Explainable AI Insights for Symbolic Computation: A case study on selecting the variable ordering for cylindrical algebraic decomposition

Lynn Pickering [a], Tereso del Río Almajano [b],
Matthew England [b], Kelly Cohen [a]

[a] *Aerospace Engineering and Engineering Mechanics, University of Cincinnati, Cincinnati, 45221, OH, USA*
[b] *Research Centre for Computational Science and Mathematical Modelling, Coventry University, Coventry, CV1 5FB, West Midlands, UK*

A R T I C L E   I N F O

A B S T R A C T

In recent years there has been increased use of machine learning (ML) techniques within mathematics, including symbolic computation where it may be applied safely to optimise or select algorithms. This paper explores whether using explainable AI (XAI) techniques on such ML models can offer new insight for symbolic computation, inspiring new implementations within computer algebra systems that do not directly call upon AI tools. We present a case study on the use of ML to select the variable ordering for cylindrical algebraic decomposition. It has already been demonstrated that ML can make the choice well, but here we show how the SHAP tool for explainability can be used to inform new heuristics of a size and complexity similar to those human-designed heuristics currently commonly used in symbolic computation.

1. Introduction

1.1. Machine learning and mathematics

Machine Learning (ML) refers to statistical techniques that learn rules from data. They allow computer systems to improve their performance on a task without any change to their explicit programming. ML underpins the prominent AI advances of recent years, driven forward by the growth in available data and computing power, new ML architectures, and hardware designed specifically for them.

There are increasing attempts to use ML in mathematics. For example, Lample and Charton (2020) trained a transformer to integrate expressions and find analytical solutions to differential equations; while in a survey article He (2022) details a variety of attempts to use ML to predict properties from mathematical structures such as groups and graphs. We note a recent special issue of this Journal of Symbolic Computation dedicated to *Algebraic Geometry and Machine Learning* (Hauenstein et al., 2023) including ML approaches to e.g. find the real discriminant locus (Bernal et al., 2023). However, it has been observed that mathematical and logical reasoning is an area of particular difficulty for ML, especially the recent natural language tools such as ChatGPT.[1]

Symbolic computation is not an obvious domain in which to apply ML. The Computer Algebra Systems (CASs) which implement symbolic computation algorithms are mainly designed to produce exact answers; rarely using numeric computation in preference for symbolic solutions. Lample and Charton (2020) reported that their Transformer[2] was able to integrate far more examples correctly within a time limit than various CASs. However, this analysis combined two very different cases of failure: when a solution could not be found within a time limit and when the wrong solution is found. While the CASs may not solve problems quickly, they should *never* produce the wrong answer. Thus there is little appetite amongst CAS developers to use ML for directly producing output.

1.2. Safe use of machine learning in computer algebra systems

This does not mean that ML has nothing to offer Symbolic Computation. CAS algorithms often come with choices that have no effect on the mathematical correctness of the end result but can have a big impact on the resources required. Further, they can affect how the end result is presented (i.e. two very different, but mathematically equivalent, expressions). Consider for example the algorithm of Buchberger (2006) to produce a Gröbner Basis for an ideal. It does not specify the order in which S-pairs are computed, the order in which the corresponding S-polynomial is reduced by the generating set, the monomial ordering to be used, nor the underlying variable ordering. Any decision for these choices allows the production of a correct Gröbner Basis but each decision affects the size of the basis produced, the polynomials within, and the time taken to compute them.

Sometimes there are well-documented strategies on how CASs make such choices (e.g. sugar selection for S-pairs (Giovini et al., 1991)) but in many cases, they rely on human-made heuristics or "*magic constants*" (Carette, 2004) which are often not scientifically validated or even documented. Such choices are good candidates for ML: the underlying relationships are complex but not the main object of study. The earliest examples of ML for CAS optimisation known to the authors are: Huang et al. (2014) using a support vector machine to choose the variable ordering for cylindrical algebraic decomposition; Kuipers et al. (2015) using a Monte-Carlo tree search to find the representation of polynomials that are most efficient to evaluate; Simpson et al. (2016) using ML classifiers to pick from various algorithms that compute the resultant; Huang et al. (2016) using ML to decide whether to precondition input for cylindrical algebraic decomposition; and Kobayashi et al. (2016) using ML to decide the order of sub-formulae solving for real quantifier elimination.

