Comparing machine learning models to choose the variable ordering for cylindrical algebraic decomposition

Matthew England and Dorian Florescu

Faculty of Engineering, Environment and Computing,
Coventry University, Coventry CV1 5FB, UK
{Matthew.England, Dorian.Florescu}@coventry.ac.uk

Abstract. There has been recent interest in the use of machine learning (ML) approaches within mathematical software to make choices that impact on the computing performance without affecting the mathematical correctness of the result. We address the problem of selecting the variable ordering for cylindrical algebraic decomposition (CAD), an important algorithm in Symbolic Computation. Prior work to apply ML on this problem implemented a Support Vector Machine (SVM) to select between three existing human-made heuristics, which did better than anyone heuristic alone. The present work extends to have ML select the variable ordering directly, and to try a wider variety of ML techniques. We experimented with the NLSAT dataset and the Regular Chains Library CAD function for Maple 2018. For each problem, the variable ordering leading to the shortest computing time was selected as the target class for ML. Features were generated from the polynomial input and used to train the following ML models: k-nearest neighbours (KNN) classifier, multi-layer perceptron (MLP), decision tree (DT) and SVM, as implemented in the Python scikit-learn package. We also compared these with the two leading human constructed heuristics for the problem: Brown’s heuristic and sotd. On this dataset all of the ML approaches outperformed the human made heuristics, some by a large margin.

Keywords: computer algebra; symbolic computation; non-linear real arithmetic; cylindrical algebraic decomposition; machine learning.

1 Introduction

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, “∃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. QE is one definition for simplifying or solving a problem. The tools involved fall within the field of Symbolic Computation, implemented in Computer Algebra Systems (or more recently in SMT-solvers).
When the logical statements are expressed as Tarski formulae, Boolean combinations ($\land$, $\lor$, $\neg$, $\rightarrow$) of statements about the signs of polynomials with integer coefficients, then the QE problem is soluble \[52\]. However, the only implemented general QE procedure has algorithmic complexity doubly exponential in the number of variables \[22\], a theoretical result experienced in practice. For many problem classes QE 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 QE procedures and the formulation of problems, to “push the doubly exponential wall back” and open up a wider range of tractable applications.

QE procedures can be run in multiple ways to solve a given problem: they can be initialized with different options (e.g. variable ordering \[23\], equational constraint designation \[10\]); tasks can be completed in different orders (e.g. order of constraint analysis \[26\]); and the problem itself may be expressible in different formalisations \[55\]. Changing these settings can have a substantial effect on the computational costs (both time and memory) but does not affect the mathematical correctness of the output. They are thus suitable candidates for machine learning: tools that allow computers to make decisions that are not explicitly programmed, via the statistical analysis of large quantities of data.

We continue in Section 2 by introducing background material on the particular decision we study: the variable ordering for Cylindrical Algebraic Decomposition. Here we also outline prior attempts to solve this problem; and prior applications of machine learning to computer algebra. Then in Section 3 we describe our methodology covering datasets, software, features extracted from the problems, machine learning models tests, and how we test against human made heuristics. We present our results in Section 4 and final thoughts in Section 5.

2 Variable Ordering for CAD

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 variable ordering are either equal or disjoint. The cells are (semi)-algebraic meaning each cell can be described with a finite sequence of polynomial constraints. A CAD is produced to be invariant relative to an input, i.e. truth-invariant for a logical formula (so the formula is either true of false throughout each cell). Such a decomposition can then be used to perform quantifier elimination on the formula by testing a finite set of sample points and constructing quantifier free formula from the semi-algebraic cell descriptions.

CAD was introduced by Collins in 1975 \[19\] and works relative to a set of polynomials. Collins’ CAD produces a decomposition so that each polynomial has constant sign on each cell (thus truth invariant for any formula built with those polynomials). The algorithm first projects the polynomials into smaller and smaller dimensions; and then uses these to lift $\neg$ to incrementally build decompositions of larger and larger spaces according to the polynomials at that level. For full details on the original CAD algorithm see \[3\].
Comparing machine learning models to choose the variable ordering for CAD

QE has numerous applications throughout science and engineering (see for example the survey [51]). New applications are found regularly, such as the derivation of optimal numerical schemes [31], and the validation of economic hypotheses [45], [44]. CAD has also found application independent of QE, such as reasoning with multi-valued functions [21] (where we decompose to see where simplification rules are valid); and biological networks [7], [29] where we decompose to identify regions in parameter space where multi-stationarity can occur.

