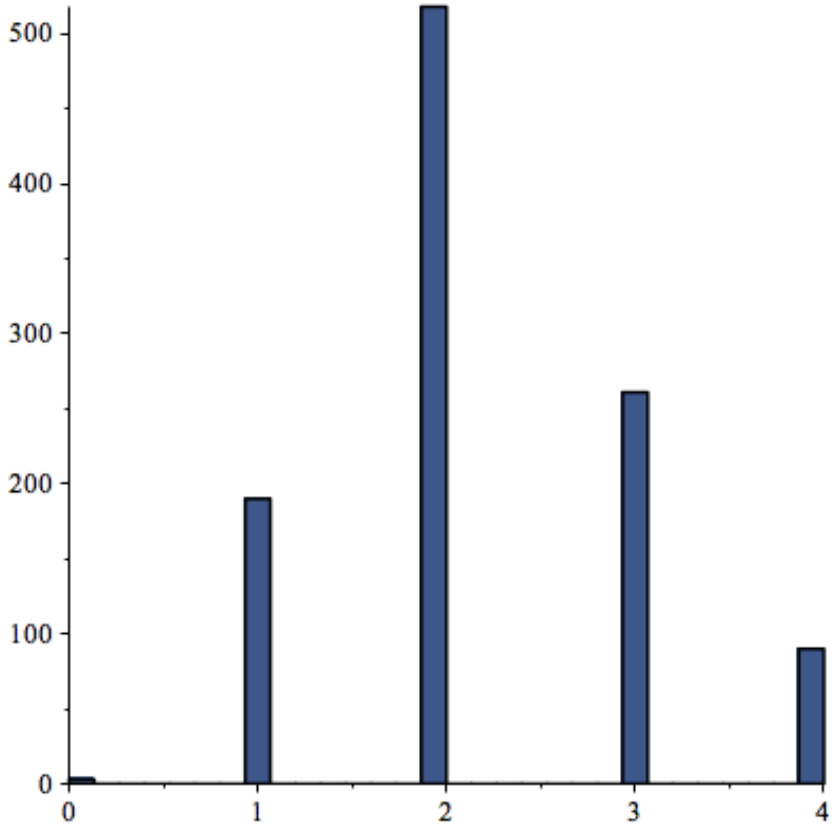


Figure A.31: Proportion of x occurring in polynomials after GB.

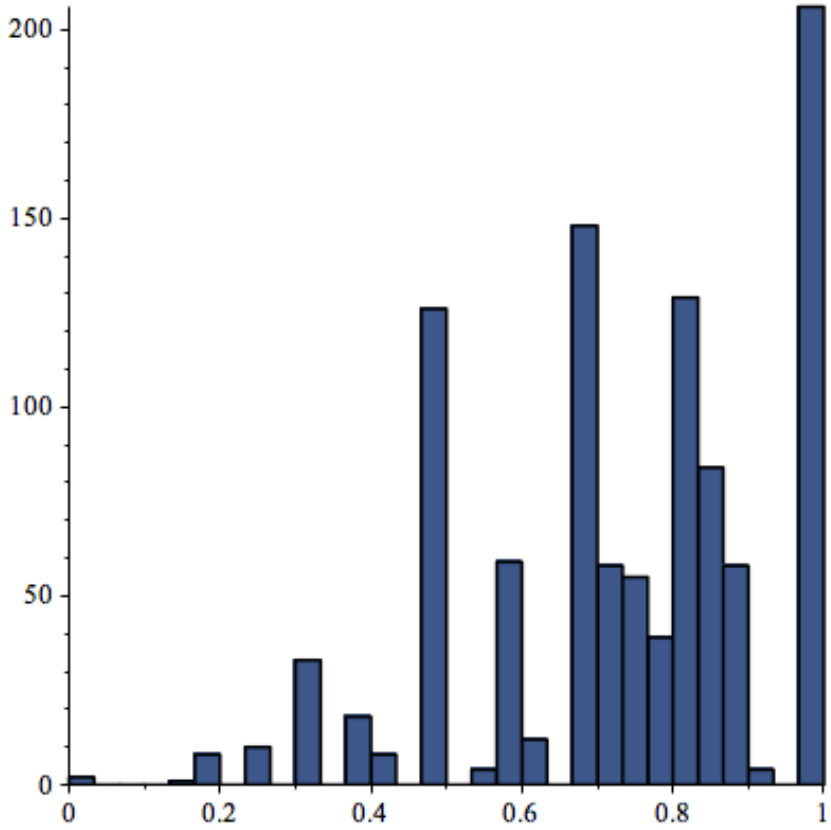


Figure A.32: Proportion of y occurring in polynomials after GB.

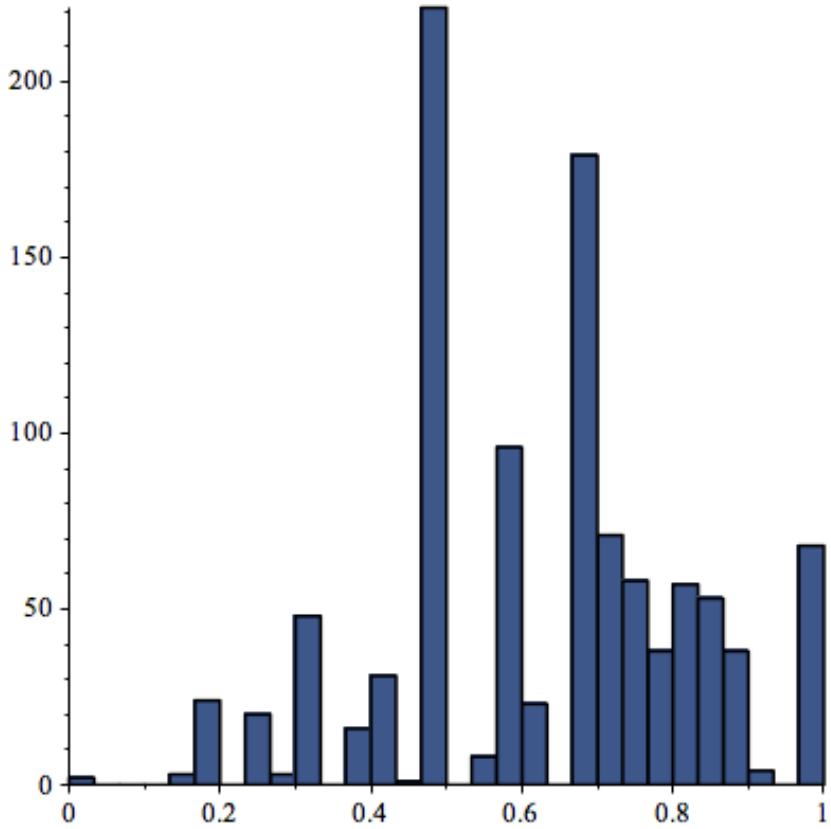


Figure A.33: Proportion of z occurring in polynomials after GB.