---

[1] See e.g. this blog post by S. Wolfram: https://writings.stephenwolfram.com/2023/01/wolframalpha-as-the-way-to-bring-computational-knowledge-superpowers-to-chatgpt/.

[2] A particular design of ML model that has found great success in natural language processing tasks: it is used for example by ChatGPT.

In all of these examples, the ML solution outperformed the previous human-made heuristics for the choices, at least on the dataset used. The algorithms of symbolic computation are often exponential in their worst-case complexity but the average case complexity is less widely studied. It is possible that these ML optimisations may offer a route to avoid the worst case, at least for many applied examples. Thus there is great value in their continued study and exposition. However, the motivation of the present paper is whether ML can offer anything *more* than these efficiency gains. A recent article in Nature by Davies et al. (2021) suggested that ML can help pure mathematicians with the development of new theorems. Can ML offer such insight to the developers of symbolic computation algorithms?

1.3. Explainable artificial intelligence

Explainable AI (XAI) refers to ML techniques which offer an explanation as to how an AI decision was made. It has become a very active area of research, see for example DARPA's XAI program (Gunning and Aha, 2019). The role of XAI is usually to allow for more effective use of the AI tools, and the building of user trust in the tools. But in our case, we hypothesise that XAI could reveal workings of the ML model that are useful to inform further development of the underlying symbolic computation.

We were motivated by the work of Peifer et al. (2020) developing an ML solution to the problem of selecting S-pairs in Buchberger's algorithm. They trained an "*agent*" (ML model) to make the decision using reinforcement learning[3] based on the number of polynomial additions required to process the choice. This agent outperformed existing strategies such as (Giovini et al., 1991). We draw attention to (Peifer et al., 2020, Section 5.1) which tried to explain the agents' strategy (albeit without employing XAI tools). It identified some simple components such as a preference for pairs whose S-polynomials are monomials and a preference for pairs whose S-polynomials are low degree. Firstly, these components suggested decisions made on the basis of the S-polynomials rather than the S-pairs,[4] which is what all the human-made heuristics consider. Hence this opens a whole new category of heuristics to consider, perhaps leading to a greater understanding of the problem. Secondly, experiments using these strategies alone showed they outperformed the existing heuristics (but not the full ML agent). Thus the analysis suggests new "*human-level*" heuristics, i.e. heuristics of a similar size to those that are human-made and that, once discovered, can be implemented without any AI tools.

The present paper seeks to identify if XAI tools can offer an automated way to make such analyses, to gain new understanding of these heuristic choices in symbolic computation algorithms.

1.4. Contributions and plan of the paper

In this paper, we return to the problem of selecting the variable ordering for cylindrical algebraic decomposition, the first CAS optimisation by ML that was considered in (Huang et al., 2014). We introduce the necessary background information on this problem in Section 2. We then apply the popular SHAP methodology for XAI to an existing ML pipeline developed for the problem: we introduce SHAP in Section 3 and report on the results of its application on this problem in Section 4. In Section 5 we process the SHAP data to suggest some XAI-recommended features for the problem, and in Section 6 we report on the performance of human-level heuristics based upon these. The code and data underpinning these results are openly available from Zenodo here: https://doi.org/10.5281/zenodo.8229298

We find that one of the features recommended by XAI is the same as that used in the current state-of-the-art human-developed heuristic for the problem (del Río and England, 2022), and further, that a human-level combination of XAI selected features allows us to outperform that prior work. Thus our contributions are: (a) a new state-of-the-art human-level heuristic, at least in three variable problems, and perhaps more importantly; (b) evidence for a new methodology to employ in CAS development that uses AI in algorithm design but not implementation.