The definition of cylindricity and both stages of the algorithm are relative to an ordering of the variables. For example, given polynomials in variables ordered as

\[ x_n \succ x_{n-1} \succ \ldots \succ x_2 \succ x_1 \]

we first project away \( x_n \) and so on until we are left with polynomials univariate in \( x_1 \). We then start lifting by decomposing the \( x_1 \)-axis, and then the \((x_1, x_2)\)-plane and so on. The cylindricity condition refers to projections of cells in \( \mathbb{R}^n \) onto a space \((x_1, \ldots, x_m)\) where \( m < n \).

There have been numerous advances to CAD since its inception, for example: on how best to implement the projection [34], [42], [12], [43]; avoiding the need for full CAD [20], [54]; symbolic-numeric lifting schemes [49], [37]; adapting to the Boolean structure in the input [8], [9], [27]; and local projection approaches [15], [50]. However, in all cases, the need for a fixed variable ordering remains.

2.2 Effect of the variable ordering

Depending on the application requirements the variable ordering may be determined, constrained, or entirely free. 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

\[ \exists x \ ax^2 + bx + c = 0 \]

using any CAD which eliminates \( x \) first; giving six possible orderings to choose from. A CAD for the polynomial under ordering \( a \prec b \prec c \) has only 27 cells, but needs 115 for the reverse ordering.

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 \( x > y > a \) or ordering \( y > x > a \).

The choice of variable ordering has been long known to have a great effect on the time and memory use of CAD, and the number of cells in the output. 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 [14].

2.3 Prior work on choosing the variable ordering

Heuristics have been developed to choose a variable ordering, with Dolzmann et al. [23] giving the best known study. After analysing a variety of metrics they proposed a polynomial degree based heuristic (the heuristic \texttt{sotd} defined later). However the second author: demonstrated examples for which that heuristic could be misled in CICM 2013 [10]; showed that tailoring it to an implementation
could improve its performance in ICMS 2014 \[28\]; and in CICM 2014 \[36\] reported that a computationally cheaper heuristic by Brown actually outperforms sotd. In CICM 2014 \[36\] we used a support vector machine to choose which of three human made heuristics to believe when picking the variable ordering. The experiments in \[36\] identified substantial subclasses on which each of the three heuristics made the best decision, and demonstrated that the machine learned choice did significantly better than any one heuristic overall. This motivated the present study where we consider a wider range of machine learning models and have these pick the ordering directly from the full range of choices.

2.4 Other applications of ML to mathematical software

The CICM 2014 paper was the first to document the application of machine learning to CAD, or in fact to any symbolic computation algorithm / computer algebra system. Since then there have been two further studies:

- The same authors studied a different choice related to CAD (whether to precondition the input with Gröbner bases) in \[35\], again finding that a support vector machine could make the choice more accurately than the human made heuristic (if features of the Gröbner Basis could be used).
- At MACIS 2016 there was a study applying a support vector machine to decide the order of sub-formulae solving for a QE procedure \[39\].

The survey paper \[25\] and the ICMS Special Session on Machine Learning for Mathematical Software it was written for demonstrated the wide range of other potential applications. As discussed there, while the use of machine learning in computer algebra is rare it has become a key tool in other mathematical software development. Most notably automated reasoning \[53\], \[40\], \[11\], \[2\]; but also satisfiability checking \[58\], \[41\].

3 Methodology

3.1 Dataset

Despite its long history and significant software contributions the Computer Algebra community had a lack of substantial datasets \[33\]: a significant barrier to machine learning. Despite efforts to address this\footnote{E.g. the PoSSo and FRISCO projects in the 90s and the SymbolicData Project \[32\]} but the most substantial dataset of problems designed for CAD is \[57\] with less than 100 examples.

However, CAD has recently found prominence in a new area: Satisfiability Modulo Theories (SMT). Here, efficient algorithms for the Boolean SAT problem study the Boolean skeleton of a problem, with a theory solver then checking if a satisfying Boolean assignment is valid in the domain (learning new clauses if not) \[4\]. For the SMT domain of Non-linear Real Arithmetic (NRA), CAD can play the role of such theory solvers\footnote{However, as discussed by \[1\] a more custom approach is beneficial.} and so their test problems may be used to
evaluate CAD. We use the nlsat dataset produced to evaluate the work in 38. The main sources of the examples are: METiTARSKI [46], an automatic theorem prover for theorems with real-valued special functions (it applies real polynomial bounds and then using QE tools like CAD); problems originating from attempts to prove termination of term-rewrite systems; verification conditions from Keymaera [48]; and parametrized generalizations of geometric problems.