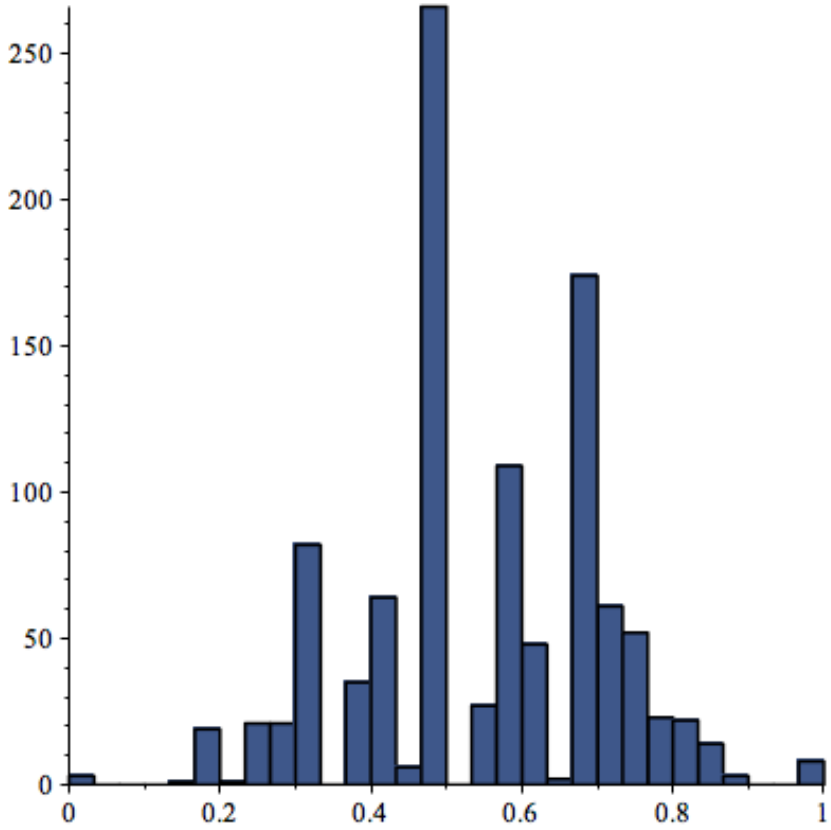


Figure A.34: Proportion of x occurring in monomials after GB.

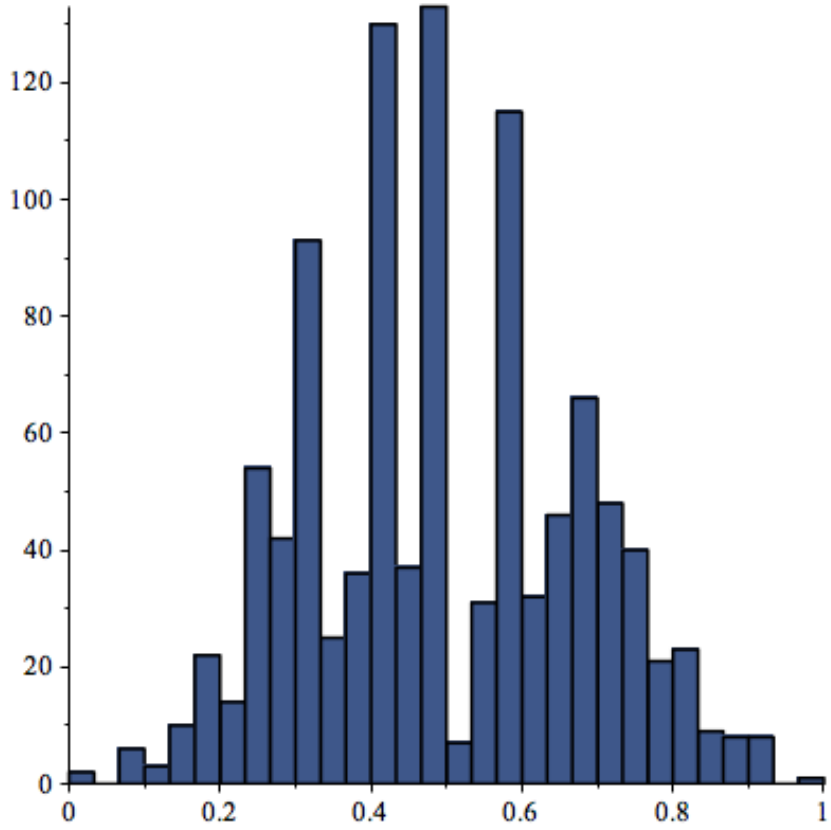


Figure A.35: Proportion of y occurring in monomials after GB.

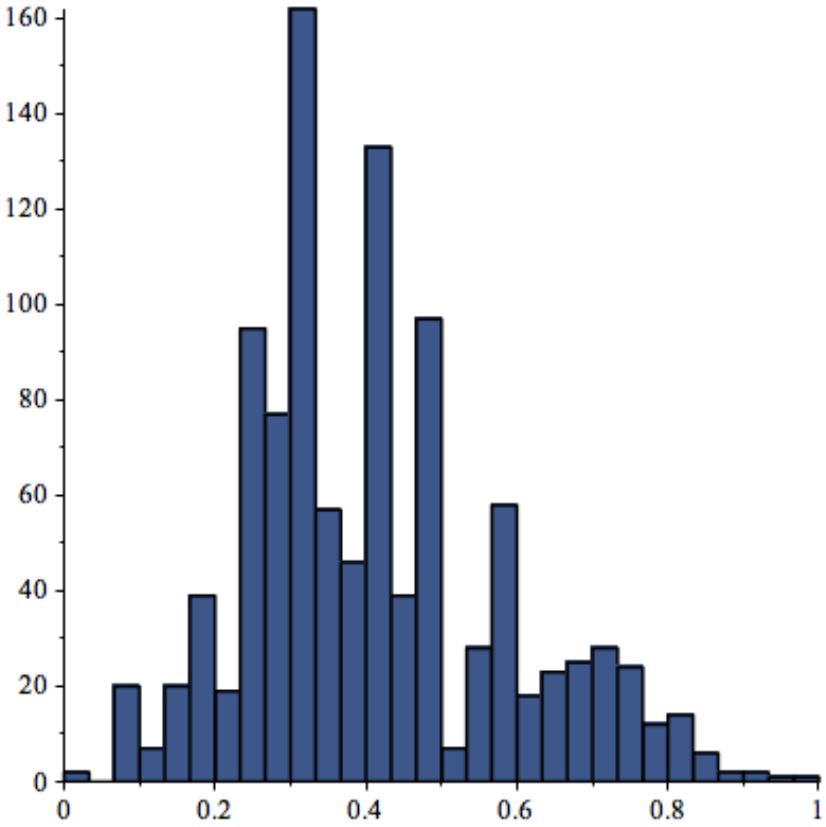


Figure A.36: Proportion of z occurring in monomials after GB.