---

[3] With reinforcement learning, instead of having a labelled dataset to train against, the agent repeatedly makes decisions which receive a score that they seek to optimise.

[4] even though it was properties of the pairs fed to the network.

2. Cylindrical algebraic decomposition

2.1. Cylindrical algebraic decomposition

Cylindrical Algebraic Decomposition (CAD) is an important symbolic computation algorithm, proposed by Collins (1975) in 1975. Given a set of multivariate polynomials and a variable ordering, CAD will decompose the corresponding real space into connected regions, called cells, such that within each the sign of each polynomial is invariant (i.e. positive, negative or zero). Each cell is semi-algebraic, meaning it may be defined by a set of polynomial constraints; and the cells are arranged in cylinders with respect to the variable ordering, meaning the projections of any two cells onto a lower dimensional space with respect to the ordering are either equal or disjoint.

CAD has found applications in various fields, ranging from robotics (Manubens et al., 2012) through economics (Mulligan et al., 2018) to biology (Röst and Sadeghimanesh, 2021). It is most commonly used for, and was proposed by Collins (1975) as, a sub-method to perform real Quantifier Elimination (QE): i.e. to transform a formula in first-order logic whose atoms are Boolean constraints to an equivalent one without any quantifiers. The decomposition into cells allows for studying the problem at a finite number of points, while the cylindrical structure allows for easy projection and negation of the cells to construct the quantifier-free solution.

However, CAD has a worst-case theoretical complexity that grows doubly exponential in the number of variables (Davenport and Heintz, 1988); something regularly encountered in practice and thus vastly reducing the scope of use cases for the algorithm. There have thus been many lines of research to improve CAD, such as improved projection operators (McCallum et al., 2019), partial CAD (Collins and Hong, 1991), equational constraints (England et al., 2020), symbolic-numeric lifting (Strzeboński, 2006), alternative computation schemes via triangular decomposition (Chen et al., 2009) and incremental processing of polynomials (Kremer and Ábrahám, 2020), and entirely new repackaging of the CAD theory (Brown, 2015), (Jovanovic and de Moura, 2012).

2.2. CAD variable ordering

CAD, including all the optimised and extended forms listed above, requires there to be a declared ordering on the variables. This controls both the order of operations in the algorithms and the form of the output (being used in the definition of the cylindrical structure). Depending on the application, there may be a free or constrained choice for the ordering. For example, when using CAD for real QE the variables must be ordered as they are quantified: but variables in blocks of the same quantifier (and free variables) can be swapped with each other.

This choice of variable ordering has been shown to have a huge impact, both practically (Dolzmann et al., 2004) and in terms of theoretical complexity: Brown and Davenport (2007) showed there exists a family of polynomial sets such that the complexity of CAD can be constant or doubly exponential depending on the chosen variable ordering. It is possible to appreciate the importance of variable ordering in the simple two-variable example of Fig. 1, where using one of the orderings generates almost twenty times more cells than using the other.

2.3. Prior human-designed heuristics for choosing the ordering

Once the importance of variable ordering was established, researchers began looking for strategies to choose a good variable ordering. The first attempts consisted of designing human-made heuristics. Brown (2004) documented a heuristic based on three simple hand-picked features of the set of polynomials, for the software QEPCAD. We call this heuristic `Brown`.