The problems in the nlsat dataset are all fully existential (the only quantifier is $\exists$) which is why they may be studied by SAT solvers. Although CAD can make adaptations based on the quantifiers in the input (most notably via Partial CAD [20]) the conclusions drawn are likely to be applicable outside of the SAT context.

We extracted 6117 problems with 3 variables from this database, meaning each has a choice of six different variable orderings. We randomly divided into two datasets for training (4612) and testing (1505). The training dataset was used to tune the parameters of the ML models. The testing dataset was unknown to the models during training, and is used to compare the performance of the different ML models, and to compare with the human constructed heuristics.

3.2 Software

We experimented using the CAD routine CylindricalAlgebraicDecompose which is part of the RegularChains Library for MAPLE. This algorithm builds decompositions first of $n$-dimensional complex space before refining to a CAD of $\mathbb{R}^n$ [18], [17], [6]. We ran the code in Maple 2018 but used an updated version of the RegularChains Library downloaded from http://www.regularchains.org which contains bug fixes and additional functionality. We ignored the quantifiers and logical connectives, using only the polynomials as input to CAD. The function thus returned a sign-invariant CAD for the polynomials.

The training and evaluation of the machine learning models was done using the scikit-learn package [47] v0.20.2 for Python 2.7. The features for machine learning were extracted using code written in the sympy package v1.3 for Python 2.7, as was Brown’s heuristic. The sotd heuristic was implemented in MAPLE as part of the ProjectionCAD package [30].

3.3 Timings

CAD construction was timed in a Maple script that was called using the os package in Python to avoid Maple caching of results.

The target variable ordering for ML was defined as the one that minimizes the computing time for a given SAT problem. All CAD function calls included a time limit. The problems in the training dataset were processed with an initial time limit of 4 seconds on all variable orderings. The time limit was the doubled if all 6 orderings timed out. The process stopped when the CAD routine was completed for at least one ordering. All problems in the training dataset could be assigned a target variable ordering using time limits smaller than 64 seconds.

3 [http://cs.nyu.edu/~dejan/nonlinear/]
The problems in the testing dataset were processed with a larger time limit of 128 seconds for all orderings. This was in order to allow a better comparison of the computing times for the ML and the heuristics. When a variable ordering timed out, the computing time was considered equal to 128 seconds.

3.4 Features

We computed the same set of 11 features for each SAT problem as [36], which are listed in Table 1. All these features could be computed from the input polynomials immediately. A possibility for future work is to consider features that are more expensive, such as those from post-processing as in [35], those from the end of CAD projection as sotd does, or perhaps even going further into partial lifting as in [56].

The ML models associate a predicted variable ordering to each set of 11 features. The training and testing feature set were normalised using the mean and standard deviation of the training set.

Table 1. The features used by ML to predict variable orderings.

| 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.      |

3.5 ML models

Four of the most commonly used deterministic ML models were tuned on the training data (for details on the methods see for example the textbook [5]):

- The K–Nearest Neighbours (KNN) classifier [5 §2.5].
- The Multi-Layer Perceptron (MLP) classifier [5 §2.5].
- The Decision Tree (DT) classifier [5 §14.4].
- The Support Vector Machine (SVM) classifier with RBF kernel [5 §6.3].

The KNN classifier is a type of instance-based classifier, i.e. it does not construct an internal model of the data but stores all the instances in the training data for prediction. So for each new data instance the model selects the nearest $k$
Comparing machine learning models to choose the variable ordering for CAD

training instances. Selection can be performed by weighting instances equally, or by weighting a training instance inversely proportional to its distance from the new instance. The prediction is given by the class with the highest count among those training instances. Three algorithms are typically used to train KNN: the \textit{brute force} algorithm computes the distances between all pairs of points, the \textit{k-dimensional tree} algorithm partitions the data along Cartesian axes, and the \textit{ball tree} algorithm partitions the data in a series of nesting hyper-spheres.

The DT is a non-parametric model that uses a tree-like model of decisions and their possible consequences. Each node in the tree represents a test on an attribute, each branch the outcome of the test. The \textit{leaves} are the end points of each branch, representing the predicted class label. There are two common criteria used to assess the quality of a split in the DT. The Gini impurity criterion verifies how often a randomly chosen element would be correctly labelled if it were randomly labeled according to the distribution of labels in the subset. The entropy criterion assesses the information gain after each split.

