Math.Comput.Sci. (2019) 13:461–488
https://doi.org/10.1007/s11786-019-00394-8

Mathematics in Computer Science

Using Machine Learning to Improve Cylindrical Algebraic Decomposition

Zongyan Huang · Matthew England · David J. Wilson · James Bridge · James H. Davenport · Lawrence C. Paulson

Received: 31 January 2017 / Revised: 7 May 2018 / Accepted: 25 February 2019 / Published online: 3 April 2019

Abstract Cylindrical Algebraic Decomposition (CAD) is a key tool in computational algebraic geometry, best known as a procedure to enable Quantifier Elimination over real-closed fields. However, it has a worst case complexity doubly exponential in the size of the input, which is often encountered in practice. It has been observed that for many problems a change in algorithm settings or problem formulation can cause huge differences in runtime costs, changing problem instances from intractable to easy. A number of heuristics have been developed to help with such choices, but the complicated nature of the geometric relationships involved means these are imperfect and can sometimes make poor choices. We investigate the use of machine learning (specifically support vector machines) to make such choices instead. Machine learning is the process of fitting a computer model to a complex function based on properties learned from measured data. In this paper we apply it in two case studies: the first to select between heuristics for choosing a CAD variable ordering; the second to identify when a CAD problem instance would benefit from Gröbner Basis preconditioning. These appear to be the first such applications of machine learning to Symbolic Computation. We demonstrate in both cases that the machine learned choice outperforms human developed heuristics.

Keywords Symbolic Computation · Computer Algebra · Machine Learning · Support Vector Machine · Cylindrical Algebraic Decomposition · Gröbner Basis · Parameter Selection

Z. Huang · J. Bridge · L. C. Paulson
Computer Laboratory, University of Cambridge, Cambridge CB3 0FD, UK
e-mail: rubyhuang87@gmail.com

J. Bridge
e-mail: james.bridge@runbox.com

L. C. Paulson
e-mail: lp15@cam.ac.uk

M. England (✉)
Faculty of Engineering, Environment and Computing, Coventry University, Coventry CV1 2JH, UK
e-mail: Matthew.England@coventry.ac.uk

D. J. Wilson · J. H. Davenport
Department of Computer Science, University of Bath, Bath BA2 7AY, UK
e-mail: J.H.Davenport@bath.ac.uk

D. J. Wilson
e-mail: david.john.wilson@me.com

Mathematics Subject Classification 68W30 (Symbolic Computation and Algebraic Computation) · 68T05 (Learning and Adaptive Systems)

1 Introduction

1.1 Quantifier Elimination

A logical statement is *quantified* if it involves the universal quantifier ∀ or the existential quantifier ∃. The *Quantifier Elimination* (QE) problem is to derive from a quantified formula an equivalent un-quantified one. A simple example would be that the quantified statement, "$\exists x$ such that $x^2 + bx + c = 0$" is equivalent to the unquantified statement "$b^2 - 4c \geq 0$", when working over the real numbers.

We can think of QE as simplifying or solving a problem. The tools involved fall within the field of *Symbolic Computation*, implemented in specialist *Computer Algebra Systems*. The exact nature of the tools mean we can draw insight from the answers: the simple example of QE above showed how the number of solutions of the quadratic depends on the discriminant. Further, many sentences which appear at first glance to not be quantified, in fact are. For example, the statement "the natural logarithm is the inverse of the exponential function" is really the statement "$\forall x$ we have $\ln(e^x) = x$". Hence solutions to the QE problem have numerous applications in logic and computational mathematics, and in turn throughout engineering and the sciences. Classical applications include motion planning in robotics [98] while recent ones include artificial intelligence to pass a university entrance exam [3], the detection of Hopf bifurcations in chemical reaction networks [51] and formal theorem proving [82].

When the logical statements are expressed as *Tarski formulae*, Boolean combinations ($\wedge$, $\vee$, $\neg$, $\rightarrow$) of statements about the signs[1] of polynomials with integer coefficients, then the QE problem is always soluble [95]. However, the only implemented general QE procedure has algorithmic complexity doubly exponential in the number of variables [40], a theoretical result that is often experienced in practice. For many problem classes QE procedures will work well at first, but as the problem size increases the doubly exponential wall is inevitably hit. It is hence of critical importance to optimise the way QE procedures are used and the formulation of problems, to "push the doubly exponential wall back" and open up a wider range of tractable applications.

1.2 Machine Learning for QE

A QE procedure can often be run in multiple ways to solve a problem: they can be initialized with different options (e.g. variable ordering [41], equational constraint designation [14]); tasks can be completed in different orders (e.g. order of constraint analysis [43]); and the problem itself may be expressible in different formalisations (e.g. the different formulations of the "piano mover's problem" [98]). Changing these settings can have a substantial effect on the computational costs (both time and memory), the tractability, or even the theoretical complexity of a problem.

Analysing these choices is a critical, and until recently understudied, problem. At the moment most QE procedures either force their users to make these choices explicitly, or make them using fairly crude human-designed heuristics which have only a limited scientific basis. The present paper describes two case studies that use machine learning techniques to make these choices instead. To the authors knowledge these are the first such applications of machine learning to QE, or Symbolic Computation more generally. One similar application has since followed by Kobayashi et al. [73] who applied machine learning to decide the order of sub-formulae solving for their QE procedure.[2] Our work followed the application of machine learning by some of the authors to theorem proving [16,68].

The two case studies both concern the Cylindrical Algebraic Decomposition (CAD) algorithm which takes a formula and produces a decomposition of real space from which a solution to a QE problem on the input can be

[1] Principally $\{= 0, > 0, < 0\}$, but the Boolean combinations allow $\{\neq 0, \geq 0, \leq 0\}$ as well.

[2] The feature set they used for their support vector machine was seeded by Table 1.

constructed. We consider the decision of variable ordering to use for CAD; and whether to precondition the input using Gröbner Basis technology. Each decision can have a large effect on the time and memory requirements of the CAD construction. We demonstrate that in each case the decision can be made using machine learning, more specifically a Support Vector Machine (SVM) with Radial Basis Function.

Machine learning, an over arching term for tools that allow computers to make decisions that are not explicitly programmed, usually involves the statistical analysis of large quantities of data. On the surface this sounds at odds with the field of Symbolic Computation which prizes exact correctness rather than inexact numerics or probabilistic tools. We are able to combine these fields because the choices we use machine learning to make are between different methods that would all produce correct and exact answers, but potentially with very different computational costs.

1.3 Paper Outline

We continue in Sect. 2 by introducing background material on CAD (Sect. 2.1); machine learning (Sect. 2.2) and the topics of our two case studies (Sects. 2.3, 2.4). In Sect. 3 we describe the methodology of our work including the software employed (Sect. 3.1); the datasets used (Sect. 3.2) and how the data will be analysed and choices made (Sect. 3.3). We then give the detailed results of the two case studies in Sects. 4 and 5 including the features used for the classification, the optimisation of the SVM parameters and the feature selection methods employed. We finish in Sect. 6 with a summary and possibilities for future work.

The two case studies have been presented in conference proceedings [66,67]. This journal article draws them together into a single work, expanding on various details, in particular: the preceding motivation, the discussion on suitable datasets in Sect. 3.2 and the possibilities for future work in Sect. 6.3.

2 Background Material

2.1 Cylindrical Algebraic Decomposition

A *Cylindrical Algebraic Decomposition* (CAD) is a *decomposition* of ordered $\mathbb{R}^n$ space into cells arranged *cylindrically*: meaning the projections of any pair of cells with respect to the given ordering are either equal or disjoint. The word algebraic is actually short for *semi-algebraic* meaning that each CAD cell can be described with a finite sequence of polynomial constraints.

A CAD is produced to be invariant for input, most commonly: *sign-invariant* for a set of polynomials (so each polynomial is either positive, zero or negative throughout each cell); or *truth-invariant* for formulae (so each formula is either true of false throughout each cell).

CADs and the first algorithm to compute them were introduced by Collins [34]. A traditional CAD performs two stages. First, *projection* calculates sets of projection polynomials S_i in variables $(x_1, \ldots, x_i)$ by applying an operator recursively to the input to derive corresponding problems in lower dimensions. Then in the *lifting* stage CADs are built incrementally by dimension according to these polynomials. First, the real line is decomposed according to the roots of the univariate polynomials 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. The theory of the projection operators allows working at a sample point to be sufficient to draw conclusions for the entire cell. Taking the union of these then gives the decomposition of $\mathbb{R}^2$ and we proceed this way to a decomposition of $\mathbb{R}^n$. For further details on the original CAD algorithm see the 1984 work of Arnon et al. [4].

CAD has worst case complexity doubly exponential in the number of variables [40] applicable due to the nature of the data structure rather than the method of computation [21]. For some applications there do exist algorithms with better complexity [6], but CAD implementations still remain the best general purpose approach for many. This is due to the extensive research to improve the efficiency of CAD since Collins' original work. Prominent advances include:

- improvements to the projection operator [17,60,62,76,78];
- only constructing cells when necessary (e.g. partial CAD [36] and sub-CAD [97]);
- symbolic–numeric lifting schemes [69,93].

Despite it now being over 40 years since Collins' original algorithm CAD is still an active area of research with recent advances including:

- making use of any Boolean structure in the input [12,13,44];
- decompositions via complex space [10,30,33];
- local projection approaches [23,94];
- non-uniform CADs (which relax the global cylindricity condition) [20]; and their interaction with (Satisfiability Module Theory) SMT-solvers [1,72].

Via its use in QE, CAD has many application ranging from epidemic modelling [22] to formal verification [83]. CAD also has independent applications: such as reasoning with multi-valued functions [38] (where we identify the branch cuts of the single valued functions implemented in computer algebra systems [45], decompose space according to these, and check the validity of proposed simplifications from the multi-valued theory); and generating semi-algebraic descriptions of regions in parameter space where multi-stationarity can occur for biological networks [11,48]. New applications are being found all the time, such as structural design (minimizing the weight of trusses) [29]; the derivation of optimal numerical schemes [50]; and the validation of economic hypotheses [80].

2.2 Machine Learning

Machine learning deals with the design of programs that can learn rules from data. This is an attractive alternative to manually constructing such rules when the underlying functional relationship is very complex, as appears to the be the case for the decisions to be made in our two case studies. Machine learning techniques have been widely used in a great many fields, such as web searching [9], text categorization [89], robotics [92], expert systems [54].

Machine learning has a long history. We reference only a few relevant highlights here and refer the reader to, for example, the textbook of Alpaydin [2] for more details. Seminal contributions include: the first computational model for *neural networks* of McCulloch and Pitts [79]; the *perceptron* proposed by Rosenblatt as an iterative algorithm for supervised classification of an input into one of several possible non-binary outputs [87]; *decision tree learning* approaches [81], which apply serial classifications to refine the output state; the *multi-layer perceptron* [64] (a modification that can distinguish data that are non-linearly separable), and the original *support vector machine* algorithm [96], a development of the perception approach.