There have been other heuristics produced which can offer greater accuracy but at greater expense, performing increasing numbers of steps in the CAD algorithm: Dolzmann et al. (2004) concluded it is best to perform the projection stage of CAD and compare sums of the total degree of the polynomials produced (`sotd`); Bradford et al. (2013) considered the initial decomposition of the real line; and Wilson et al. (2015) the open cells in the decomposition. It is perhaps surprising that the original heuristic of Brown is able to perform competitively or even beat these others despite having access
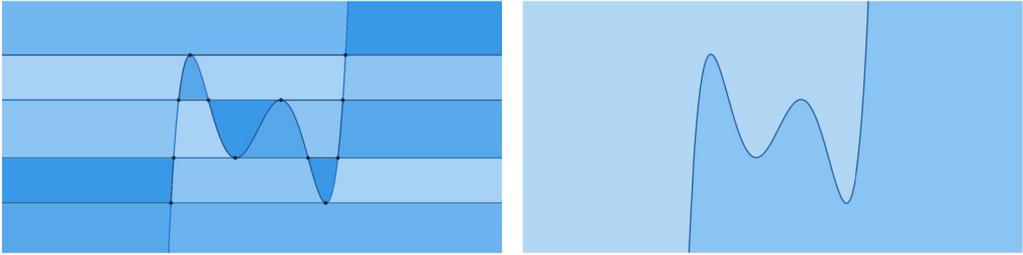
Fig. 1. CADs sign-invariant for the set of polynomials $\{x^5 + 5x^4 + 5x^3 - 5x^2 - 6x - 2y\}$. Using ordering $x \succ y$, we obtain a CAD with 57 cells (18 shaded areas, 27 curve segments and 12 points). Using the ordering $y \succ x$ generates only 3 cells (2 areas and one curve).

to less information (Huang et al., 2019). This suggests the necessary information to identify a good ordering may be available from the input alone.

Most recently, del Río and England (2022) designed a new simple heuristic `gmods` based on properties of the input polynomials, selected by studying which features of the input polynomials have the greatest impact on the complexity analysis of CAD.

2.4. Prior AI-designed heuristics for choosing the ordering

In the last decade machine learning models have also been trained to select the variable ordering for CAD. The first attempt was made by Huang et al. (2014) in 2014 who used a support vector machine for the task. Later England and Florescu (2019) experimented with a wider range of models, methods for feature engineering (Florescu and England, 2019) and improved metrics for training (Florescu and England, 2020a), culminating in a machine learning pipeline available to use for the task (Florescu and England, 2020b). Separately Chen et al. (2020) experimented with deep learning for variable selection.

In all the experiments performed in this paper, these ML models outperform existing heuristics. However, CAS developers may still be reluctant to include a method trained using ML in their software, possibly because they create a dependence on external code, or more likely due to concerns about whether over-fitting on the training data will lead to poor generalisation. Another issue is the lack of explanation these AI models are able to give about their choices.

Later, in Section 4, we will use the infrastructure for a ML selection of the CAD variable ordering presented by Florescu and England (2020b) in our own XAI analysis. In that paper, the authors present a pipeline to train four ML models on the task. When specified to three variables polynomials the pipeline utilises the ideas described by Florescu and England (2019) to represent such polynomials with 81 features (which evaluate to floating point values). It is these numerical feature vectors that are presented to the ML models.

In this paper, we propose an approach that is somewhere in between those of this subsection and the previous one. Rather than handpicking some features as in Section 2.3, we will start with the algorithmically produced features of Section 2.4 but then use XAI techniques to extract the most important features from the AI model and use these to design heuristics that could easily be understood by humans and embedded into any CAD implementation without the subsequent use of ML software.

3. Explaining machine learning models

XAI is often motivated as a means to build trust in the tool among users. But there are certainly other reasons we may require an explanation of an ML model. Perhaps the model is making a poor decision and the data scientist wishes to understand why so as to improve it; or perhaps the explanation tells us more about the underlying trends of data leading to new research directions. The latter is the motivation for explainability in this work. Within the XAI field there is a growing emphasis

on designing models that are intrinsically explainable,[5] however, the most common approach in use today is post-hoc explainability whereby a model is analysed after its creation and an explanation is generated. The review article by Du et al. (2019) offers a good overview of XAI.

Since we work with the existing ML pipeline (Florescu and England, 2020b) already designed for the problem we do not consider editing the ML itself for explainability but instead revert to the more common post-hoc analysis. That pipeline works with several ML models, and so we desire a model-agnostic method for explainability. The following is the current state-of-the-art process for this.