The SVM is a model that can perform linear and non-linear classification, by selecting an appropriate kernel. It is also known as a maximum-margin classifier, because it identifies a hyperplane in the feature space that maximises the distance to the nearest data point. The SVM is trained using a kernel, which acts as a similarity function between the training examples and a few predefined \textit{landmarks}, that can offer additional computing performance. The most common kernels are: \textit{linear}, \textit{polynomial}, \textit{sigmoidal} and \textit{radial basis function} (RBF). The RBF kernel is one of the most common kernel choices, given by

$$K(x, \ell) = e^{-\gamma \|x - \ell\|^2},$$

where $\gamma$ is the kernel hyperparameter, and $\ell \in \mathbb{R}^n$ is a predefined \textit{landmark}, and $x \in \mathbb{R}^n$ is the training vector with $n$ features.

The MLP is a class of feedforward artificial neural networks. It consists of a minimum of 3 layers: the input, hidden and output layer. Both the hidden and output layers use a nonlinear activation function that can be selected during cross-validation. Some of the common activation functions are: identity, logistic, hyperbolic tangent and rectified linear.

3.6 Training

Each model was trained using grid search 5-fold cross-validation. Specifically, the training feature set was randomly divided in 5 equal parts. Each combination of 4 parts was used to tune the model parameters, leaving the last part for fitting the hyperparameters through cross-validation. For each of the models, the grid search was performed for an initially large range for each hyperparameter. This range was further increased until all optimal hyperparameters were selected by the grid search to be inside the range, and not on the edge. Subsequently, the range was slowly decreased to home in on the optimal hyperparameter, until the performance plateaued. Each grid search lasted from a few seconds for simpler models such as KNN to a few minutes for more complex models such as MLP. The optimal hyperparameters selected during cross-validation are in Table 2.
Table 2. The hyperparameters of the ML models optimised with 5-fold cross-validation on the training dataset.

| Decision Tree | Value |
|---------------|-------|
| Hyperparameter |       |
| Criterion      | Gini impurity |
| Maximum tree depth | 17   |

| K-Nearest Neighbours | Value |
|----------------------|-------|
| Hyperparameter       |       |
| Train instances weighting | Inversely proportional to distance |
| Algorithm            | Ball Tree |

| Support Vector Machine | Value |
|------------------------|-------|
| Hyperparameter         |       |
| Regularization parameter C | 316   |
| Kernel                 | Radial basis function |
| γ                      | 0.08  |
| Tolerance for stopping criterion | 0.0316 |

| Multi-Layer Perceptron | Value |
|------------------------|-------|
| Hyperparameter         |       |
| Hidden layer size      | 18    |
| Activation function    | Hyperbolic tangent |
| Algorithm              | Quasi-Newton based optimiser |
| Regularization parameter α | 5 \cdot 10^{-5} |

3.7 Comparing with human made heuristics

The ML approaches were compared in terms of prediction accuracy and resulting CAD computing time against the two best known human constructed heuristics.

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 only in the notes of an ISSAC conference.

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 was the recommendation made after the study [23].

[23] https://www.usna.edu/Users/cs/wcbrown/research/ISSAC04/handout.pdf
Comparing machine learning models to choose the variable ordering for CAD

Unlike the ML, these human made heuristics can end up predicting several variable orderings (i.e. when they cannot discriminate). In practice if this were to happen the heuristic would select one randomly (or perhaps lexicographically), however that final pick is not particularly meaningful for an evaluation. To accommodate this, for each problem, the prediction accuracy of such a heuristic is judged to be the the percentage of its predicted variable orderings that are also target orderings. The average of this percentage over all problems in the testing dataset represents the prediction accuracy. Similarly, the computing time for such methods was assessed as the average computing time over all predicted orderings, and it is this that is summed up for all problems in the testing dataset.

4 Results

We compare the four ML models on the percentage of problems where they selected the optimum ordering, and the total computation time (in seconds) for solving all the problems with their chosen orderings. We also compare the ML models with the two human constructed heuristics (with the adaptations outlined in Section 3.7) and finally the outcome of a random choice between the 6 orderings. The results are presented in Table 3. We might expect a random choice to be correct one sixth of the time for this data set (16.6%). The actual accuracy is a little higher because the dataset is not uniform, and for some problem instances there were multiple variable orderings with equally fast timings.

Moreover, we evaluate the distribution of the computation time for the ML methods and the heuristics. The differences between the computation time of each method and the minimum computation time, given as a percentage of the minimum time, are depicted in Figure 1.

4.1 Range of possible outcomes

We note that the minimum total computing time achieved by selecting an optimal variable ordering for every problem would be 8,623s. Choosing at random would take 30,235 seconds, almost 4 times as much. The maximum total computing time, determined by selecting the variable ordering with the longest computing time, is 64,534s. The choices with the quickest time among the methods considered were achieved by the Decision Tree model: 9,994s, which is 16% more than the minimal possible. So there are clearly great savings to be made by taking this choice into account.