Support Vector Machines (SVMs) are the machine learning tools used for our case studies. The standard SVM classifier takes a set of input data and predicts one of two possible classes from the input, making them a non-probabilistic binary classifier. An SVM model represents the examples as points in space, mapped so that the points of the categories are divided by a clear gap, as wide as possible. The original work [96] used linear classification, but SVMs can also efficiently perform a non-linear classification by mapping their inputs into high-dimensional feature spaces using a kernel function [8,37].

SVMs today give a powerful and robust method for both classification and regression. *Classification* refers to the assignment of input examples into a given set of classes (the output being the class labels). *Regression* refers to a supervised pattern analysis in which the output is real-valued. Given a set of examples, each marked as belonging to one of the two classes, an SVM training algorithm builds a model that assigns new examples into one of the classes. Modern SVM technology can deal efficiently with high-dimensional data, and is flexible in modelling diverse sources of data.

As noted above, a modern SVM will typically use a kernel function to map data into a high dimensional feature space. Kernel functions enable operations in feature space without ever computing the coordinates of the data in that space. Instead they simply compute the inner products between all pairs of data vectors. This operation is generally

computationally cheaper than the explicit computation of the coordinates. We refer the reader to the textbook of Shawe-Taylor and Cristianini [91] for further details on kernel functions.

The success of SVMs has led to a rapid spread of the use of machine learning. For example the survey paper of Byun and Lee [26] identifies their use in face recognition, object detection, handwriting recognition, image retreieval and speech recognition; while the textbook of Schölkopf, Tsuda and Vert details applications in computational biology [88]. This paper describes two case studies which are the first applications (to the best of the authors' knowledge) in computer algebra software.

2.3 Case Study A topic: CAD variable ordering

A CAD is defined with respect to a variable ordering (the cylindricity condition is that the projections of any two cells onto a smaller coordinate space *with respect to the ordering* is either equal or disjoint). The variable ordering dictates the order in which variables are eliminated during the projection phase and the order of subspaces in which CADs are produced during lifting. For example, a variable order of $z \succ y \succ x$ would mean that cells in the CAD of (x, y, z)-space should have equal or disjoint projections when projected onto (x, y)-space or x-space. A CAD algorithm would eliminate z first and then y during projection; and during lifting would first decompose the x-axis and then the (x, y)-plane.

Depending on the application requirements the variable ordering may be determined, constrained, or entirely free (as with the application to branch cut analysis of multi-valued functions described above). The most common application, QE, requires that the variables be eliminated in the order in which they are quantified in the formula but makes no requirement on the free variables. For example, we could eliminate the quantifier in

$$\exists x \quad ax^2 + bx + c = 0$$

using any CAD which eliminates x first. The other variables could be studied in any order and so there are six possibilities we can choose from when running the CAD algorithm. Our own CAD implementation builds a CAD for the polynomial under ordering $a \prec b \prec c$ with only 27 cells, but uses 115 for the reverse ordering. Also, note that since we can switch the order of quantified variables in a statement when the quantifier is the same, we also have some choice on the ordering of quantified variables. For example, a QE problem of the form $\exists x \exists y \forall a\, \phi(x, y, a)$ could be solved by a CAD under either ordering $x \succ y \succ a$ or ordering $y \succ x \succ a$.

It has long been documented that this choice can dramatically affect the feasibility of a problem. In fact, Brown and Davenport presented a class of problems in which one variable ordering gave output of double exponential complexity in the number of variables and another output of a constant size [21]. Heuristics have been developed to help with this choice, with Dolzmann et al. [41] giving the best known study. After analysing a variety of metrics they proposed a polynomial degree based heuristic (the heuristic sotd defined below). However, in CICM 2013 some of the present authors demonstrated examples for which that heuristic could be misled [14]. We provided an alternative (the heuristic ndrr below) which addressed the intricacies of those examples, but we saw for other examples that ndrr had its own shortcomings.

Our thesis became that the best heuristic to use is dependent upon the problem considered. However, the relationship between the problems and heuristics is far from obvious. Hence we investigate whether machine learning can help with these choices, both for CAD itself and QE by CAD.

We identified the following three heuristics for picking a CAD variable ordering in the literature:

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 of 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 labelled after Brown who documented it in the notes to an ISSAC tutorial [19].

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 the outcome of the study by Dolzmann, Seidl, and Sturm which found it to be a good heuristic for both CAD and QE by CAD [41].

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 shown by Bradford et al. to assist with examples where sotd failed [14].

Brown's heuristic has the advantage of being very cheap, since it acts only on the input and checks only simple properties. The ndrr heuristic is the most expensive (requiring real root isolation), but is the only one to explicitly consider the real geometry of the problem, rather than the geometry in complex space measured by the degrees.

All three heuristics may identify more than one variable ordering as a suitable choice. In this case we take the heuristic's choice to be the first of these after they had been ordered lexicographically. This final choice will depend on the convention used for displaying the variable ordering. For example, the software QEPCAD and the notes where Brown introduces his heuristic [19] use the convention of ordering variables from left to right so that the last one is projected first; while the algorithms in MAPLE and the papers introducing sotd and ndrr [14,41] use the opposite convention. For this case study the heuristics were implemented in MAPLE and so ties were broken by picking the first lexicographically on the second convention. This corresponds to picking the first under a reverse lexicographical order under the QEPCAD convention. The important point here is that all three heuristics had ties broken under the same convention and so were treated fairly.

2.4 Case Study B topic: Preconditioning CAD with a Gröbner Basis

A *Gröbner Basis* G is a particular generating set of an ideal I defined with respect to a monomial ordering. One definition is that the ideal generated by the leading terms of I is generated by the leading terms of G. Gröbner Bases (GB[3]) allow for the calculation of properties of the ideal such as dimension and number of zeros. They are one of the main practical tools for working with polynomial systems. The GB definition, their properties and the first algorithm to derive one was introduced by Buchberger in his Ph.D. thesis of 1965 (republished in English as [24]).

Like CAD, there has been much research to improve GB calculation, with the F_5 algorithm [52] probably the most used approach today. The calculation of a GB is, also like CAD, necessarily doubly exponential in the worst case [75] (at least when using a lexicographic monomial ordering). Despite this, the computation of GB can often be done very quickly and is usually a superior tool to CAD for a problem involving only polynomial equalities.

From this arises the natural question: is the process of replacing a conjunction of polynomial equalities in a CAD problem by their GB a useful precondition for CAD? I.e. let $E = \{e_1, e_2, \dots\}$ be a set of polynomials; $G = \{g_1, g_2, \dots\}$ be a GB for E; and B be any Boolean combination of constraints, $f_i\ \sigma_i\ 0$, where $\sigma_i \in \{<, >, \leq, \geq, \neq, =\}$) and $F = \{f_1, f_2, \dots\}$ is another set of polynomials. Then the two formulae

$$\Phi = (e_1 = 0 \wedge e_2 = 0 \wedge \dots) \wedge B \text{ and}$$
$$\Psi = (g_1 = 0 \wedge g_2 = 0 \wedge \dots) \wedge B$$

are equivalent and a CAD truth-invariant for either could be used to solve problems involving Φ (such as eliminating any quantifiers applied to Φ). So is it worth producing G before calculating the CAD?

The first attempt to answer this question was given by Buchberger and Hong [25] who experimented with then recent implementations of GB [7] and CAD [36] (both in C on top of the SAC- 2 system [35]). Of the ten test

[3] We use the abbreviation GB for both Gröbner Basis and Gröbner Bases as grammar requires.

problems they studied: 6 were improved by the GB preconditioning, with the speed-up varying from twofold to 1700-fold; 1 problem resulted in a tenfold slow-down; 1 timed out when GB preconditioning was applied, while it would complete without it; and the other 2 were intractable both for CAD alone and the GB preconditioning step.

The problem was revisited 20 years later by Wilson et al. [101]. The authors recreated the experiments of Buchberger and Hong [25] using QEPCAD- B [18] for the CAD and MAPLE- 16 for the GB. As we would expect, there had been a big decrease in the computation timings, especially the GB: the two test problems previously intractable [25] could now have their GB calculated quickly. However, two of the CAD problems were still hindered by GB preconditioning. The experiments were extended to: a wider example set (an additional 12 problems); the alternative CAD implementation in MAPLE- 16 [33]; and the case where we further precondition by reducing inequalities of the system (the set F above) with respect to the GB. The key conclusion remained that GB preconditioning would in general benefit CAD (sometimes significantly) but could on occasion hinder it (to the point of making a tractable CAD problem intractable).

The authors of [101] also defined a metric to assist with the decision of when to precondition, the *Total Number of Indeterminates* (TNoI) of a set of polynomials A:

$$\texttt{TNoI}(A) = \sum_{a \in A} \texttt{NoI}(a) \tag{1}$$

where $\texttt{NoI}(a)$ is the number of indeterminates[4] in a polynomial a. Then their heuristic was to build a CAD for the preconditioned polynomials only if the TNoI decreased following preconditioning. For most test problems in the study the heuristic made the correct choice, but there were examples to the contrary, and little correlation between the change in TNoI and level of speed-up/slow-down.

For Case Study B we consider if machine learning can be applied to the decision of whether preconditioning CAD input with GB is beneficial for a particular problem. We work on the reasonable assumption that GB computation is cheap for the problems on which CAD is tractable (in fact CAD must compute resultants which overestimate the GB [47]). Hence we allow the classifier to use algebraic features of both the input problem and the GB itself to decide *whether we want to use* the GB.

3 Methodology

3.1 Software

3.1.1 Computer Algebra Software

For each case study we focussed on a single CAD implementation.

Case Study A used QEPCAD [18]: an interactive command line program written in C whose name stands for **Q**uantifier **E**limination with **P**artial **CAD**. It is a competitive implementation of both CAD and QE that allows the user a good deal of control and information during its execution. We used QEPCAD with its default settings which implement McCallum's projection operator [76] and partial CAD [36]. It can also makes use of an equational constraint automatically (via the projection operator [77]) when one is explicit in the formula, (where *explicit* means the formula is a conjunction of the equational constraint with a sub-formula).

Case Study B used the CAD algorithm in MAPLE- 17 that implements the Regular Chains based algorithm [33]. This is part of the RegularChains Library[5] [31,32] whose CAD procedures differ from the traditional projection and lifting framework of Collins. They instead first decompose $\mathbb{C}^n$ cylindrically and then refine to a CAD of $\mathbb{R}^n$.

[4] In the context of the present paper indeterminates is synonymous with variables. For the underlying application there may be a distinction between variables and parameters of the problem: but CAD, and correspondingly these metrics, would treat both the same.