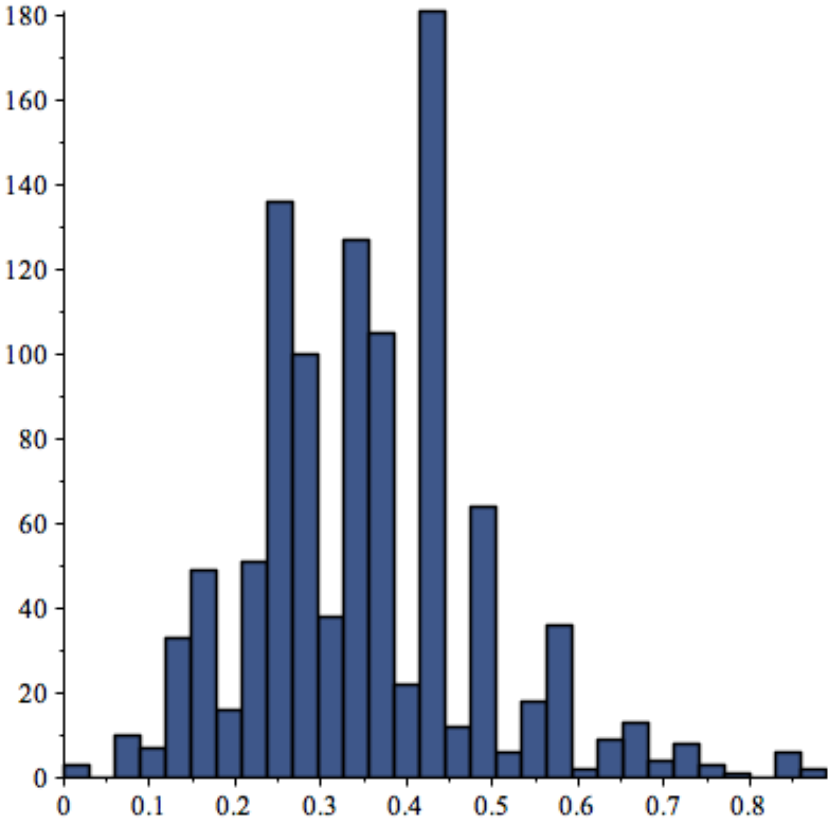


Figure A.37: lg(`TNoI` before) - lg(`TNoI` after)

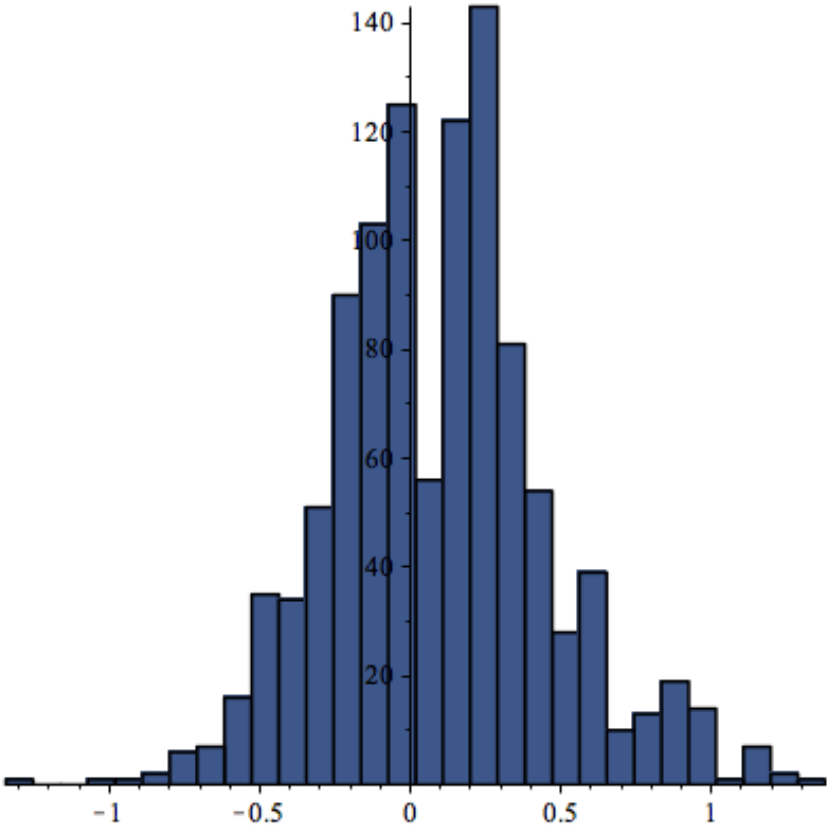


Figure A.38: lg(`stds` before) - lg(`stds` after)

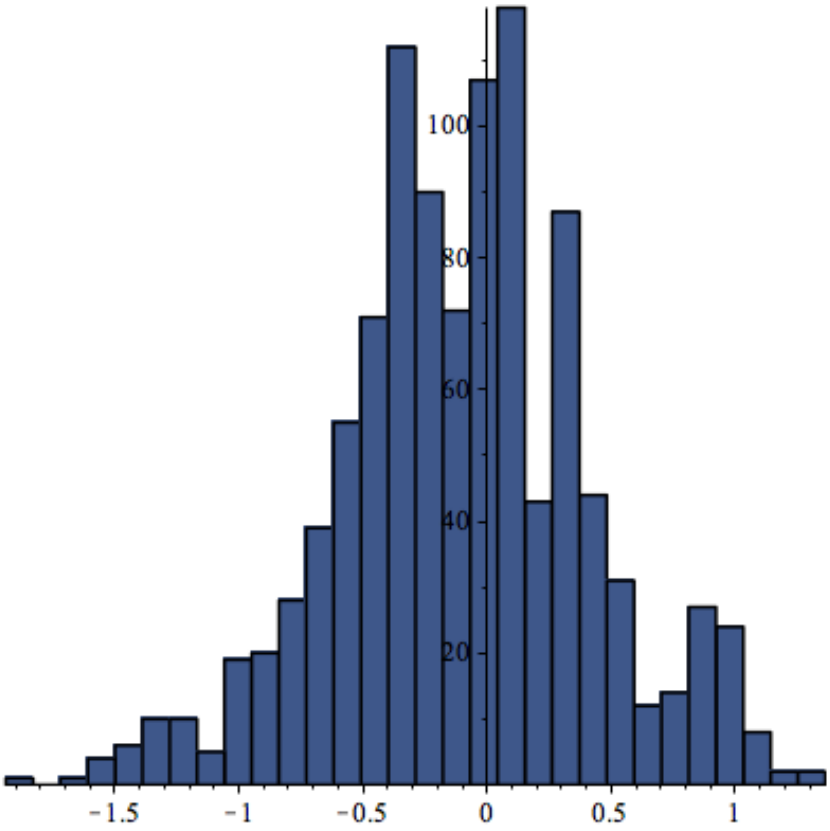


Figure A.39: lg(`tds` before) - lg(`tds` after)