3.1. SHAP

Molnar (2023) gives an overview of many explainability methods along with their advantages and disadvantages. The choice of explainability method will have an impact on what conclusions may be drawn. SHAP (SHapley Additive exPlanations), a model agnostic method, (Lundberg and Lee, 2017) is chosen as our method of post-explainability analysis for identifying the features that have a greater impact on the output of the ML models. The SHAP approach was introduced in 2017 by Lundberg and Lee (2017) as a unification of six existing explanation methods already present in the literature: LIME, DeepLIFT, Layer-Wise Relevance Propagation, Classic Shapley Regression Values, Shapley Sampling Values, and Quantitative Input Influence. Because SHAP unifies these various methods, it can be used for both a global and local explanation of models: i.e. it can offer explanations for performance on an individual problem or an entire dataset. A theorem in Lundberg and Lee (2017) shows that there is only one possible additive feature attribution method (SHAP) that holds local accuracy, missingness (a missing feature must have no impact) and consistency properties.

The name and theory behind SHAP is based on the classical concept of Shapley values from game theory, which quantifies the contribution of each player to a game (Shapley, 1953). It makes use of an explainer model, which is an approximation of the original model. The idea is to figure out the contribution of each player to the score of the game by playing the game with all the combinations of players, or the power set of the players. Take an example game with three players: Player 1, Player 2, and Player 3. Each player plays the game by themselves, then each player plays the game with each of the other two, and finally they all play the game together. We calculate the difference between the final scores of each game where Player 1 was not present, and when Player 1 was present under a weighting to get the contribution of Player 1 to the game. In our application of SHAP, the features of our polynomial sets are the players and the ML models are the games. SHAP removes and adds each feature to the model to see what effect that feature has.

Since we are working over the power set of all the features in the model, such a calculation becomes very expensive quickly: exponential on the number of features in a model (Lundberg and Lee, 2017). However, the SHAP open source package of Lundberg (2022) uses many estimators and powerful shortcuts to make the extensive power set of calculations possible for larger models. For example, Kernel SHAP, used to analyse all the models other than decision tree in this work, uses a special weighted linear regression in computing a feature's importance (Lundberg and Lee, 2017). For the decision tree model, the tree explainer is used. This is a fast method to estimate SHAP values for tree models (Lundberg et al., 2020).

3.2. SHAP waterfall plots

Fig. 2 shows an example of a SHAP waterfall plot: one visualization in SHAP for understanding a model's decision on a single instance. The ML model studied here was trained to decide whether a wine has a quality rating higher than 5 (on a scale of $1 - 10$).

The horizontal axis plots the probability of the wine having quality higher than 5 on the scale. We see that the model predicts this with probability 0.993 for this instance (this particular wine), indicating a strong predicting that it will be ranked in the top half of the quality scale.

---

[5] There are well-established views on which models are intrinsically explainable and an assumed tradeoff against performance with e.g. neural networks offering the best performance but the least explanation. However, in practice this trade-off is not always so clear, see e.g. (Herm et al., 2022).
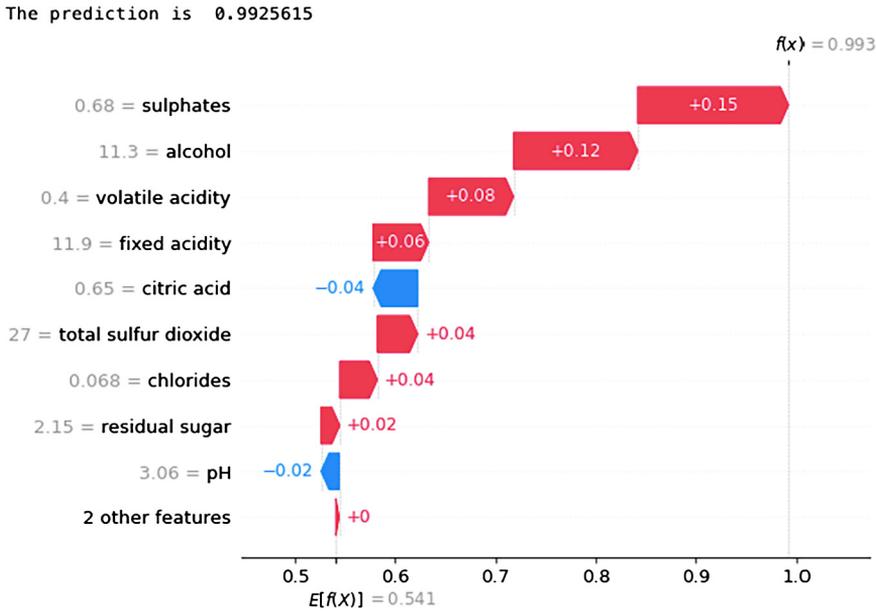
The prediction is 0.9925615