[5] http://www.regularchains.org.

Previous experiments showed this implementation has the same issues of GB preconditioning as the traditional approach [101].

All other computer algebra computations were done using tools in MAPLE and took minimal computation time:

- The variable ordering heuristics for Case Study A were calculated using tools in the authors `ProjectionCAD` library [49].
- The GB calculations for Case Study B used the implementation that ships with MAPLE: a meta algorithm calling multiple GB implementations. The GBs were computed with a purely lexicographical ordering of monomials based on the same variable ordering as the CAD.
- The calculation of the algebraic features for both machine learning experiments was done using a mixture of built-in MAPLE commands and the `ProjectionCAD` library.

3.1.2 Machine Learning Software

Both case studies use the package SVM- LIGHT[6] [70] for the machine learning computations. This software is an implementation of SVMs in C which consists of two programs: SVM LEARN which fits the model parameters based on the training data and user inputs (such as the kernel function and the parameter values); and SVM CLASSIFY which uses the generated model to classify new samples.

SVM- LIGHT calculates a hyperplane of the n-dimensional transformed feature space, which is an affine subspace of dimension $n-1$ dividing the space into two, corresponding to the two distinct classes. SVM- CLASSIFY outputs margin values which are a measure of how far the sample is from this separating hyperplane. Hence the margins are a measure of the confidence in a correct prediction. A large margin represents high confidence in a correct prediction. The accuracy of the generated model is largely dependent on the selection of the kernel functions and parameter values.

3.2 Datasets

3.2.1 Historic Issues

Despite its long history and significant software contributions the Computer Algebra community had a lack of substantial datasets in use, a significant barrier to machine learning.

There have been attempts to address this problem, such as the 1990s projects PoSSo and FRISCO for polynomials systems and symbolic–numeric problems respectively. PoSSO collected a series of benchmark examples for GB, and a descendant of these can still be found online.[7] However, this does not appear to be maintained and the polynomials are not stored in a machine-readable form. Polynomials from this list still turn up in various papers, but there is no systematic reference, and it is not clear whether people are really referring to the same example (several of the examples are families which means that a benchmark has to contain specific instances). The PoSSo Project aimed for "level playing field" comparisons, but at the time different implementations ran on different hardware/operating systems meaning this was not really achievable.

The topic of benchmarking in computer algebra has most recently been taken up by the SymbolicData Project[8] [55] which is beginning to build a database of examples in XML format (although currently not with any suitable for CAD). The software described in [61] was built to translate problems in that database into executable code for various computer algebra systems. The authors of [61] discuss the peculiarities of computer algebra that make benchmarking particularly difficult including the fact that results of computations need not be unique and that the evaluation of the correctness of an output may not be trivial (or may be the subject of research itself).

[6] http://svmlight.joachims.org.

[7] http://www-sop.inria.fr/saga/POL/.

[8] http://wiki.symbolicdata.org.

For CAD there are a number of "classic" example that reappear throughout the literature, such as those from [25,41]. Some of the present authors made an effort to collect these as a freely available online resource[9] [100] which contained references to the literature and encodings for various computer algebra systems.

3.2.2 Dataset for Case Study A

The resource discussed above [100] did not have a sufficient number of CAD problems to run a machine learning experiment. Hence we decided to use instead the `nlsat` dataset[10] produced to evaluate the work in [72]. The main sources of these examples are: MetiTarski [82], an automatic theorem prover for theorems involving real-valued special functions (it applies real polynomial bounds and then using real QE tools like CAD); problems originating from attempts to prove termination of term-rewrite systems; verification conditions from Keymaera [84]; and parametrized generalizations of the problems from [63].

The problems from the `nlsat` dataset are all fully existential (the only quantifier is $\exists$) making them satisfiability or SAT problems. They could thus be tackled by the new generation of SMT-solvers [72]. Note that our decision to use only SAT problems was based solely on availability of data rather than it being a requirement of our technology, and the conclusions drawn are likely to be applicable outside of the SAT context.

We extracted 7001 three-variable CAD problems for the experiment. The number of variables was restricted for two reasons. First to make it feasible to test all possible variable orderings and second to avoid the possibility that QEPCAD will produce errors or warnings related to well-orientedness with the McCallum projection [76].

Two experiments were undertaken, applying machine learning to CAD itself and to QE by CAD. The problems for the former were obtained by simply removing all quantifiers. We performed separate experiments since for quantified problems QEPCAD can use the partial CAD techniques to stop the lifting process early if the outcome is already determined, while the full process is completed for unquantified ones and the two outputs can be quite different.

3.2.3 Dataset for Case Study B

For Case Study B we are more restricted as we need problems that are expressed with a conjunction of at least two equalities in order for their to be a non-trivial GB. From the `nlsat` dataset 493 three-variable problems and 403 four-variable problems fit this criteria, which should have been a sufficient quantity for a machine learning experiment. GB preconditioning was applied to each problem and cell counts from computing the CAD with the original polynomials and their replacement with the GB were computed and compared. For every one of these problems the GB preconditioning was beneficial or made no difference; surprising as the experiments on much smaller datasets [25,101] had shown much greater volatility. It seems that a great deal of the problems involved had no solution to just the equational part (a fact the GB quickly identified) and so no CAD was required. The `nlsat` dataset will need to be widened if it is to be used more extensively for computer algebra research, something that is now underway as part of the SC^2 Project[11] [1].

Since existing datasets were not suitable for the experiment, we had no choice but to generate our own problems. The generation process aimed for an unbiased dataset which would be computationally feasible for computing multiple CADs, and have some comparable structure (number of terms and polynomials) to existing CAD problems.

We generated 1200 problems using the random polynomial generator `randpoly` in Maple- 17. Each problem has two sets of three polynomials; the first to represent conjoined equalities and the second for the other polynomial constraints (respectively E and F from the description in Sect. 2.4). The number of variables was at most 3, labelled x, y, z and under ordering $x \prec y \prec z$; the number of terms per polynomial at most 2; the coefficients were restricted to integers in $[-20, 20]$; and the total degree was varied between 2, 3 and 4 (with 400 problems

[9] http://researchdata.bath.ac.uk/69/.

[10] http://cs.nyu.edu/~dejan/nonlinear/.

[11] http://www.sc-square.org/.

generated for each). A time limit of 300 CPU seconds was set for each CAD computation (all GB computations completed quickly) from which 1062 problems finished to constitute the final dataset. Of these, 75% benefited from GB preconditioning.

We acknowledge the restrictions of random examples, and that the degree bound will result in a higher likelihood of the equations forming zero dimensional ideas. But we note that our randomly generated dataset matches the results of [25,101] with respect to most problems benefiting from GB, but a substantial minority not. So we consider it suitable for the experiment.

3.3 Evaluating the Data

3.3.1 Evaluating CADs

All computations were performed on a 2.4 GHz Intel processor, however this is not actually relevant as we evaluated the CAD performance using cell counts instead of timings. We note that numerous previous studies have shown the CAD cell count to be closely correlated to timings. Using cell count as the measure has the advantages of being discrete, machine independent and (up the theory used) implementation independent. It also correlates with the cost of any post-processing of the CAD (as would be required to produce an equivalent quantifier free formula for example). The time taken to run the heuristics in Case Study A and to compute the GB in Case Study B, were minimal compared to the CAD computations and so not considered in the experiments.

For Case Study A, since each problem has three-variables and all the quantifiers are the same, all six possible variable orderings are admissible. 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. Since our problems are all fully existential, the output for all quantified problems is therefore simply either true or false. So it was not the number of cells in the output that was used to make decisions but instead the number of cells constructed during the process. The best ordering was defined as the possibility resulting in the smallest cell count, (and if more than one ordering gives the minimal both orderings are considered the best).

For Case Study B the variable ordering was assumed fixed so only two CADs needed to be produced for a given example: one each for the input both before and after GB preconditioning. The preconditioning was deemed to be beneficial if the CAD produced following it had less cells that it would have otherwise.

3.3.2 Making a Choice

For Case Study A each problem has six possible orderings to choose from. Rather than choosing between them directly we used SVMs to choose between the choice of the three previously identified heuristics (which seemed to excel on different problems). This is both easier for the current problems and could be applied to those with far more ordering possibilities.

Three classifiers were trained to predict whether each heuristic would make the correct choice for a given problem. In a simple scenario only one of the three classifiers would return a positive result for any given problem (making the choice of best heuristic for the problem immediate). However, in reality more than one classifier will return a positive result for some problems, while no classifiers may return a positive for others. Thus, instead we used the relative magnitudes of the classifiers in our experiment. The classifier with most positive (or least negative) margin value was selected to indicate the recommended decision procedure for the problem (we note that all features are normalised to avoid scaling issues).

For Case Study B the decision of whether to precondition was already a binary choice and there was only one previously known heuristic. So for this experiment we decided to include the metric of that heuristic (TNoI) as a feature of the SVM, aiming to improve upon it with the inclusion of other features to produce the machine learned choice. Thus only a single classifier was trained with its decision being the machine learned choice.

Table 1 The features used to classify examples in Case Study A

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

4 Case Study A: CAD Variable Ordering

In this section we give the full details of the machine learning experiment for Case Study A, starting with the problem features identified in Sect. 4.1, then the parameter optimisation that we performed for the classifier in Sect. 4.2, and the results in Sect. 4.3. We finish by comparing the existing heuristics on the entire problem set in Sect. 4.4 (a detour from the machine learning experiment but prompted due to the interesting results of Sect. 4.3).

4.1 Problem Features

For both experiments (quantified and quantifier free), the datasets were randomly split into training sets (3545 problems each), validation sets (1735 problems each) and test sets (1721 problems each).

To apply machine learning, we need to identify features of the CAD problems that might be relevant to the correct choice of the heuristics. A feature is an aspect or measure of the problem that may be expressed numerically. Table 1 lists the 11 features that we identified. Here (x_0, x_1, x_2) are the three variable labels that are used in all of our problems. 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 all polynomials.

The number of features is quite modest, compared to other machine learning experiments. They were chosen as easily computable features that describe the algebraic structure which could affect the performances of the heuristics.

Each problem contributes a feature vector of the training set which is then associated with a label: $+1$ (positive examples) or -1 (negative examples), indicating in which of two classes it was placed. To take Brown's heuristic as an example, a corresponding training set was derived with each problem labelled $+1$ if Brown's 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 using 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.

4.2 Parameter Optimisation

As discussed in Sect. 2.2, SVMs use kernel functions to map the data into higher dimensional spaces where the data may be more easily separated. The software we use, SVM- LIGHT, has four standard kernel functions: linear, polynomial, sigmoid tanh, and radial basis function [91]. For each kernel function, there are associated parameters which must be set. An earlier experiment by some of the authors applying machine learning to an automated theorem prover [15] compared these four kernels and found the Radial Basis Function (RBF) kernel performed the best in finding a relation between the simple algebraic features and the best heuristic choice. The similarity of that context to the present one meant that we selected the same kernel was selected for this experiment.