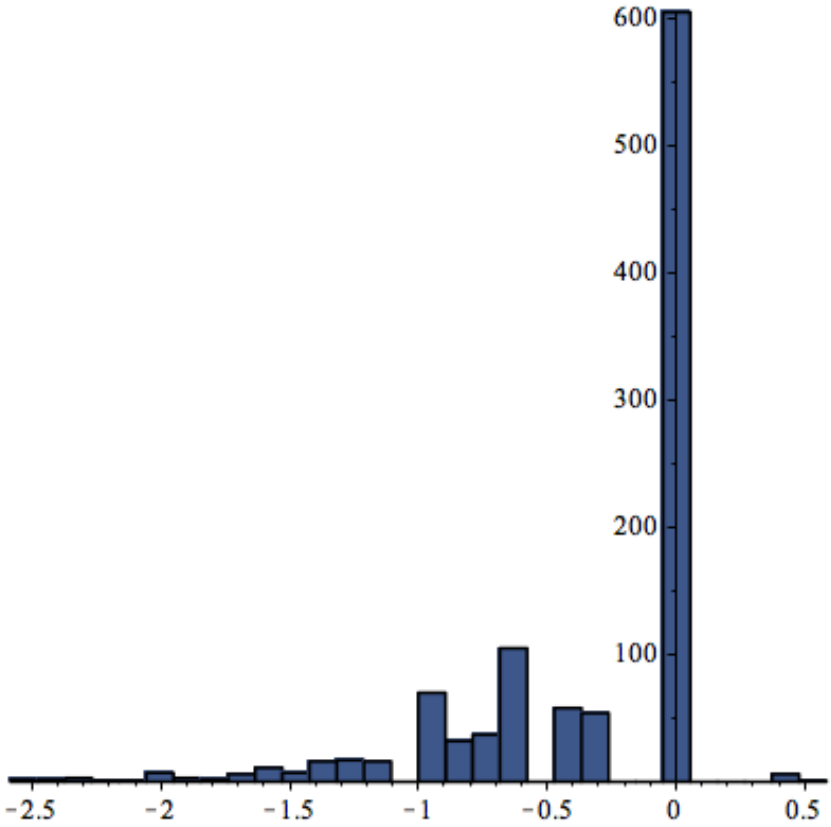


Figure A.40: Correlations between proportion of x_0 occurring in polynomials and monomials.

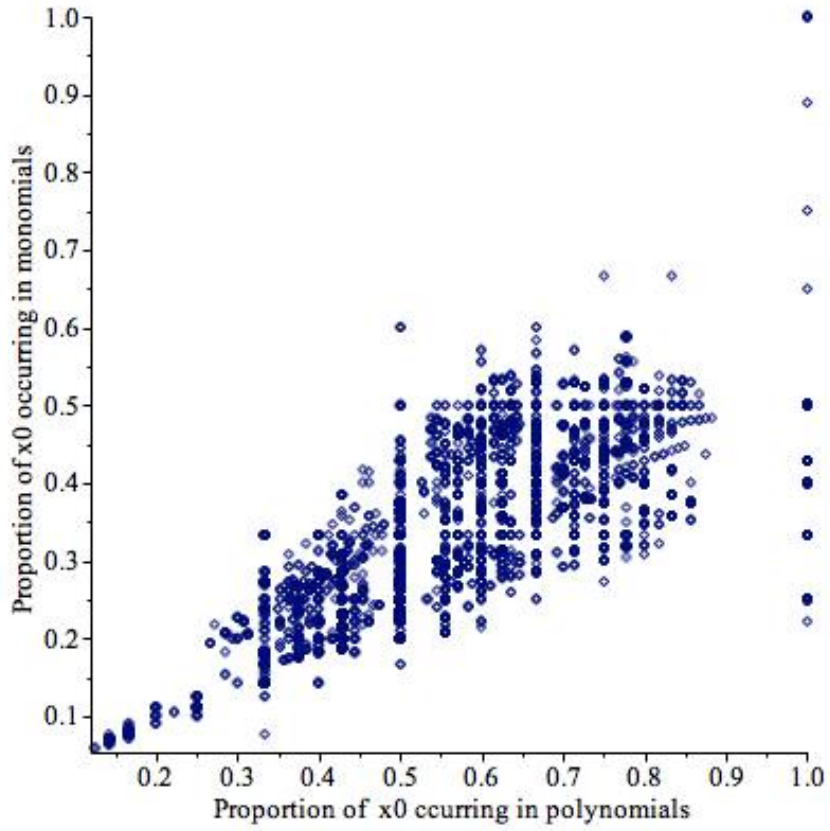


Figure A.41: Correlations between proportion of x_1 occurring in polynomials and monomials.

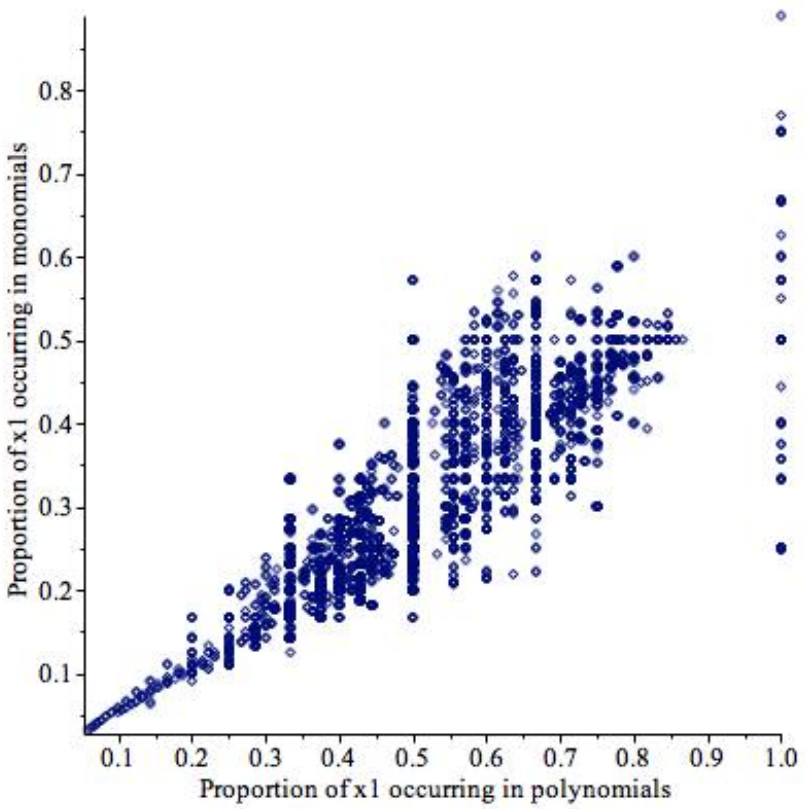


Figure A.42: Correlations between proportion of x_2 occurring in polynomials and monomials.