Fig. 2. SHAP waterfall plot reproduced from an online tutorial (https://medium.com/dataman-in-ai/the-shap-with-more-elegant-charts-bc3e73fa1c0c) where it appears as the second image in Section 3.2. The plot explains the prediction by an XGBoost model for whether a particular wine has a rating in the top half of the quality scale, using the "*Red Wine Quality*" Kaggle dataset (https://www.kaggle.com/datasets/uciml/red-wine-quality-cortez-et-al-2009).

The features that contribute to a prediction that a wine has a quality rating strictly over 5 are the red bars, while the features that contribute to a prediction that a wine has a quality rating of 5 or under are the blue bars. The features from top to bottom are those that have the largest effect on the final model output. Thus we see that, for this example, the feature *sulphates* (which is a measure of the presence of this wine additive) is the feature with the most impact on this prediction. The instance from the data set under study had 0.68 value for *sulphates* as indicated in gray on the vertical axis: the value of this feature in the whole dataset ranged from $0.33 - 2$.

Below the horizontal axis we see $E[f(X)]$ which is the base value for this probability. It is the average of the output across all the instances, which would be the best guess a model would have for this dataset if there were no features for the instance to consider. Thus the waterfall plot explains the difference between a particular prediction $f(x)$, and this base prediction $E[f(x)]$ for the dataset, and how each feature contributed to this difference.

3.3. Multi-class SHAP

The ML pipeline we experiment with is designed for multi-classification, i.e. classification with more than two output classes. SHAP may also be applied for multi-classification. It simply creates a binary model for each output, with the binary SHAP model that reaches an output closest to one for an instance giving the prediction. Section 4.2.1 provides a more detailed example of this in practice.

4. SHAP applied to ML for CAD variable ordering

We work with the feature engineering approach to the problem developed by Florescu and England (2019). This algorithmically generates features to represent systems of polynomials to ML models. It was inspired by the earlier hand-picked features based on variable degree and sparsity (Brown, 2004), (Huang et al., 2014) and generalises this to the widest set of such features that could be generated in a similar way. When applied to three variable polynomial systems this framework creates 81 distinct

features, which the pipeline of Florescu and England (2020b) uses as the input to train four ML models (implemented in Python from the scikit-learn package (Pedregosa et al., 2011)) which are: Decision Tree (DT), K-Nearest Neighbors (KNN), Support Vector Machine (SVM), and Multi-Layer Perceptron (MLP).[6]

We fit model hyperparameters using the same cross-validation procedure of Florescu and England (2020b). The DT uses a random splitter with a max depth of 5, and the gini criterion to measure the quality of a split in the tree. The KNN uses the ball tree algorithm to find the 18 nearest neighbors and distance to weight the points in the neighborhood. The SVM has a gamma of 0.08, and the radial basis function kernel. The MLP has logistic activation, hidden layer sizes of 2, and the lbfgs solver.[7] The output that each model predicts is a class $(1-6)$ representing one of the six possible variable orderings of three variables. In these papers, the experiments showed the ML models to all outperform the best performing human-made heuristic of the time, that of Brown (2004). Although Brown's heuristic has since been outperformed by the heuristic `gmods` obtained by del Río and England (2022).