The RBF function is defined as

$$K(x, x\prime) = \exp\left(-\gamma ||x - x\prime||^2\right) \tag{2}$$

where K is the kernel function, x and $x\prime$ are feature vectors. There is a single parameter γ in the RBF kernel function (2): as γ rises the SVM seeks a closer fit, but runs the risk over-fitting [91]. Besides γ, two other parameters are involved in the SVM fitting process. The parameter C governs the trade-off between margin and training error, while the cost factor j is used to correct imbalance in the training set. Given a training set, we can easily compute the value of parameter j by looking at the sign of the samples and setting j equal to the ratio between negative and positive samples. However, it is not so trivial to find the optimal values of γ and C.

In machine learning, *Matthews Correlation Coefficient* (MCC) [5,74] is often used to evaluate the performance of the binary classifications. It is defined as

$$\text{MCC} = \frac{\text{TP} * \text{TN} - \text{FP} * \text{FN}}{\sqrt{(\text{TP} + \text{FP})(\text{TP} + \text{FN})(\text{TN} + \text{FP})(\text{TN} + \text{FN})}}, \tag{3}$$

where TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives and FN is the number of false negatives. 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.

A grid-search optimisation procedure 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 MCC in (3). We tested a commonly used range of values in our grid search process [65]: γ took values from $\{2^{-15}, 2^{-14}, 2^{-13}, \ldots, 2^{3}\}$; and C took values from $\{2^{-5}, 2^{-4}, 2^{-3}, \ldots, 2^{15}\}$. Following the completion of the grid-search, the values for the parameters giving optimal MCC results were selected for each of the three CAD heuristic classifiers. We also performed a similar calculation, selecting parameters to maximise the F_1-score [71], but the results using MCC were superior.

The classifiers with optimal (γ, C) were applied to the test set to output the margin values described in Sect. 3.1.2. As discussed in Sect. 3.3.2 the machine learned choice is taken as the heuristic with most positive (or least negative) margin value.

4.3 Results

We use the number of problems for which a selected variable ordering is optimal to measure the efficacy of each heuristic separately, and of the heuristic selected by machine learning.

Table 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 2 breaks down the results into a set of mutually exclusive outcomes that describe all possibilities. The column headed Machine Learning indicates the heuristic selected by the machine learned model with the next three columns indicating each of the fixed heuristics tested. For each of these four heuristics, we may ask the question "Did this heuristic select the optimal variable ordering?" A Y in the table indicates yes and an N indicates no, with each of the 13 cases listed covering all possibilities. Note that at least one of the fixed heuristics must have a Y since, by definition, the optimal ordering is obtained by at least one heuristic, while if they all have a Y it is not possible for machine learning to fail. 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 pick it. 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 2, case 1 is where machine learning cannot make any difference as all heuristics are equally optimal. We compare the remaining cases in pairs (as indicated in the table). Each pair is constructed by keeping the behaviour of the fixed heuristics identical and looking at the two cases where machine learning picked a winning heuristic (one of the ones with a Y) or not. We see in every pair that machine learning succeeds far more often than it fails for. For each pair we can compare with a random heuristic selection. For example, consider the pair formed by cases 2 and 3: where 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, machine learned selection is successful $146/(146+39)$ or approximately 79% of the time, which is significantly better.

We repeated this calculation for the quantified case and the other pairs, as shown in Table 3. In each case the values have been compared to the chance of success when picking a random heuristic, and so there are two distinct sets in Table 3: those where only one heuristic was optimal and those where two are. We see that machine learning performed stronger for some classes of problems than others. For example, in quantifier free examples when only one heuristic is optimal machine learning does considerably better if that one is ndrr; while if only one is not optimal machine learning makes a poorer choice on average if that one is Brown. Nevertheless, the machine learning selection is better than random in every case in both experiments.

By summing the numbers in Table 2 in which Y appears in a row for the machine learned selection and each individual heuristic, we get Table 4. This compares, for both the quantifier free and quantified problem sets, the learned selection with each of the existing heuristics on their own.

Table 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%)

Table 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

Table 5 Comparison of how often each of the existing heuristics makes the best selection on the entire problem set

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

For the quantifier free problems there were 399 problems where every heuristic picked the optimal, 499 where two did and 823 where one did. Hence for this problem set the chance of picking a successful heuristic at random is

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

which compares with $100 * 1312/1721 \simeq 76\%$ for machine learning. For the quantified problems the figures are 64% for a random choice and 77% for a machine learned choice. Hence machine learning performs significantly better than a random choice in both cases.

We can also compare to using only the heuristic that performed the best on this data, the Brown heuristic. With that we would pick a successful ordering for 64% of the quantifier free problems and 74% of the quantified problems. So we see that a machine learned choice is also superior to using any one heuristic, with the improvement most impressive for the quantifier free problems.

4.4 Further Analysis of the Existing Heuristics

Of the three existing heuristics, Brown seems to be the best, albeit by a small margin. Its performance was a little surprising to us, both because the Brown heuristic is not so well known (having never been formally published) and because it requires little computation (taking only simple measurements on the input). We hence took a detour from our machine learning experiment to compare the performance of the three existing heuristics on the full problem set.

We first calculated in Table 5 for how many problems (and thus for what percentage) of the entire problem set each heuristic made the most competitive selection. We see again Brown's heuristic is most likely to make the best choice, both when quantified and when quantifier free.

We next investigate how much of a cell count saving is offered by each heuristic. We made the following calculations for each problem:

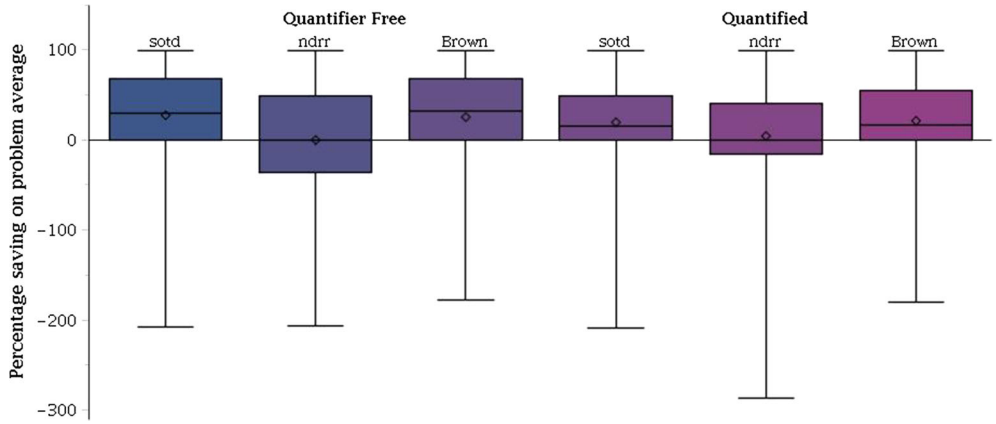

Fig. 1 Box plots of the percentage cell count savings made by the existing heuristics

Table 6 Average percentage cell count savings of the existing heuristics

	Mean average			Median value		
	sotd (%)	ndrr (%)	Brown (%)	sotd (%)	ndrr (%)	Brown (%)
Quantifier free	27.32	− 0.20	25.28	29.47	0.00	32.28
Quantified	19.47	4.15	21.03	14.68	0.00	16.67

Table 7 How often each heuristics avoids a time out on the set of problems where this occurred for at least one choice

	sotd	ndrr	Brown
Quantifier free	559	537	594
Quantified	512	530	478

(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 chosen variable ordering timed out (5262 of the quantifier free problems and 5332 of the quantified problems). The data is visualised in the box plots of Fig. 1, where the boxes indicate the second and third quartiles. The mean and median values are given in Table 6 (and marked on the plots with circles and lines respectively).

While Brown's heuristic makes the best choice the most frequently, for the quantifier free problems the average saving of using sotd is actually higher. Ndrr performs the worst on average, but as we saw in the previous section there is a substantial class of problems where it makes a better choice than the others. This is illustrated if we consider the problems excluded from the previous discussion: those where at least one ordering timed out. Table 7 describes how often each heuristic avoids a time out. We see that for quantified problems ndrr does the best.

5 Case Study B: Preconditioning CAD with a Gröbner Basis

In this section we give the full details of the machine learning experiment for Case Study B, starting with the problem features identified in Sect. 5.1, then the initial experiment and results obtained in Sect. 5.2, the feature

selection process that was run leading to improved results in Sect. 5.3, and finally a comparison with the human developed heuristic in Sect. 5.4.

5.1 Problem Features

Table 8 shows the 28 problem features we identified for Case Study B. They fall into three sets: those generated from the polynomials in the original problem (#1–12), those obtained after applying GB preconditioning (#13–25), and those involving both (#26–28). There is one more feature for the input after GB because the number of polynomials before GB was the same for each problem.

In the feature descripions x, y, and z are the three variable labels used in the problems (the variable ordering was fixed as $x \prec y \prec z$). The abbreviations `tds` and `stds` stand for *maximum total degree* and *sum of total degrees* respectively:

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

Note that the `stds` measure differs from the `sotd` heuristic from Sect. 2.3. This is because `stds` measures the input polynomials only, while `sotd` measures the full set of CAD projection polynomials, and so is much more expensive.

We started with those features used for Case Study A and extended by including the simpler degree measure and the metric `TNoI` (see Eq. (1) from Sect. 2.4). Finally we included the base 2 logarithm of the ratio of differences between some of the key metrics. All features could be calculated immediately with MAPLE.

In addition to training a classifier using all the features in Table 8, we trained classifiers using two subsets: one containing the features labelled 1–12 concerning the original set of polynomials; and one with the features labelled 13–25 concerning the polynomials after GB preconditioning. We refer to the first subset as *before features*, the second as *after features* and the full set as *all features*.

As an example, consider sets of polynomials

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

The GB computed for E is

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

Then the *all features* vector becomes

$$\Big[12, 12, 2, 2, 1, 1, \tfrac{2}{3}, \tfrac{2}{3}, \tfrac{2}{3}, \tfrac{1}{3}, \tfrac{5}{12}, \tfrac{5}{12},$$
$$6, 10, 10, 2, 2, 1, 1, \tfrac{2}{3}, \tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{3}, \tfrac{1}{3}, \tfrac{1}{4},$$
$$0.263, 2.263, 0\Big],$$

with the *before features* and *after features* vectors formed from the first and second lines respectively.