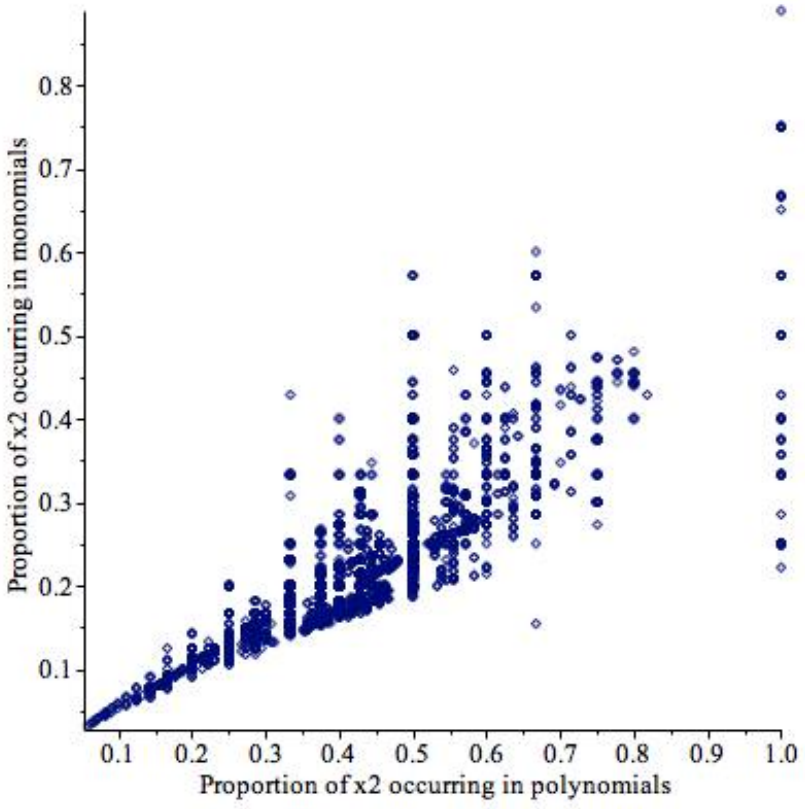


Figure A.43: Correlations between maximum degree of x_0 among all polynomials and proportion of x_0 occurring in polynomials

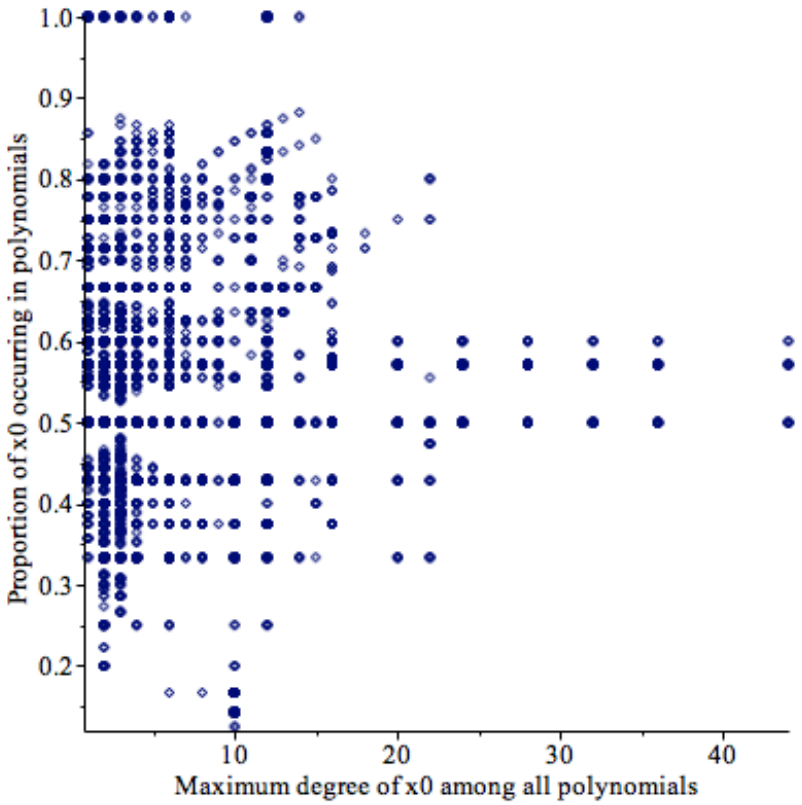


Figure A.44: Correlations between maximum degree of x_1 among all polynomials and proportion of x_1 occurring in polynomials

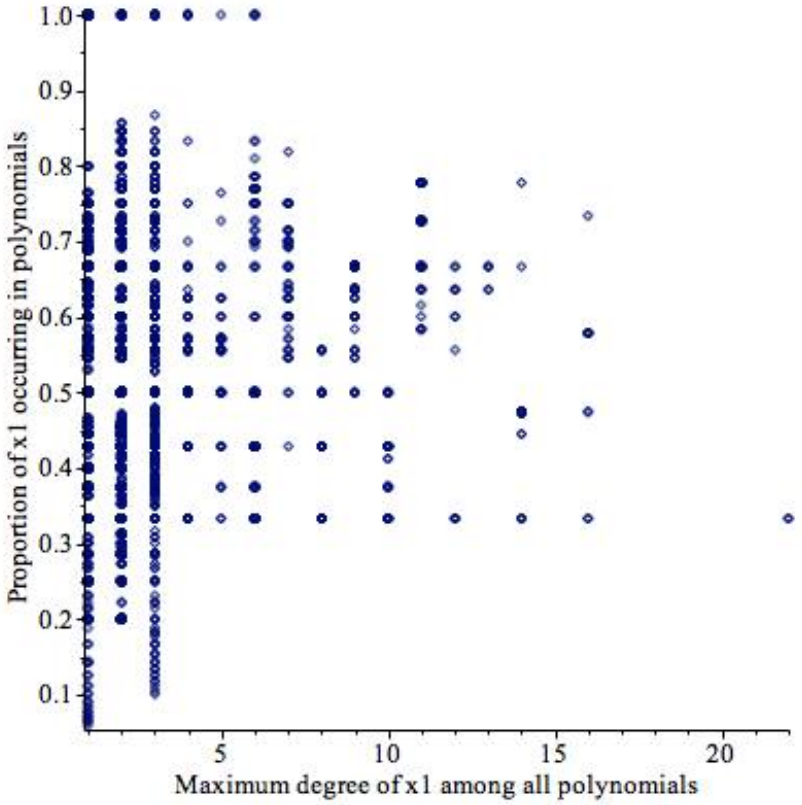


Figure A.45: Correlations between maximum degree of x_2 among all polynomials and proportion of x_2 occurring in polynomials