Table 3. The comparative performance of DT, KNN, MLP, SVM, and the Brown and std heuristics on the testing dataset.

|                  | DT   | KNN  | MLP  | SVM  | Brown | std   | random |
|------------------|------|------|------|------|-------|-------|--------|
| Accuracy         | 62.6%| 63.3%| 61.6%| 58.8%| 51%   | 49.5% | 22.7%  |
| Computation Time (s) | 9,994| 10,105| 9,822| 10,725| 10,951| 11,938| 30,235 |
4.2 Human-made heuristics

Of the two human-made heuristics, Brown performed the best, as it did in [36]. As was noted there this is surprising since the sotd heuristic has access to additional information (not just the input polynomials but also there projection). Unlike the ML models and Brown heuristic obtaining an ordering for a problem instance with sotd is not instantaneous. Generating an ordering for all problems in the testing set took over half an hour. Brown could solve all problems in 10,951s, 27% more than the minimum. While sotd is only 0.7% less accurate than Brown in identifying the best ordering, it is much slower at 11,938s or 38% more than the minimum. This shows that while Brown is not much better at identifying the best ordering it does a much better job at discarding the worst!

4.3 ML choices

The results show that all ML approaches outperform the human constructed heuristics in terms of both accuracy and timings. Figure 1 shows that the human-
Comparing machine learning models to choose the variable ordering for CAD

made heuristics result in computing times that are often significantly larger than 1% of the corresponding minimum time for each problem. The ML methods, on the other hand, all result in almost 1000 problems (∼ 75% of the testing dataset) within 1% of the minimum time.

The key finding of the present paper is that there are significantly better models to use for the problem than SVM: each of the other three had 10% higher accuracy and much lower timings. It is interesting to note that the MLP model leads to a lower accuracy than DT and KNN, but achieves the lowest overall computing time. This suggests that on this dataset, the MLP model is best at leaving out variable orderings that lead to long computing times, even if it has a slightly higher chance of missing the target variable ordering.

4.4 Conclusion

For a user of computer algebra the key consideration is that their computation not hang, rather than it finish as quickly as possible on the widest variety of questions. Hence of the tested models we would recommend the MLP for use going forward. However, we emphasise that this conclusion applies only for present dataset and we need wider experimentation to see if this finding is replicated.

5 Final Thoughts

The experiment shows a clear advance on the prior state of the art: both in terms of accuracy of predictions for CAD variable ordering; and in understanding which models are the best suited for this problem. But we acknowledge that there is much more to do and emphasise that this is only the initial findings of the project. The following extensions will be undertaken shortly:

– Extending the experiment to include problems with more variables from the dataset. Unlike some CAD implementations the one used does not change algorithm when \( n > 3 \); however, like all CAD implementation the time required will increase exponentially.
– Using the trained classifiers to test on CAD problems from outside the dataset, in particular, those which are not fully existentially quantified (SAT problems). In such cases the algorithms can change (terminate early) and it is not clear if models trained for the SAT case can be applied there.
– Using the trained classifiers to test on CAD implementations other than the one in the MAPLE RegularChains Library. For example, can the classifiers also usually pick variable orderings for QEPCAD-B [13] or Redlog [24]?

Finally, we note the wide variety of additional problems in computer algebra systems open to study with machine learning [25].

Acknowledgements

The authors are supported by EPSRC Project EP/R019622/1: Embedding Machine Learning within Quantifier Elimination Procedures.
References

1. E. Ábrahám, J. Abbott, B. Becker, A.M. Bigatti, M. Brain, B. Buchberger, A. Cimatti, J.H. Davenport, M. England, P. Fontaine, S. Forrest, A. Griggio, D. Kroening, W.M. Seiler, and T. Sturm. SC²: Satisfiability checking meets symbolic computation. In M. Kohlhase, M. Johansson, B. Miller, L. de Moura, and F. Tompa, eds, Intelligent Computer Mathematics: Proc. CICM 2016, vol. 9791 of Lecture Notes in Computer Science, pages 28–43. Springer Intl., 2016.

2. A.A. Alemi, F. Chollet, N. Een, G. Irving, C. Szegedy, and J. Urban. Deepmath - Deep sequence models for premise selection. In Proc. of the 30th International Conference on Neural Information Processing Systems, NIPS’16, pages 2243–2251. Curran Associates Inc., 2016.

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

4. C. Barrett, R. Sebastiani, S.A. Seshia, and C. Tinelli. Satisfiability modulo theories. In A. Biere, M. Heule, H. van Maaren, and T. Walsh, eds, Handbook of Satisfiability (Vol. 185 Frontiers in Artificial Intelligence and Applications), Chapter 26, pages 825–885. IOS Press, 2009.