The feature generation process was applied to create the three training sets separately (although the feature labels used were all as in Table 8). Each problem was labelled $+1$ if GB preconditioning is beneficial for CAD construction, or -1 otherwise. After feature generation the training data was standardised so each feature had zero mean and unit variance across the training set. The same standardisation was then applied to features in the test set.

Table 8 Initial feature set to classify examples in Case Study B

Feature number	Description
1	`TNoI` before GB
2	`stds` before GB
3	`tds` of polynomials before GB
4	Max degree of x in polynomials before GB
5	Max degree of y in polynomials before GB
6	Max degree of z in polynomials before GB
7	Proportion of polynomials with x before GB
8	Proportion of polynomials with y before GB
9	Proportion of polynomials with z before GB
10	Proportion of monomials with x before GB
11	Proportion of monomials with y before GB
12	Proportion of monomials with z before GB
13	Number of polynomials after GB
14	`TNoI` after GB
15	`stds` after GB
16	`tds` of polynomials after GB
17	Max degree of x in polynomials after GB
18	Max degree of y in polynomials after GB
19	Max degree of z in polynomials after GB
20	Proportion of polynomials with x after GB
21	Proportion of polynomials with y after GB
22	Proportion of polynomials with z after GB
23	Proportion of monomials with x after GB
24	Proportion of monomials with y after GB
25	Proportion of monomials with z after GB
26	$\log_2$(`TNoI` before GB) − $\log_2$(`TNoI` after GB)
27	$\log_2$(`stds` before GB) − $\log_2$(`stds` after GB)
28	$\log_2$(`tds` before GB) − $\log_2$(`tds` after GB)

5.2 Initial Experiment and Results

The 1062 problems were partitioned into 80% training (849 problems) and 20% test (213 problems), stratified to maintain relative proportions of positive and negative examples.

The classification was done in SVM- LIGHT using the *radial basis function* (RBF) kernel; with parameter values chosen to optimize MCC using a grid-search optimisation procedure with a fivefold stratified cross validation used. The details are the same as for Case Study A so we refer to Sect. 4.2 for details. This procedure was repeated for the three feature sets.

The classification accuracy was used to measure the efficacy of the machine learning decisions. The test set of 213 problems contained 159 positive samples and 54 negative samples (i.e. 75% of the test problems benefited from GB preconditioning).

The results of the machine learned choices are summarised in Table 9. First we note that when making a choice based on the *before features* training set 75% of the problems were predicted accurately. I.e. making a decision based on these features results in no more correct decisions than blindly deciding to GB precondition each and

Table 9 Accuracy of predictions before feature selection

Feature set	Number	% of test set
All features	162	76
Before features	159	75
After features	167	78

every time. However, the other two feature sets resulted in superior decisions. Although only a small improvement on preconditioning blindly, the reader should recall that the wrong choice can give large changes to the size of the CAD or even change the tractability of the problem [25,101].

The results certainly indicate that the features of the GB itself are required to decide whether to use the preconditioning. They seem to indicate that the features from before the GB was applied actually hinder the learning process, but this would be an unsound conclusion. Earlier research showed that a variable completely useless by itself can provide a significant performance improvement when taken in conjunction with others [56]. To be confident about which features were significant and which were superfluous, further feature selection experiments are required, as described in the next section. We will see that the optimal feature subset must contain features from both before and after the GB computation.

5.3 Feature Selection

The feature selection experiments were conducted with WEKA (Waikato Environment for Knowledge Analysis) [57], a Java machine learning library which supports tasks such as data preprocessing, clustering, classification, regression and feature selection. Each data point is represented as a fixed number of features. The inputs are samples of 29 features, where the first 28 are the real-valued features from Table 8, and the final one is a *nominal* feature denoting its class.

5.3.1 The Filter Method

First we applied a correlation based feature selection method as described in [59]. Unlike other filter methods these measure the rank of feature subsets instead of individual features [58]. A feature subset which contains features highly correlated with the class but uncorrelated with each other is preferred.

The metric below is used to measure the quality of a feature subset, and takes into account feature-class correlation as well as feature-feature correlation.

$$G_s = \frac{k\overline{r_{ci}}}{\sqrt{k + k(k-1)\overline{r_{ii'}}}} \tag{4}$$

Here, k is the number of features in the subset, $\overline{r_{ci}}$ denotes the average feature-class correlation of feature i, and $\overline{r_{ii'}}$ the average feature-feature correlation between feature i and i'. The numerator of Eq. (4) 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 G_s, the better the feature subset.

To apply this heuristic we must calculate the correlations. With the exception of the class attribute all 28 features are continuous, so in order to have a common measure for computing the correlations we first discretize using the method of Fayyad and Irani [53]. After that, a correlation measure based on the information-theoretical concept of entropy is used: a measure of the uncertainty of a random variable. We define the *entropy of a variable X* [90] as

$$H(X) = -\sum_i p(x_i) \log_2 \left(p(x_i)\right). \tag{5}$$

Table 10 Feature selection by the filter method

Feature number	Description
14	`TNoI` after GB
13	Number of polynomials after GB
2	`stds` before GB
26	$\log_2$(`TNoI` before GB) − $\log_2$(`TNoI` after GB)
21	Proportion of polynomials with y after GB
15	`stds` after GB
23	Proportion of monomials with x after GB
19	Max degree of z in polynomials after GB
25	Proportion of monomials with z after GB
27	$\log_2$(`stds` before GB) − $\log_2$(`stds` after GB)

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

$$H(X|Y) = -\sum_j p(y_j) \sum_i p(x_i|y_j) \log_2 \left(p(x_i|y_j)\right), \tag{6}$$

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)* measures the amount by which the entropy of X decreases by additional information about X provided by Y [86]. It is given by

$$IG(X, Y) = H(X) - H(X|Y). \tag{7}$$

The *symmetrical uncertainty (SU)* (a modified information gain measure) is then used to measure the correlation between two discrete variables (X and Y) [85]:

$$SU(X, Y) = 2.0 \times \left(\frac{H(X) - H(X|Y)}{H(X) + H(Y)} \right). \tag{8}$$

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 Eq. (4).

Recall that our aim here is to find the optimal subset of features which maximises the metric given in Eq. (4). The size of our feature set is 28 meaning there are

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

possible subsets, 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 metric from Eq. (4) is computed for every single feature addition, and the feature with the best score improvement 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 (which may not be the absolute optimal subset of features) is shown in Table 10, ordered by importance.

Table 11 Feature selection by the wrapper method

Feature number	Description
14	TNoI after GB
9	Proportion of polynomials with z before GB
22	Proportion of polynomials with z after GB
4	Max degree of x in polynomials before GB
12	Proportion of monomials with z before GB

5.3.2 The Wrapper Method

The wrapper feature selection method evaluates attributes using accuracy estimates provided by the target learning algorithm. Evaluation of each feature set was conducted with a SVM with RBF kernel function. The SVM algorithm is run on the dataset, with the same data partitions as used previously. Similarly, a fivefold 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 each training / validation fold starting with an empty set of features: each feature was added; a model was fitted to the training dataset; the classifier was then tested on the validation set. This was done on all the features giving a score for each reflecting 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. The same greedy procedure as used for the filter method in Sect. 5.3.1 was applied to obtain the best feature subset.

Due to the large number of cases, the parameters (C, γ) were selected from an optimised sub range instead of the full grid search used previously. This suffices to demonstrate the performance of a reduced feature set. The experiments of Sect. 5.2 found (for all three feature sets) that C taken from $\{2^5, 2^6, 2^7, 2^8, 2^9, 2^{10}\}$ and γ taken from $\{2^{-5}, 2^{-6}, 2^{-7}, 2^{-8}, 2^{-9}, 2^{-10}\}$ provided good classifier performance.

The 36 pairs of (C, γ) values were tested and an optimal feature subset with the highest accuracy was found for each. Then the one with the highest accuracy was selected as the final set, which is shown in Table 11 ordered by importance. We see that most of the features selected (9, 12 and 22) related to variable z. Recall that the projection order used in the CAD was always $x \prec y \prec z$, i.e. the variable z is projected first. Hence it makes sense that this variable would have the greatest effect and thus be identified in the feature selection.

We examined the performance on further reduced feature sets, obtained by the feature ranking of the wrapper method. Figure 2 shows the overall prediction accuracies of a sample classifier. We see that when using a single feature (the best ranked feature was TNoI after GB for both filter and wrapper methods) this predictor achieved an accuracy score of 0.756 in this run. Then the performance steadily increases with the size of the feature set until the fifth feature; but taking any sixth feature gives no further improvement and hence resulted in the cut-off chosen by the wrapper method.

As the wrapper method identified only a few features an error analysis on the misclassified data points is feasible. Figure 3 shows 40 misclassified points and their values for features 4 and 14, while Fig. 4 shows the remaining features (9, 12, and 22). It is interesting that feature 4 of all misclassified samples is either 1 or 2, when for the whole dataset roughly a third of samples had this feature value 3 or 4. This indicates that the algorithm performs better on instances with a higher maximum degree of x among all polynomials before GB preconditioning.

5.3.3 Results with Reduced Feature Sets

Having obtained the reduced feature sets, we ran the experiment again to evaluate the new choices. The dataset was again repartitioned into 80% training and 20% test set, 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

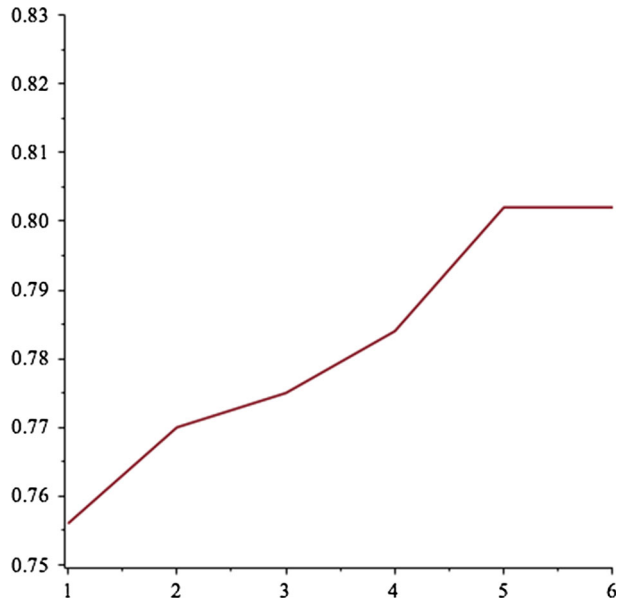

Fig. 2 Performance of a sample classifier with different sizes of feature sets. The vertical axis measures prediction accuracy and the horizontal the number of features

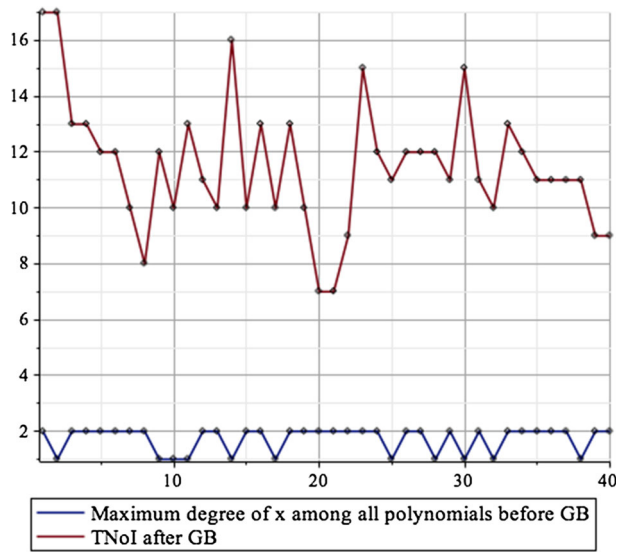

Fig. 3 Values of features 4 and 14 for 40 misclassified data points

the original range of (C, γ) pairs was conducted. The choices with maximum averaged MCC were selected and the resulting classifier was then evaluated.