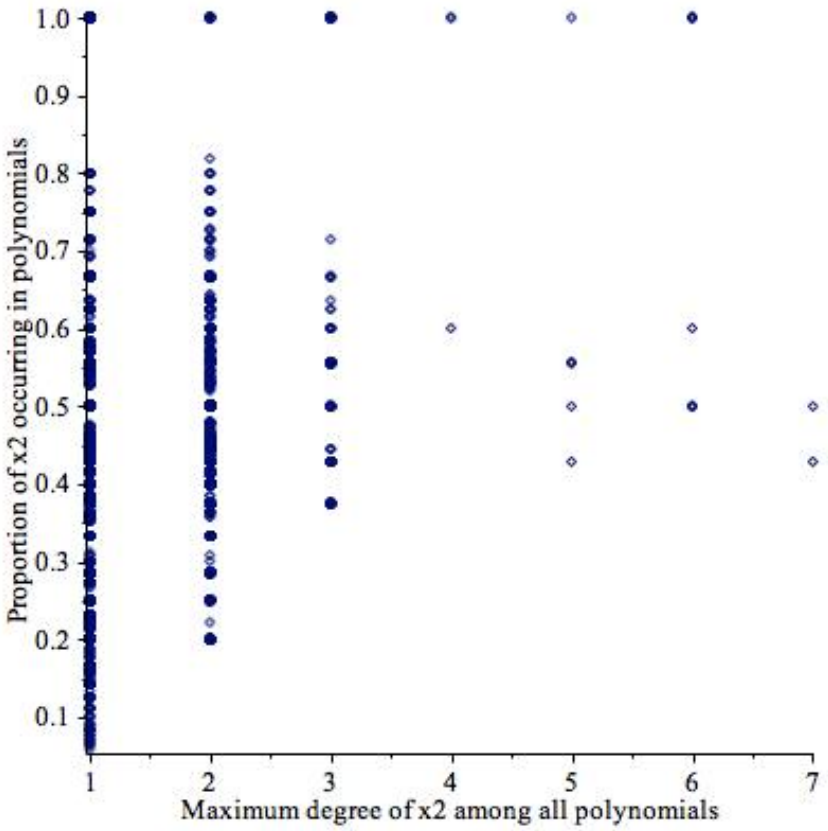


Figure A.46: Correlations between TNoI before GB and stds before GB.

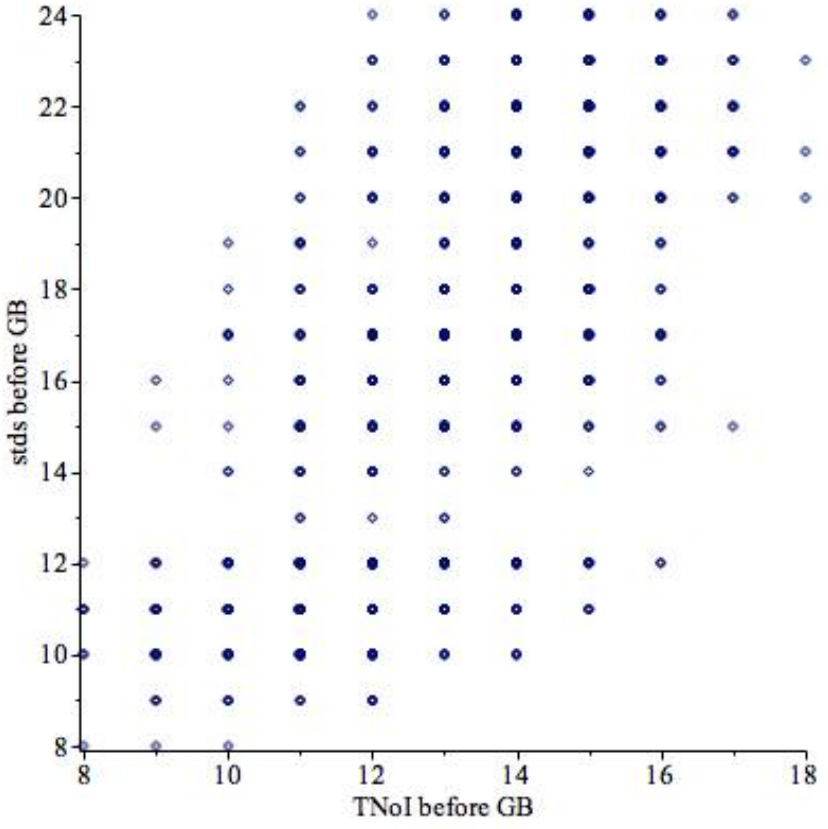


Figure A.47: Correlations between TNoI before and after GB.

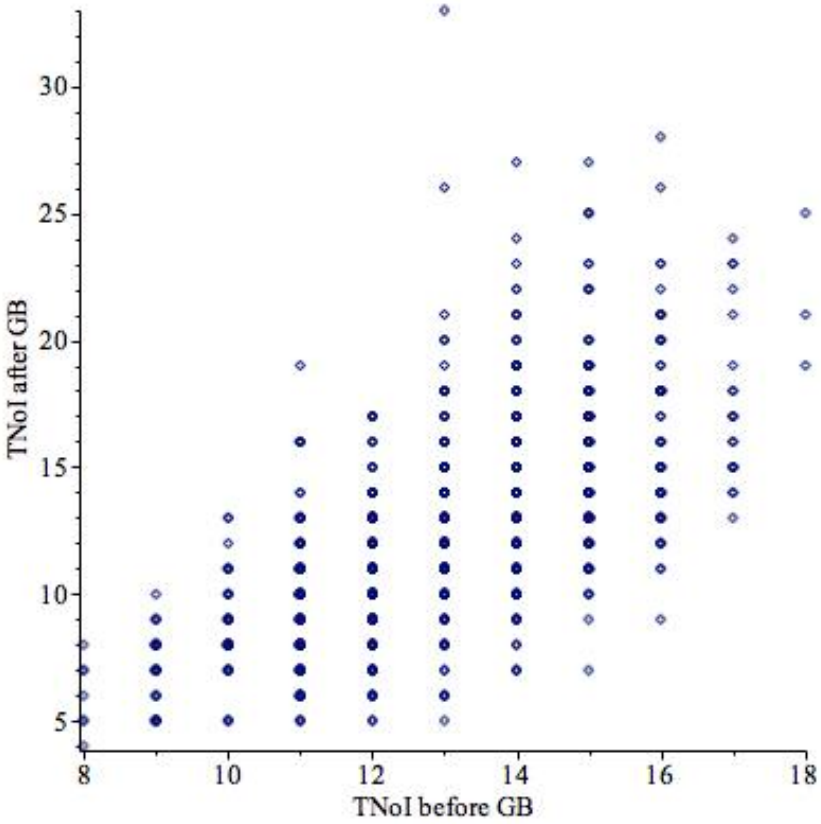


Figure A.48: Correlations between stds before and after GB.