5. C.M. Bishop. Pattern Recognition and Machine Learning. Springer, 2006.

6. R. Bradford, C. Chen, J.H. Davenport, M. England, M. Moreno Maza, and D. Wilson. Truth table invariant cylindrical algebraic decomposition by regular chains. In V.P. Gerdt, W. Koepf, W.M. Seiler, and E.V. Vorozhtsov, eds, Computer Algebra in Scientific Computing, vol. 8660 of Lecture Notes in Computer Science, pages 44–58. Springer Intl., 2014.

7. R. Bradford, J.H. Davenport, M. England, H. Errami, V. Gerdt, D. Grigoriev, C. Hoyt, M. Kosta, O. Radulescu, T. Sturmf, and A. Weber. A case study on the parametric occurrence of multiple steady states. In Proc. of the 2017 ACM International Symposium on Symbolic and Algebraic Computation, ISSAC ’17, pages 45–52. ACM, 2017.

8. R. Bradford, J.H. Davenport, M. England, S. McCallum, and D. Wilson. Cylindrical algebraic decompositions for boolean combinations. In Proc. of the 38th International Symposium on Symbolic and Algebraic Computation, ISSAC ’13, pages 125–132. ACM, 2013.

9. R. Bradford, J.H. Davenport, M. England, S. McCallum, and D. Wilson. Truth table invariant cylindrical algebraic decomposition. Journal of Symbolic Computation, 76:1–35, 2016.

10. R. Bradford, J.H. Davenport, M. England, and D. Wilson. Optimising problem formulations for cylindrical algebraic decomposition. In J. Carette, D. Aspinall, C. Lange, P. Sojka, and W. Windsteiger, eds, Intelligent Computer Mathematics, vol. 7961 of Lecture Notes in Computer Science, pages 19–34. Springer Berlin Heidelberg, 2013.

11. J.P. Bridge, S.B. Holden, and L.C. Paulson. Machine learning for first-order theorem proving. Journal of Automated Reasoning, 53:141–172, 2014.

12. C.W. Brown. Improved projection for cylindrical algebraic decomposition. Journal of Symbolic Computation, 32(5):447–465, 2001.

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

14. C.W. Brown and J.H. Davenport. The complexity of quantifier elimination and cylindrical algebraic decomposition. In Proc. of the 2007 International Symposium on Symbolic and Algebraic Computation, ISSAC ’07, pages 54–60. ACM, 2007.
Comparing machine learning models to choose the variable ordering for CAD