The work described in this section uses SHAP to gain insight into how those ML models of Florescu and England (2020b) achieve this performance. We seek to understand whether the ML models find the most important features to be those selected by humans, which are then supplemented with additional information; or whether they instead rely on some previously overlooked feature. This section first looks at the decision-making process for a single test instance to gain a local understanding of a particular ML model (the MLP). The process from the local decision to the global decision is then explained, and the top five features across the entire model, for each model are identified and discussed.

4.1. Methodology

The methodology used in applying SHAP to ML algorithms for picking the CAD variable ordering is described next, starting with the data set used to train the algorithms, the specific manner in which SHAP was implemented, and details on the features used as input to the ML algorithms.

4.1.1. Dataset

We started by recreating the experiments of Florescu and England (2020b). These used a dataset of random problems generated with the same statistical properties as the industrial problems hosted in the QF_NRA category of the SMT-LIB (Barrett et al., 2016). We identified that the dataset used in Florescu and England (2020b) had a majority of problems with class 5 or 6 as the output class. In this paper, we applied the additional step of balancing the dataset so that the classes are more equal in size. We take advantage of the fact that the nature of a polynomial set does not change with the names of its variables. This allows for the random permutation of variable names (x_1, x_2, x_3), and a corresponding change of the best ordering for the output, to obtain a balanced version of the dataset. As a reference of which output class number corresponds to which order of the variables, we refer to Table 1.

We note that while this renaming makes no difference to the meaning of the dataset, training an ML model on an unbalanced dataset can lead to the ML model simply choosing the class that occurs the most often, and being rewarded for higher accuracy in this way. In general, we might expect this to lead to overfitting, where the model may perform far worse on data from a different source. In our experiments the Decision Tree was found to have particularly suffered from this, making far poorer choices when trained with the balanced dataset, however, the other three models performed similarly well with balanced and unbalanced data and in particular, still outperformed the human-designed heuristics in the unbalanced case. The effect of such balancing on performance is explored in more detail by del Rio and England (2023).

From the point of view of this paper, balancing is particularly important for explainability. Our initial SHAP analysis on the models trained on the unbalanced dataset clearly reflected a simplis-

---

[6] We refer a reader seeking the details of these models to e.g. the textbook Bishop (2006).

[7] We refer a reader seeking the details of these implemented hyperparameters to the scikit-learn documentation for each respective model as linked from: https://scikit-learn.org/stable/supervised_learning.html.

Table 1

The six possible variable orderings for our dataset.

Ordering Name	Ordering
Ordering 1	$x_1 \succ x_2 \succ x_3$
Ordering 2	$x_1 \succ x_3 \succ x_2$
Ordering 3	$x_2 \succ x_1 \succ x_3$
Ordering 4	$x_2 \succ x_3 \succ x_1$
Ordering 5	$x_3 \succ x_1 \succ x_2$
Ordering 6	$x_3 \succ x_2 \succ x_1$

Table 2

Wall clock runtimes of the SHAP analysis for each model with 500 training points and 100 test points. CPU used is Intel(R) Xeon(R) CPU E5-1620 v3 @ 3.50 GHz.

Model	Total Run Time (HH:MM:SS)
KNN	07:02:42
MLP	00:09:52
DT	00:00:10
SVM	11:38:43

tic preference for the majority classes and gave us little insight into the important features of this application.

4.1.2. SHAP

SHAP is run on the four ML models. Table 2 shows the very significant differences in SHAP runtime between the models. This is due to both the nature of how long a single ML model takes to calculate a prediction, and the approximations that the SHAP library is able to make use of. We note that SHAP is a procedure we are running once to gain insight, and so are not greatly concerned with its one-off cost. Due to the long-running time for SHAP, 500 training points are used for each SHAP calculation, chosen randomly from the 2947 total training instances used to train the full models described above, and the SHAP calculations are performed on just 100 test points. This gives sufficient insight into the model decisions without requiring more than 12 hours of calculation time. We refer back to Section 3.1 for information on how SHAP makes its calculations.