The testing 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 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 shows boxplots of the accuracies generated by 50 runs of the five-fold cross validation. Both reduced feature sets generated similar results and show an improvement on the base case where GB preconditioning is always used before CAD construction.

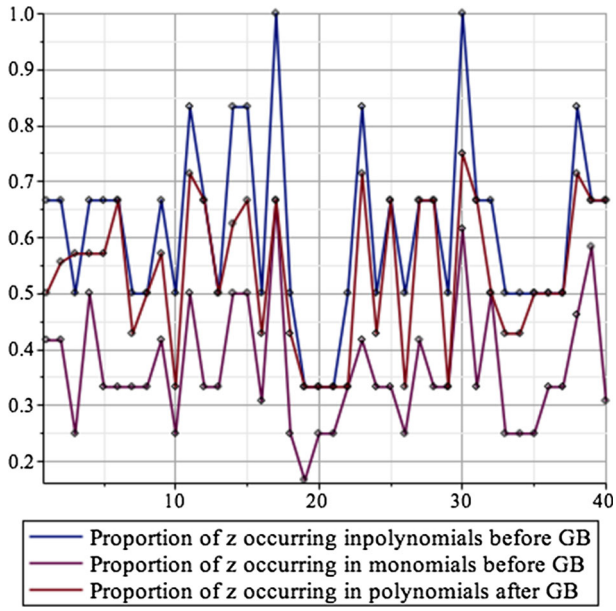

Fig. 4 Values of features 9, 12 and 22 for 40 misclassified data points

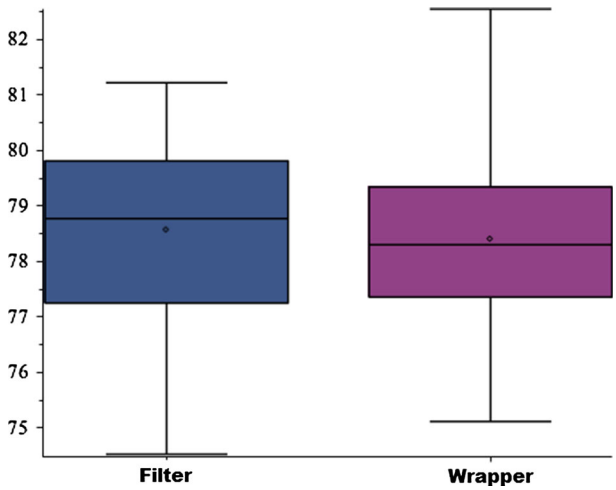

Fig. 5 Boxplots of prediction accuracys from 50 runs of the fivefold cross validation with the suggested feature sets from the two feature selection methods

The average overall prediction accuracy of the filter subset and the wrapper subset is 79% and 78% respectively (Fig. 2 shows a higher rate but that was just for one sample run). All 50 runs of the wrapper subset performed above the base line, while the top three quartiles of the results of both sets achieve higher than 77% percentage accuracy.

5.4 Comparison with the Human Developed Heuristic

We may compare the machine learned choice with the human developed `TNoI` heuristic [101], whose performance on the 213 test problems is shown in Table 12. It correctly predicted whether GB preconditioning was beneficial for 118 examples, only 55%. So for this dataset it would have been better on average to precondition blindly than to make a decision using `TNoI` alone. The `TNoI` heuristic performed better in the experiments by Wilson et al. [101]. Those experiments involved only 22 problems (compared to 213 in the test set here) but they were human constructed to have certain geometric properties while the ones here were random.

Table 12 The performance of the TNoI-based heuristic [101]

	Total	Correct prediction
GB beneficial	158	77 (48%)
GB not beneficial	54	41 (76%)
Total	213	118 (55%)

We also note that the TNoI heuristic actually performed quite differently for positive and negative examples of our dataset, as shown by the separated data in Table 12. It was able to identify most of the cases where GB-preconditioning is detrimental but failed to identify many of the cases where it was beneficial. The TNoI after GB was identified as important by both feature methods, but it seems to need to be used in conjunction with other features to be effective here.

6 Final Thoughts

6.1 Summary

We have presented two case studies that apply machine learning techniques (specifically support vector machines) to make choices for CAD problem instances: the variable ordering to use and whether to precondition with a Gröbner Basis. Both case studies showed the potential of machine learning to outperform human developed heuristics.

For Case Study A the experimental results confirmed our thesis, drawn from personal experience, 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. Using machine learning to select the best CAD heuristic yielded considerably better results than choosing one heuristic at random, or just using any of the individual heuristics in isolation, indicating there is a relation between the simple algebraic features and the best heuristic choice. This could lead to the development of a new individual heuristic in the future. We note that we observed Brown's heuristic to be the most competitive for our example set, despite it involving less computation than the others. This heuristic was presented during an ISSAC tutorial in 2004 [19], but does not seem to be formally published. It certainly deserves to be better known.

For Case Study B a machine learned choice on whether to precondition was found to yield better results than either always preconditioning blindly, or using the previously human developed TNoI heuristic [101]. Two feature selection experiments showed that a small feature subset could be used to achieve this. The two subsets identified were different but both needed features from before and after the GB preconditioning. For one, having fewer features actually improved the learning efficiency. The smaller improvement achieved by machine learning in Case Study B is to be expected since it was a binary choice in which one option is known to usually be better. It is still significant as making the wrong choice can greatly increase the size of the CAD, or even change the tractability of the problem.

6.2 Datasets

As discussed in Sect. 3.2 we had some difficulties obtaining suitable data for the experiments. For the former experiment we had to employ a dataset from a different field (SMT solving) to obtain enough problems to train the classifier. This restricted the problems to those with a single block of existential quantifiers. In the latter experiment we had to create a brand new dataset of random polynomials. We acknowledge that it would be preferable to run such experiments on an established dataset of meaningful problems but this was not simply not available. We emphasise the interesting finding that the established nlsat dataset has a hidden uniformity despite its size.[12] It should be

[12] Something that was noted in multiple presentations at the inaugural International Workshop on Satisfiability Checking and Symbolic Computation in 2016 and is now being addressed by the SC^2 Project [1].

noted that the randomly created dataset used matched the previously reported results on the topic of investigation [25,101].

The experiments described involved building 1000s of CADs, certainly the largest such experiment that the authors are aware of. For comparison, the best known previous study on CAD variable ordering heuristics [41] tested with six examples.

6.3 Future Work

First, as noted above it is desirable to re-run both experiments on further datasets of problems as they become available. There is a large set derived from university mathematics entrance exams [73], which but may be publicly available in the future. An obvious extension of Case Study A would be to test additional heuristics, such as the greedy sotd heuristic [41], one based on the number of full dimensional cells [99], or those developed for CAD by Regular Chains [46]. For Case Study B there are now further CAD optimisations for problems with multiple equalities under development [39,44,47] which may affect the role of GB preconditioning from CAD. Also, as pointed out by one of the referees, one could consider trying to classify whether to reduce individual inequalities with the GB.

We choose SVMs with an RBF kernel for these experiments based on some similar work for theorem provers but 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 the feature selection. The features used here were all derived from the polynomials involved in the input. One possible extension would be to consider also the type of relations present and how they are connected logically. This is likely to be increasingly beneficial as new CAD theory is developed which takes this into account [10,12,13,20,44,72].

These two experiments were conducted independently: the possibility of GB preconditioning was not considered in Case Study A while in Case Study B the variable ordering for CAD and the monomial ordering for GB were fixed. In reality such decisions may need to be taken in tandem and it is possible that preconditioning could change the best choice of variable ordering and the variable ordering change whether we would precondition. How best to tackle this, other than applying heuristics in serial, is another topic for future consideration.

Finally, we note that CAD is not unique amongst computer algebra algorithms in requiring the user to make such a choice of problem formulation. Computer algebra systems (CASs) often have a choice of possible algorithms to use when solving a problem. Since a single formulation or algorithm is rarely the best for the entire problem space, CASs usually use *meta-algorithms* to make such choices, where decisions are based on some numerical parameters [27]. These are often not as well documented as the base algorithms, and may be rather primitive. To the best of our knowledge, the present paper appears to be the first applying machine learning to problem formulation for computer algebra. The positive results should encourage investigation of similar applications throughout the field.[13]

Acknowledgements This work was supported by EPSRC Grant EP/J003247/1; the European Union's Horizon 2020 research and innovation programme under Grant Agreement No 712689 (SC^2); and the China Scholarship Council (CSC). The authors acknowledge both the anonymous referees of the present paper, and those of [66,67], whose comments also helped improve this article.

References

1. Ábrahám, E., Abbott, J., Becker, B., Bigatti, A.M., Brain, M., Buchberger, B., Cimatti, A., Davenport, J.H., England, M., Fontaine, P., Forrest, S., Griggio, A., Kroening, D., Seiler, W.M., Sturm, T.: SC^2: satisfiability checking meets symbolic computation. In:

[13] Since this article went to press such investigations are taking place: for example, see the special session at the 2018 International Congress on Mathematical Software on Machine Learning for Mathematical Software, as surveyed by [42].