15. C.W. Brown and M. Kosta. Constructing a single cell in cylindrical algebraic decomposition. *Journal of Symbolic Computation*, 70:14 – 48, 2015.
16. B. Caviness and J. Johnson. *Quantifier Elimination and Cylindrical Algebraic Decomposition*. Texts & Mono. Symbolic Computation. Springer-Verlag, 1998.
17. C. Chen and M. Moreno Maza. An incremental algorithm for computing cylindrical algebraic decompositions. In R. Feng, W. Lee, and Y. Sato, eds, *Computer Mathematics*, pages 199–221. Springer Berlin Heidelberg, 2014.
18. C. Chen, M. Moreno Maza, B. Xia, and L. Yang. Computing cylindrical algebraic decomposition via triangular decomposition. In *Proc. International Symposium on Symbolic and Algebraic Computation*, ISSAC ’09, pages 95–102. ACM, 2009.
19. G.E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Proc. 2nd GI Conference on Automata Theory and Formal Languages*, pages 134–183. Springer-Verlag (reprinted in the collection [16]), 1975.
20. G.E. Collins and H. Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *Journal of Symbolic Computation*, 12:299–328, 1991.
21. J.H. Davenport, R. Bradford, M. England, and D. Wilson. Program verification in the presence of complex numbers, functions with branch cuts etc. In *14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, SYNASC ’12, pages 83–88. IEEE, 2012.
22. J.H. Davenport and J. Heintz. Real quantifier elimination is doubly exponential. *Journal of Symbolic Computation*, 5(1-2):29–35, 1988.
23. A. Dolzmann, A. Seidl, and T. Sturm. Efficient projection orders for CAD. In *Proc. of the 2004 International Symposium on Symbolic and Algebraic Computation*, ISSAC ’04, pages 111–118. ACM, 2004.
24. A. Dolzmann and T. Sturm. REDLOG: Computer algebra meets computer logic. *SIGSAM Bull.*, 31(2):2–9, 1997.
25. M. England. Machine learning for mathematical software. In J.H. Davenport, M. Kauers, G. Labahn, and J. Urban, eds, *Mathematical Software – Proc. ICMS 2018*, vol. 10931 of *Lecture Notes in Computer Science*, pages 165–174. Springer Intl., 2018.
26. M. England, R. Bradford, C. Chen, J.H. Davenport, M. Moreno Maza, and D. Wilson. Problem formulation for truth-table invariant cylindrical algebraic decomposition by incremental triangular decomposition. In S.M. Watt, J.H. Davenport, A.P. Sexton, P. Sojka, and J. Urban, eds, *Intelligent Computer Mathematics*, vol. 8543 of *Lecture Notes in Artificial Intelligence*, pages 45–60. Springer Intl., 2014.
27. M. England, R. Bradford, and J.H. Davenport. Improving the use of equational constraints in cylindrical algebraic decomposition. In *Proc. of the 2015 International Symposium on Symbolic and Algebraic Computation*, ISSAC ’15, pages 165–172. ACM, 2015.
28. M. England, R. Bradford, J.H. Davenport, and D. Wilson. Choosing a variable ordering for truth-table invariant cylindrical algebraic decomposition by incremental triangular decomposition. In H. Hong and C. Yap, eds, *Mathematical Software – ICMS 2014*, vol. 8592 of *Lecture Notes in Computer Science*, pages 450–457. Springer Heidelberg, 2014.
29. M. England, H. Errami, D. Grigoriev, O. Radulescu, T. Sturm, and A. Weber. Symbolic versus numerical computation and visualization of parameter regions for multistationarity of biological networks. In V.P. Gerdt, W. Koepf, W.M. Seiler, and E.V. Vorozhtsov, eds, *Computer Algebra in Scientific Computing (CASC)*, vol. 10490 of *Lecture Notes in Computer Science*, pages 93–108. Springer Intl., 2017.
30. M. England, D. Wilson, R. Bradford, and J.H. Davenport. Using the Regular Chains Library to build cylindrical algebraic decompositions by projecting and lifting. In H. Hong and C. Yap, eds, *Mathematical Software – ICMS 2014*, vol. 8592 of *Lecture Notes in Computer Science*, pages 458–465. Springer Heidelberg, 2014.

31. M. Erascu and H. Hong. Real quantifier elimination for the synthesis of optimal numerical algorithms (Case study: Square root computation). *Journal of Symbolic Computation*, 75:110–126, 2016.

32. H.G. Graebe, A. Nareike, and S. Johanning. The SymbolicData project: Towards a computer algebra social network. In M. England, J.H. Davenport, A. Kohlhase, M. Kohlhase, P. Libbrecht, W. Neuper, P. Quaresma, A.P. Sexton, P. Sojka, J. Urban, and S.M. Watt, eds, *Joint Proc. of the MathUI, OpenMath and ThEdu Workshops and Work in Progress track at CICM*, Vol. 1186 in CEUR-WS, 2014.

33. A. Heinle and V. Levandovskyy. The SDEval benchmarking toolkit. *ACM Communications in Computer Algebra*, 49(1):1–9, 2015.

34. H. Hong. An improvement of the projection operator in cylindrical algebraic decomposition. In *Proc. International Symposium on Symbolic and Algebraic Computation*, ISSAC ’90, pages 261–264. ACM, 1990.

35. Z. Huang, M. England, J.H. Davenport, and L. Paulson. 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)*, pages 45–52. IEEE, 2016.

36. Z. Huang, M. England, D. Wilson, J.H. Davenport, L. Paulson, and J. Bridge. Applying machine learning to the problem of choosing a heuristic to select the variable ordering for cylindrical algebraic decomposition. In S.M. Watt, J.H. Davenport, A.P. Sexton, P. Sojka, and J. Urban, eds, *Intelligent Computer Mathematics*, vol. 8543 of *Lecture Notes in Computer Science*, pages 236–251. Springer Intl., 2016.

37. H. Iwane, H. Yanami, H. Anai, and K. Yokoyama. An effective implementation of a symbolic-numeric cylindrical algebraic decomposition for quantifier elimination. In *Proc. Symbolic Numeric Computation*, SNC ’09, pages 55–64. 2009.

38. D. Jovanovic and L. de Moura. Solving non-linear arithmetic. In B. Gramlich, D. Miller, and U. Sattler, eds, *Automated Reasoning: 6th International Joint Conference (IJCAR)*, vol. 7364 of *Lecture Notes in Computer Science*, pages 339–354. Springer, 2012.