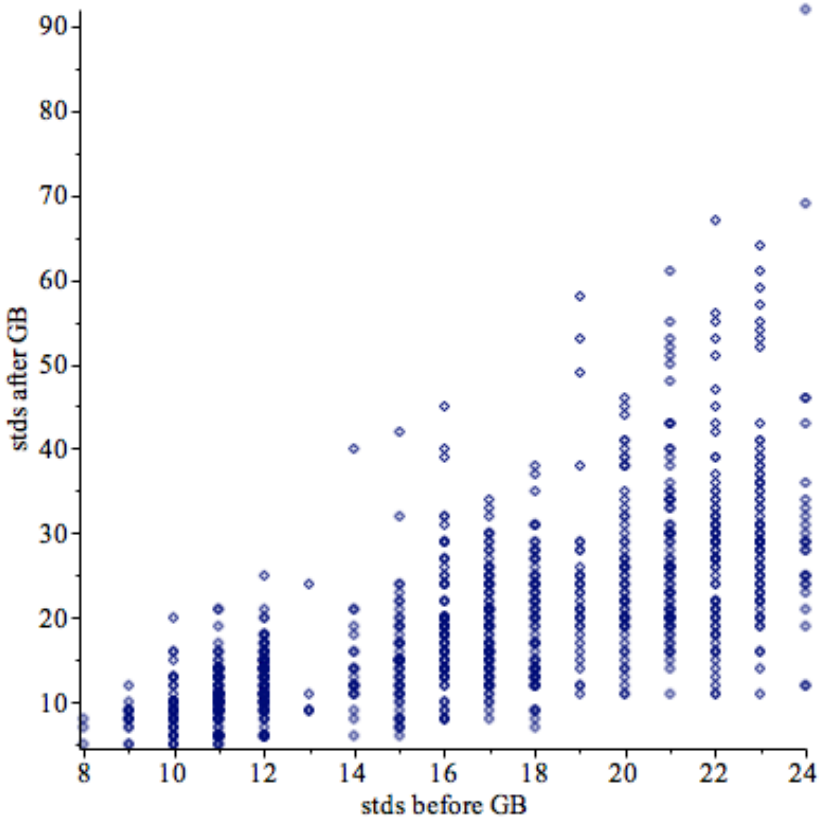


Figure A.49: Correlations between tds before and after GB.

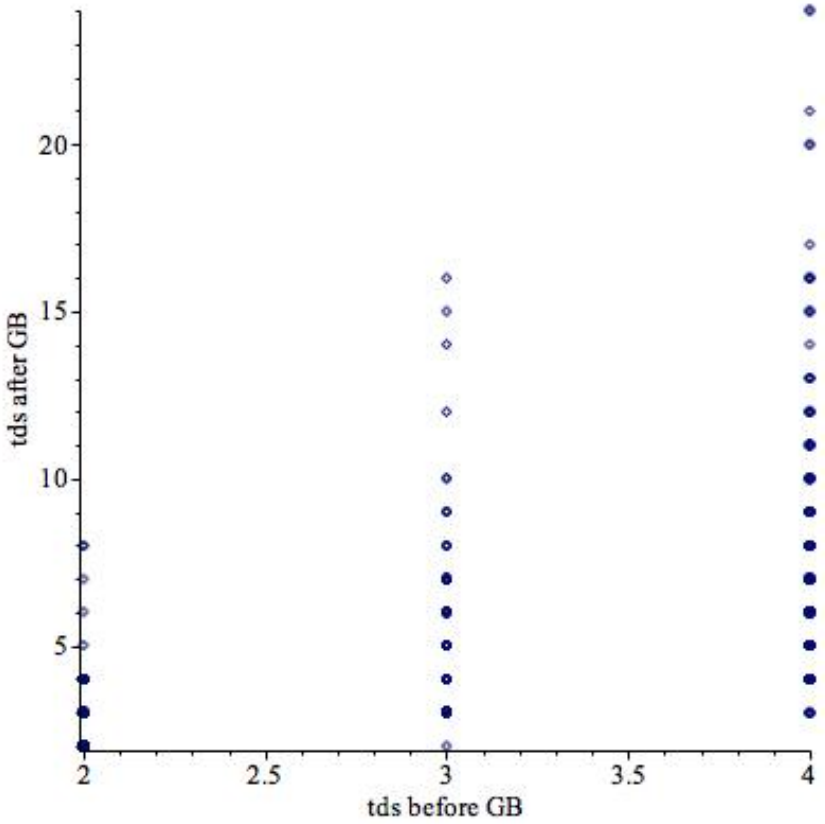


Figure A.50: Correlations between lg(TNoI) difference and TNoI before GB.

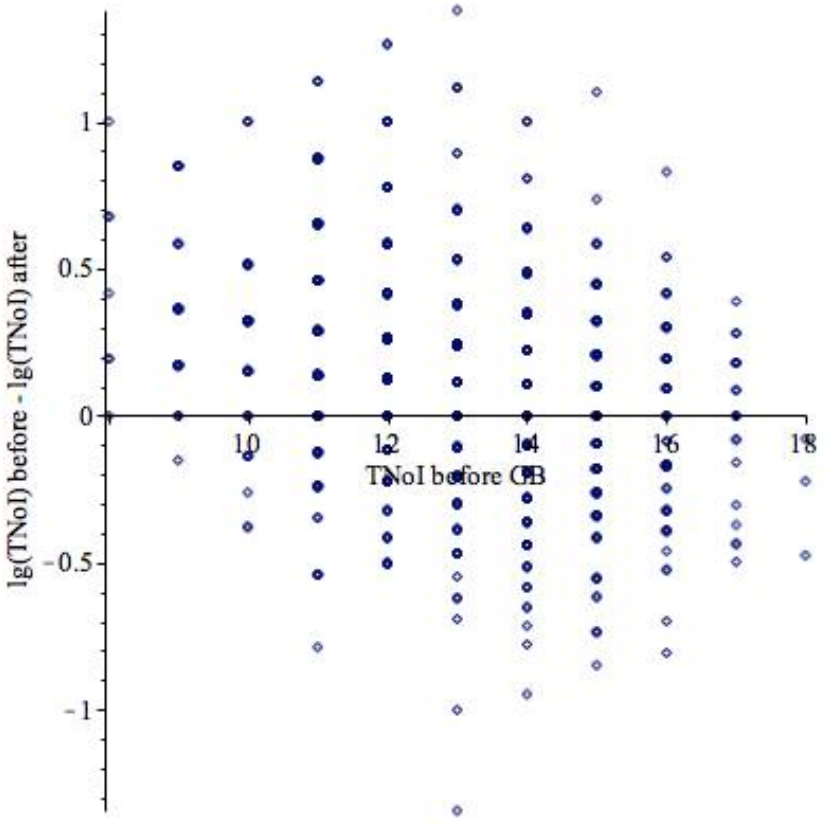


Figure A.51: Correlations between lg(stds) difference and stds before GB.