Kohlhase, M., Johansson, M., Miller, B., de Moura, L., Tompa, F. (eds.) Intelligent Computer Mathematics: Proceedings CICM 2016, volume 9791 of Lecture Notes in Computer Science, pp. 28–43. Springer (2016)
2. Alpaydin, E.: Introduction to Machine Learning. MIT Press, Cambridge (2004)
3. Arai, N.H., Matsuzaki, T., Iwane, H., Anai, H.: Mathematics by machine. In: Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation, ISSAC '14, pp. 1–8. ACM (2014)
4. Arnon, D., Collins, G.E., McCallum, S.: Cylindrical algebraic decomposition I: the basic algorithm. SIAM J. Comput. **13**, 865–877 (1984)
5. Baldi, P., Brunak, S., Chauvin, Y., Andersen, C.A.F., Nielsen, H.: Assessing the accuracy of prediction algorithms for classification: an overview. Bioinformatics **16**(5), 412–424 (2000)
6. Basu, S., Pollack, R., Roy, M.E.: Algorithms in Real Algebraic Geometry. Volume 10 of Algorithms and Computations in Mathematics. Springer, Berlin (2006)
7. Böge, W., Gebauer, R., Kredel, H.: Gröbner bases using SAC2. In: Caviness, B.F. (ed.) EUROCAL '85, Volume 204 of Lecture Notes in Computer Science, pp. 272–274. Springer, Berlin (1985)
8. Boser, B.E., Guyon, I.M., Vapnik, V.N.: A training algorithm for optimal margin classifiers. In: Proceedings of the Fifth Annual Workshop on Computational Learning Theory, COLT '92, pp. 144–152. ACM (1992)
9. Boyan, J., Freitag, D., Joachims, T.: A machine learning architecture for optimizing web search engines. In: AAAI Workshop on Internet Based Information Systems, pp. 1–8 (1996)
10. Bradford, R., Chen, C., Davenport, J.H., England, M., Moreno Maza, M., Wilson, D.: Truth table invariant cylindrical algebraic decomposition by regular chains. In: Gerdt, V.P., Koepf, W., Seiler, W.M., Vorozhtsov, E.V. (eds.) Computer Algebra in Scientific Computing, Volume 8660 of Lecture Notes in Computer Science, pp. 44–58. Springer, Berlin (2014)
11. Bradford, R., Davenport, J.H., England, M., Errami, H., Gerdt, V., Grigoriev, D., Hoyt, C., Košta, M., Radulescu, O., Sturm, T., Weber, A.: A case study on the parametric occurrence of multiple steady states. In: Proceedings of the 2017 ACM International Symposium on Symbolic and Algebraic Computation, ISSAC '17, pp. 45–52. ACM (2017)
12. Bradford, R., Davenport, J.H., England, M., McCallum, S., Wilson, D.: Cylindrical algebraic decompositions for boolean combinations. In: Proceedings of the 38th International Symposium on Symbolic and Algebraic Computation, ISSAC '13, pp. 125–132. ACM (2013)
13. Bradford, R., Davenport, J.H., England, M., McCallum, S., Wilson, D.: Truth table invariant cylindrical algebraic decomposition. J. Symb. Comput. **76**, 1–35 (2016)
14. Bradford, R., Davenport, J.H., England, M., Wilson, D.: Optimising problem formulations for cylindrical algebraic decomposition. In: Carette, J., Aspinall, D., Lange, C., Sojka, P., Windsteiger, W. (eds.) Intelligent Computer Mathematics, Volume 7961 of Lecture Notes in Computer Science, pp. 19–34. Springer, Berlin (2013)
15. Bridge, J.P.: Machine learning and automated theorem proving. Technical Report UCAM-CL-TR-792, University of Cambridge, Computer Laboratory (2010)
16. Bridge, J.P., Holden, S.B., Paulson, L.C.: Machine learning for first-order theorem proving. J. Autom. Reason. 1–32 (2014)
17. Brown, C.W.: Improved projection for cylindrical algebraic decomposition. J. Symb. Comput. **32**(5), 447–465 (2001)
18. Brown, C.W.: QEPCAD B: a program for computing with semi-algebraic sets using CADs. ACM SIGSAM Bull. **37**(4), 97–108 (2003)
19. Brown, C.W.: Companion to the tutorial: cylindrical algebraic decomposition. Presented at ISSAC '04 (2004). http://www.usna.edu/Users/cs/wcbrown/research/ISSAC04/handout.pdf
20. Brown, C.W.: Open non-uniform cylindrical algebraic decompositions. In: Proceedings of the 2015 International Symposium on Symbolic and Algebraic Computation, ISSAC '15, pp. 85–92. ACM (2015)
21. Brown, C.W., Davenport, J.H.: The complexity of quantifier elimination and cylindrical algebraic decomposition. In: Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation, ISSAC '07, pp. 54–60. ACM (2007)
22. Brown, C.W., El Kahoui, M., Novotni, D., Weber, A.: Algorithmic methods for investigating equilibria in epidemic modelling. J. Symb. Comput. **41**, 1157–1173 (2006)
23. Brown, C.W., Kosta, M.: Constructing a single cell in cylindrical algebraic decomposition. J. Symb. Comput. **70**, 14–48 (2015)
24. Buchberger, B.: Bruno Buchberger's Ph.D. thesis (1965): an algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal. J. Symb. Comput. **41**(3–4), 475–511 (2006)
25. Buchberger, B., Hong, H.: Speeding up quantifier elimination by Gröbner bases. Technical report, 91-06. RISC, Johannes Kepler University (1991)
26. Byun, H., Lee, S.: A survey on pattern recognition applications of support vector machines. Int. J. Pattern Recognit. Artif. Intell. **17**(03), 459–486 (2003)
27. Carette, J.: Understanding expression simplification. In: Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation, ISSAC '04, pp. 72–79. ACM (2004)
28. Caviness, B., Johnson, J.: Quantifier Elimination and Cylindrical Algebraic Decomposition. Texts and Monographs in Symbolic Computation. Springer, Berlin (1998)
29. Charalampakis, A.E., Chatzigiannelis, I.: Analytical solutions for the minimum weight design of trusses by cylindrical algebraic decomposition. Arch. Appl. Mech. **88**(1), 39–49 (2018)
30. Chen, C., Moreno Maza, M.: An incremental algorithm for computing cylindrical algebraic decompositions. In: Feng, R., Lee, W., Sato, Y. (eds.) Computer Mathematics, pp. 199–221. Springer, Berlin (2014)

31. Chen, C., Moreno Maza, M.: Real quantifier elimination in the RegularChains library. In: Hong, H., Yap, C. (eds.) Mathematical Software—ICMS 2014, Volume 8592 of Lecture Notes in Computer Science, pp. 283–290. Springer, Heidelberg (2014)
32. Chen, C., Moreno Maza, M.: Quantifier elimination by cylindrical algebraic decomposition based on regular chains. Journal of Symbolic Computation **75**, 74–93 (2016)
33. Chen, C., Moreno Maza, M., Xia, B., Yang, L.: Computing cylindrical algebraic decomposition via triangular decomposition. In: Proceedings of the 2009 International Symposium on Symbolic and Algebraic Computation, ISSAC '09, pp. 95–102. ACM (2009)
34. Collins, G.E.: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In: Proceedings of the 2nd GI Conference on Automata Theory and Formal Languages, pp. 134–183. Springer (reprinted in the collection [28]) (1975)
35. Collins, G.E.: The SAC-2 computer algebra system. In: Caviness, B.F. (ed.) EUROCAL '85, Volume 204 of Lecture Notes in Computer Science, pp. 34–35. Springer, Berlin (1985)
36. Collins, G.E., Hong, H.: Partial cylindrical algebraic decomposition for quantifier elimination. J. Symb. Comput. **12**, 299–328 (1991)
37. Cortes, C., Vapnik, V.: Support-vector networks. Mach. Learn. **20**(3), 273–297 (1995)
38. Davenport, J.H., Bradford, R., England, M., Wilson, D.: 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, pp. 83–88. IEEE (2012)
39. Davenport, J.H., England, M.: Need polynomial systems be doubly exponential? In: Greuel, G.M., Koch, T., Paule, P., Sommese, A. (eds.) Mathematical Software—Proceedings of ICMS 2016, Volume 9725 of Lecture Notes in Computer Science, pp. 157–164. Springer, Berlin (2016)
40. Davenport, J.H., Heintz, J.: Real quantifier elimination is doubly exponential. J. Symb. Comput. **5**(1–2), 29–35 (1988)
41. Dolzmann, A., Seidl, A., Sturm, T.: Efficient projection orders for CAD. In: Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation, ISSAC '04, pp. 111–118. ACM (2004)
42. England, M.: Machine Learning for Mathematical Software. In: Davenport, J.H., Kauers, M., Labahn, G., Urban, J. (eds.) Mathematical Software—ICMS 2018, Volume 10931 of Lecture Notes in Computer Science, pp. 165–174. Springer, Heidelberg (2018)
43. England, M., Bradford, R., Chen, C., Davenport, J.H., Moreno Maza, M., Wilson, D.: Problem formulation for truth-table invariant cylindrical algebraic decomposition by incremental triangular decomposition. In: Watt, S.M., Davenport, J.H., Sexton, A.P., Sojka, P., Urban, J. (eds.) Intelligent Computer Mathematics, Volume 8543 of Lecture Notes in Artificial Intelligence, pp. 45–60. Springer, Berlin (2014)
44. England, M., Bradford, R., Davenport, J.H.: Improving the use of equational constraints in cylindrical algebraic decomposition. In: Proceedings of the 2015 International Symposium on Symbolic and Algebraic Computation, ISSAC '15, pp. 165–172. ACM (2015)
45. England, M., Bradford, R., Davenport, J.H., Wilson, D.: Understanding branch cuts of expressions. In: Carette, J., Aspinall, D., Lange, C., Sojka, P., Windsteiger, W. (eds.) Intelligent Computer Mathematics, Volume 7961 of Lecture Notes in Computer Science, pp. 136–151. Springer, Berlin (2013)
46. England, M., Bradford, R., Davenport, J.H., Wilson, D.: Choosing a variable ordering for truth-table invariant cylindrical algebraic decomposition by incremental triangular decomposition. In: Hong, H., Yap, C. (eds.) Mathematical Software—ICMS 2014, Volume 8592 of Lecture Notes in Computer Science, pp. 450–457. Springer, Heidelberg (2014)
47. England, M., Davenport, J.H.: The complexity of cylindrical algebraic decomposition with respect to polynomial degree. In: Gerdt, V.P., Koepf, W., Werner, W.M., Vorozhtsov, E.V. (eds.) Computer Algebra in Scientific Computing: 18th International Workshop, CASC 2016, Volume 9890 of Lecture Notes in Computer Science, pp. 172–192. Springer (2016)
48. England, M., Errami, H., Grigoriev, D., Radulescu, O., Sturm, T., Weber, A.: Symbolic versus numerical computation and visualization of parameter regions for multistationarity of biological networks. In: Gerdt, V.P., Koepf, W., Seiler, W.M., Vorozhtsov, E.V. (eds.) Computer Algebra in Scientific Computing (CASC), Volume 10490 of Lecture Notes in Computer Science, pp. 93–108. Springer, Berlin (2017)
49. England, M., Wilson, D., Bradford, R., Davenport, J.H.: Using the regular chains library to build cylindrical algebraic decompositions by projecting and lifting. In: Hong, H., Yap, C. (eds.) Mathematical Software—ICMS 2014, Volume 8592 of Lecture Notes in Computer Science, pp. 458–465. Springer, Berlin (2014)
50. Erascu, M., Hong, H.: Real quantifier elimination for the synthesis of optimal numerical algorithms (case study: square root computation). J. Symb. Comput. **75**, 110–126 (2016)
51. Errami, H., Eiswirth, M., Grigoriev, D., Seiler, W.M., Sturm, T., Weber, A.: Detection of Hopf bifurcations in chemical reaction networks using convex coordinates. J. Comput. Phys. **291**, 279–302 (2015)
52. Faugère, J.C.: A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). In: Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation, ISSAC '02, pp. 75–83. ACM (2002)
53. Fayyad, U.M., Irani, K.B.: Multi-interval discretization of continuous-valued attributes for classification learning. In: Proceedings of the International Joint Conference on Uncertainty in AI, pp. 1022–1027. http://trs-new.jpl.nasa.gov/dspace/handle/2014/35171 (1993)
54. Forsyth, R., Rada, R.: Machine Learning: Applications in Expert Systems and Information Retrieval. Halsted Press, New York (1986)
55. Graebe, H.G., Nareike, A., Johanning, S.: The SymbolicData project: towards a computer algebra social network. In: England, M., Davenport, J.H., Kohlhase, A., Kohlhase, M., Libbrecht, P., Neuper, W., Quaresma, P., Sexton, A.P., Sojka, P., Urban, J., Watt,