39. M. Kobayashi, H. Iwane, T. Matsuzaki, and H. Anai. Efficient subformula orders for real quantifier elimination of non-prenex formulas. In S.I. Kotsireas, M.S. Rump, and K.C. Yap, eds, *Mathematical Aspects of Computer and Information Sciences (MACIS ’15)*, vol. 9582 of *Lecture Notes in Computer Science*, pages 236–251. Springer Intl., 2016.

40. D. Kühnwein, J.C. Blanchette, C. Kaliszyk, and J. Urban. MaSh: Machine learning for sledgehammer. In S. Blazy, C. Paulin-Mohring, and D. Pichardie, eds, *Interactive Theorem Proving*, vol. 7998 of *Lecture Notes in Computer Science*, pages 35–50. Springer Berlin Heidelberg, 2013.

41. J.H. Liang, V.K. Hari Govind, P. Poupart, K. Czarnecki, and V. Ganesh. An empirical study of branching heuristics through the lens of global learning rate. In S. Gaspers and T. Walsh, eds, *Theory and Applications of Satisfiability Testing – SAT 2017*, vol. 10491 of *Lecture Notes in Computer Science*, pages 119–135. Springer Intl., 2017.

42. S. McCallum. An improved projection operation for cylindrical algebraic decomposition. In B. Caviness and J. Johnson, eds, *Quantifier Elimination and Cylindrical
Comparing machine learning models to choose the variable ordering for CAD

43. S. McCallum, A. Parusiński, and L. Paunescu. Validity proof of Lazard’s method for CAD construction. *Journal of Symbolic Computation*, 92:52–69, 2019.

44. C. Mulligan, R. Bradford, J.H. Davenport, M. England, and Z. Tonks. Non-linear real arithmetic benchmarks derived from automated reasoning in economics. In A.M. Bigatti and M. Brain, eds, *Proc. 3rd Workshop on Satisfiability Checking and Symbolic Computation (SC^2 2018)*, Vol. 2189 in CEUR-WS, pages 48–60, 2018.

45. C.B. Mulligan, J.H. Davenport, and M. England. TheoryGuru: A Mathematica package to apply quantifier elimination technology to economics. In J.H. Davenport, M. Kauers, G. Labahn, and J. Urban, eds, *Mathematical Software – Proc. ICMS 2018*, vol. 10931 of *Lecture Notes in Computer Science*, pages 369–378. Springer Intl., 2018.

46. L.C. Paulson. Metitarski: Past and future. In L. Beringer and A. Felty, eds, *Interactive Theorem Proving*, vol. 7406 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2012.

47. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

48. A. Platzer, J.D. Quesel, and P. Rümmer. Real world verification. In R.A. Schmidt, editor, *Automated Deduction (CADE-22)*, vol. 5663 of *Lecture Notes in Computer Science*, pages 485–501. Springer Berlin Heidelberg, 2009.

49. A. Strzeboński. Cylindrical algebraic decomposition using validated numerics. *Journal of Symbolic Computation*, 41(9):1021–1038, 2006.

50. A. Strzeboński. Cylindrical algebraic decomposition using local projections. *Journal of Symbolic Computation*, 76:36–64, 2016.

51. T. Sturm. New domains for applied quantifier elimination. In V.G. Ganzha, E.W. Mayr, and E.V. Vorozhtsov, eds, *Computer Algebra in Scientific Computing*, vol. 4194 of *Lecture Notes in Computer Science*, pages 295–301. Springer Berlin Heidelberg, 2006.

52. A. Tarski. *A Decision Method For Elementary Algebra And Geometry*. RAND Corporation, Santa Monica, CA (reprinted in the collection [14]), 1948.

53. J. Urban. MaLARea: A metasystem for automated reasoning in large theories. In *Empirically Successful Automated Reasoning in Large Theories (ESARLT ’07)*, vol. 257 of *CEUR-WS*, page 14, 2007.

54. D. Wilson, R. Bradford, J.H. Davenport, and M. England. Cylindrical algebraic sub-decompositions. *Mathematics in Computer Science*, 8:263–288, 2014.

55. D. Wilson, J.H. Davenport, M. England, and R. Bradford. A “piano movers” problem reformulated. In *15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, SYNASC ’13, pages 53–60. IEEE, 2013.

56. D. Wilson, M. England, J.H. Davenport, and R. Bradford. 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, pages 53–60. IEEE, 2014.

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

58. L. Xu, F. Hutter, H.H. Hoos, and K. Leyton-Brown. SATzilla: Portfolio-based algorithm selection for SAT. *J. A.I. Research*, 32:565–606, 2008.