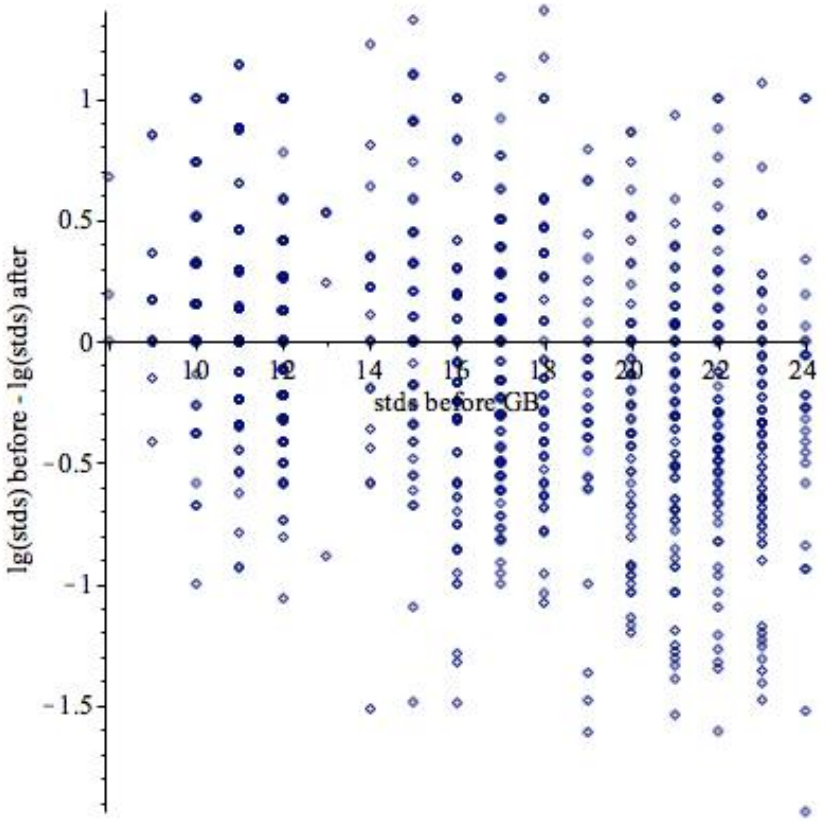
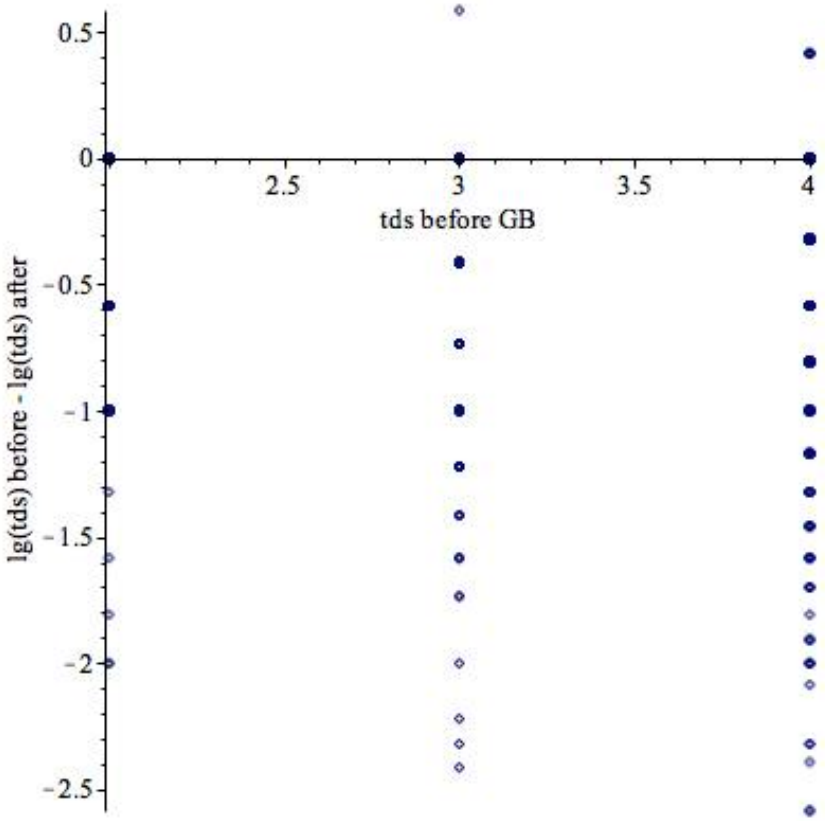


Figure A.52: Correlations between lg(tds) difference and tds before GB.



Appendix B

Varying hyper-parameters

In two of the experiments, I performed a grid-search algorithm to determine the hyper-parameters (C, γ) of the RBF kernel function used in the learning. While our grid-search returned one optimum, I further examined the neighbourhood of the optimal solution to ensure that the process was stable with respect to these parameter choices. In the following tables I display the Matthews correlation coefficients for the optimal parameter choice found by the grid optimization, as well as 8 neighbouring parameter pairs in the grid. In particular, Tables B.1 to B.6 shows the values for the different heuristics used in Chapter 4, while Table B.7 refers to learning used in Chapter 5. Since the MCC scores for the choices around the optimum are fairly close to the optimum, the process seems to be stable.

Table B.1: MCC scores of classifier for Brown heuristic with quantifier free data

	$lg\gamma = -1$	$lg\gamma = 0$	$lg\gamma = 1$
$lgC = -2$	0.435	0.453	0.467
$lgC = -1$	0.445	0.474	0.469
$lgC = 0$	0.468	0.472	0.462

Table B.2: MCC scores of classifier for Brown heuristic with quantified data

	$lg\gamma = -1$	$lg\gamma = 0$	$lg\gamma = 1$
$lgC = -2$	0.342	0.351	0.337
$lgC = -1$	0.349	0.366	0.332
$lgC = 0$	0.345	0.346	0.312

Table B.3: MCC scores of classifier for sotd heuristic with quantifier free data

	$lg\gamma = -8$	$lg\gamma = -7$	$lg\gamma = -6$
$lgC = 12$	0.387	0.409	0.38
$lgC = 13$	0.414	0.456	0.346
$lgC = 14$	0.401	0.422	0.388

Table B.4: MCC scores of classifier for sotd heuristic with quantified data

	$lg\gamma = -3$	$lg\gamma = -2$	$lg\gamma = -1$
$lgC = 1$	0.286	0.332	0.332
$lgC = 2$	0.302	0.343	0.328
$lgC = 3$	0.318	0.335	0.336

Table B.5: MCC scores of classifier for ndrr heuristic with quantifier free data

	$lg\gamma = -3$	$lg\gamma = -2$	$lg\gamma = -1$
$lgC = 0$	0.452	0.446	0.435
$lgC = 1$	0.445	0.568	0.427
$lgC = 2$	0.439	0.43	0.429

Table B.6: MCC scores of classifier for ndrr heuristic with quantified data

	$lg\gamma = -1$	$lg\gamma = 0$	$lg\gamma = 1$
$lgC = -1$	0.38	0.384	0.386
$lgC = 0$	0.38	0.403	0.391
$lgC = 1$	0.388	0.395	0.376

Table B.7: MCC scores of classifier for Gröbner basis preconditioning

	$lg\gamma = -9$	$lg\gamma = -8$	$lg\gamma = -7$
$lgC = 5$	0.5	0.511	0.521
$lgC = 6$	0.505	0.536	0.521
$lgC = 7$	0.517	0.515	0.528