S.M. (eds.) Joint Proceedings of the MathUI, OpenMath and ThEdu Workshops and Work in Progress Track at CICM, Number 1186 in CEUR Workshop Proceedings (2014)
56. Guyon, I., Elisseeff, A.: An introduction to variable and feature selection. J. Mach. Learn. Res. **3**, 1157–1182 (2003)
57. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software: an update. SIGKDD Explor. Newsl. **11**(1), 10–18 (2009)
58. Hall, M.A.: Correlation-based feature selection for discrete and numeric class machine learning. In: Proceedings of the Seventeenth International Conference on Machine Learning, ICML '00, pp. 359–366. Morgan Kaufmann Publishers Inc. (2000)
59. Hall, M.A., Holmes, G.: Benchmarking attribute selection techniques for discrete class data mining. IEEE Trans. Knowl. Data Eng. **15**(6), 1437–1447 (2003)
60. Han, J., Dai, L., Xia, B.: Constructing fewer open cells by gcd computation in CAD projection. In: Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation, ISSAC '14, pp. 240–247. ACM (2014)
61. Heinle, A., Levandovskyy, V.: The SDEval benchmarking toolkit. ACM Commun. Comput. Algebra **49**(1), 1–9 (2015)
62. Hong, H.: An improvement of the projection operator in cylindrical algebraic decomposition. In: Proceedings of the International Symposium on Symbolic and Algebraic Computation, ISSAC '90, pp. 261–264. ACM (1990)
63. Hong, H.: Comparison of several decision algorithms for the existential theory of the reals. Technical report, RISC, Linz (1991)
64. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Netw. **2**(5), 359–366 (1989)
65. Hsu, C., Chang, C., Lin, C.: A practical guide to support vector classification. Technical report, Department of Computer Science, National Taiwan University (2003)
66. Huang, Z., England, M., Davenport, J.H., Paulson, L.: Using machine learning to decide when to precondition cylindrical algebraic decomposition with Groebner bases. In: 18th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC '16), pp. 45–52. IEEE (2016)
67. Huang, Z., England, M., Wilson, D., Davenport, J.H., Paulson, L., Bridge, J.: Applying machine learning to the problem of choosing a heuristic to select the variable ordering for cylindrical algebraic decomposition. In: Watt, S.M., Davenport, J.H., Sexton, A.P., Sojka, P., Urban, J. (eds.) Intelligent Computer Mathematics, Volume 8543 of Lecture Notes in Artificial Intelligence, pp. 92–107. Springer, Berlin (2014)
68. Huang, Z., Paulson, L.: An application of machine learning to RCF decision procedures. In: 20th Automated Reasoning Workshop, University of Dundee, UK, ARW '13 (2013)
69. Iwane, H., Yanami, H., Anai, H., Yokoyama, K.: An effective implementation of a symbolic-numeric cylindrical algebraic decomposition for quantifier elimination. In: Proceedings of the 2009 Conference on Symbolic Numeric Computation, SNC '09, pp. 55–64 (2009)
70. Joachims, T.: Making large-scale support vector machine learning practical. In: Schölkopf, B., Burges, C.J.C., Smola, A.J. (eds.) Advances in Kernel Methods, pp. 169–184. MIT Press, Cambridge (1999)
71. Joachims, T.: A support vector method for multivariate performance measures. In: Proceedings of the 22nd International Conference on Machine Learning, ICML '05, pp. 377–384. ACM (2005)
72. Jovanovic, D., de Moura, L.: Solving non-linear arithmetic. In: Gramlich, B., Miller, D., Sattler, U. (eds.) Automated Reasoning: 6th International Joint Conference (IJCAR), Volume 7364 of Lecture Notes in Computer Science, pp. 339–354. Springer (2012)
73. Kobayashi, M., Iwane, H., Matsuzaki, T., Anai, H.: Efficient subformula orders for real quantifier elimination of non-prenex formulas. In: Kotsireas, S.I., Rump, M.S., Yap, K.C. (eds.) Mathematical Aspects of Computer and Information Sciences (MACIS '15), Volume 9582 of Lecture Notes in Computer Science, pp. 236–251. Springer, Berlin (2016)
74. Matthews, B.W.: Comparison of the predicted and observed secondary structure of T4 phage lysozyme. Biochim. Biophys. Acta (BBA) Protein Struct. **405**(2), 442–451 (1975)
75. Mayr, E.W., Meyer, A.R.: The complexity of the word problems for commutative semigroups and polynomial ideals. Adv. Math. **46**(3), 305–329 (1982)
76. McCallum, S.: An improved projection operation for cylindrical algebraic decomposition. In: Caviness, B., Johnson, J. (eds.) Quantifier Elimination and Cylindrical Algebraic Decomposition, Texts and Monographs in Symbolic Computation, pp. 242–268. Springer, Berlin (1998)
77. McCallum, S.: On projection in CAD-based quantifier elimination with equational constraint. In: Proceedings of the 1999 International Symposium on Symbolic and Algebraic Computation, ISSAC '99, pp. 145–149. ACM (1999)
78. McCallum, S., Hong, H.: On using Lazard's projection in CAD construction. J. Symb. Comput. **72**, 65–81 (2016)
79. McCulloch, W.S., Pitts, W.: A logical calculus of the ideas immanent in nervous activity. Bull. Math. Biophys. **5**(4), 115–133 (1943)
80. Mulligan, C.B.: Automated economic reasoning with quantifier elimination. Working Paper 22922, National Bureau of Economic Research (2016)
81. Patel, B.R., Kaushik, K.R.: A survey on decision tree algorithm for classification. Int. J. Eng. Dev. Res. **2**, 1–5 (2014)
82. Paulson, L.C.: Metitarski: past and future. In: Beringer, L., Felty, A. (eds.) Interactive Theorem Proving, Volume 7406 of Lecture Notes in Computer Science, pp. 1–10. Springer, Berlin (2012)
83. Platzer, A., Quesel, J.D.: KeYmaera: a hybrid theorem prover for hybrid systems (system description). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) Automated Reasoning, Volume 5195 of Lecture Notes in Computer Science, pp. 171–178. Springer, Berlin (2008)

84. Platzer, A., Quesel, J.D., Rümmer, P.: Real world verification. In: Schmidt, R.A. (ed.) Automated Deduction (CADE-22), Volume 5663 of Lecture Notes in Computer Science, pp. 485–501. Springer, Berlin (2009)
85. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical Recipes in C (2nd Ed.): The Art of Scientific Computing. Cambridge University Press, Cambridge (1992)
86. Quinlan, J.R.: Induction of decision trees. Mach. Learn. **1**(1), 81–106 (1986)
87. Rosenblatt, F.: The perceptron: a probabilistic model for information storage and organization in the brain. Psychol. Rev. **65**(6), 386 (1958)
88. Schölkopf, B., Tsuda, K., Vert, J.-P.: Kernel methods in computational biology. MIT Press, Cambridge (2004)
89. Sebastiani, F.: Machine learning in automated text categorization. ACM Comput. Surv. **34**(1), 1–47 (2002)
90. Shannon, Claude E.: A mathematical theory of communication. Mob. Comput. Commun. Rev. **5**(1), 3–55 (2001)
91. Shawe-Taylor, J., Cristianini, N.: Kernel Methods for Pattern Analysis. Cambridge University Press, Cambridge (2004)
92. Stone, P., Veloso, M.: Multiagent systems: a survey from a machine learning perspective. Auton. Robot. **8**(3), 345–383 (2000)
93. Strzeboński, A.: Cylindrical algebraic decomposition using validated numerics. J. Symb. Comput. **41**(9), 1021–1038 (2006)
94. Strzeboński, A.: Cylindrical algebraic decomposition using local projections. J. Symb. Comput. **76**, 36–64 (2016)
95. Tarski, A.: A Decision Method for Elementary Algebra and Geometry. RAND Corporation, Santa Monica, CA (reprinted in the collection [28]) (1948)
96. Vapnik, V.N., Chervonenkis, A.Y.: A note on one class of perceptrons. Autom. Remote Control **25**(1), 821–837 (1964)
97. Wilson, D., Bradford, R., Davenport, J.H., England, M.: Cylindrical algebraic sub-decompositions. Math. Comput. Sci. **8**, 263–288 (2014)
98. Wilson, D., Davenport, J.H., England, M., Bradford, R.: A "piano movers" problem reformulated. In: 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC '13, pp. 53–60. IEEE (2013)
99. Wilson, D., England, M., Davenport, J.H., Bradford, R.: Using the distribution of cells by dimension in a cylindrical algebraic decomposition. In: 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC '14, pp. 53–60. IEEE (2014)
100. Wilson, D.J., Bradford, R.J., Davenport, J.H.: A repository for CAD examples. ACM Commun. Comput. Algebra **46**(3), 67–69 (2012)
101. Wilson, D.J., Bradford, R.J., Davenport, J.H.: Speeding up cylindrical algebraic decomposition by Gröbner bases. In: Jeuring, J., Campbell, J.A., Carette, J., Reis, G., Sojka, P., Wenzel, M., Sorge, V. (eds.) Intelligent Computer Mathematics, Volume 7362 of Lecture Notes in Computer Science, pp. 280–294. Springer, Berlin (2012)