4.1.3. Features

The derivation of the features used in the dataset is explained by Florescu and England (2019). In brief, instead of handpicking a set of features we algorithmically generate multiple features. This is done by acting upon a list of lists of lists with the innermost list capturing the degrees of variables in a monomial, which are gathered together next in polynomials, and then finally for the whole system. The actions performed on these are simple (cheap) operations such as taking the maximum, sum or average of the numbers. By looking at all possible combinations this method generates 81 distinct features for three variable systems that are the input features for the ML models. We define in Table 3 some of the terminology used in creating these features.

For example, $sum(max(v_1(S)))$ is one of the features that multiple models place importance upon. Its direct translation into plain English is "*the sum across all polynomials of the maximum of the degrees of its monomials for the variable x_1*", i.e. $\sum_p \max_{m \in p} degree(x_1, m)$ where m represents monomials and

Table 3
Translation of Features.

Notation	Text Translation
S	Lists of polynomials. E.g. $S = [x_1, x_1^2 - 2x_1x_2^2 + x_2^2 - 3]$.
$v_i(S)$	A list that has for each polynomial in S a list of the degrees of x_i in its monomials. E.g. $v_2(S) = [[0], [0, 2, 2, 0]]$.
$sv_i(S)$	A list that has for each polynomial in S a list of the total degrees in its monomials if x_i appears in the monomial, and 0 otherwise. E.g. $sv_1(S) = [[1], [2, 3, 0, 0]]$.
$op(L)$	If L is a list it performs operation op upon it. If L is a list of lists it returns a list containing the the result of the operation for each of the original lists. E.g. $max(avg(sv_1(S))) = max([1, 5/4]) = 5/4$.
$sg(L)$	This returns the same structure as L where each numerical value is substituted by its sign. E.g. $sum(sg(avg(sg(v_2(S))))) = sum(sg([0, 1/2]))$ $= sum([0, 1]) = 1$.

p polynomials. This simplifies to, "*the sum across all polynomials of their degrees for the variable x_1*", i.e. $\sum_p degree(x_1, p)$.

Similarly, the meaning of $avg(sg(max(v_1(S))))$ is "*the average across all polynomials of the sign of the maximum of the degrees of its monomials for the variable x_1*", which simplifies to "*the average number of polynomials that contain the variable x_1*".

4.2. SHAP results

In this section, we interpret the results from SHAP when applied to our problem using the methodology above. First, we explain how it can be used to give an explanation for the classification of a single problem instance: a local SHAP analysis. Then we aggregate across all instances to give an overview of what SHAP tells us about the most relevant features for each model over the dataset: a global SHAP analysis.

4.2.1. Local SHAP analysis

For each ML method, for each of the six binary classification models, and for each instance, SHAP can produce a waterfall plot (see Section 3.2) to show the effect each feature has on the final output. As an example, let us look in detail at the MLP model for a particular test instance. An explanation for the binary classifier of whether or not the instance belongs to class 5 (selects Ordering 5) is given via the waterfall plot in Fig. 3. The blue arrows in the waterfall indicate that the feature is pushing the prediction towards an output of 0 (that Ordering 5 is the wrong ordering), and the red arrows that the feature pushes towards an output of 1 (that Ordering 5 is the right ordering).

The binary classifier suggested the instance should be classified as Ordering 5 with a probability of 0.71. This was the highest probability among the six binary classifiers and so Ordering 5 was selected as the classification. Orderings $1 - 4$ get a score very close to 0. Ordering 6 receives a score much higher than 0, but not as high as Ordering 5. This is not surprising: Orderings 5 and 6 both select the same variable to project first and it is sensible to assume that this would have the greatest impact on the calculation time of the CAD, since its effects will be magnified in the subsequent projections.

4.2.2. Global SHAP analysis

To get the global importance of the features for each model over the dataset we may aggregate the absolute values of the contribution of the features across all instances and binary classifiers. After this, the five most important features for each model found by SHAP are given in Table 4.
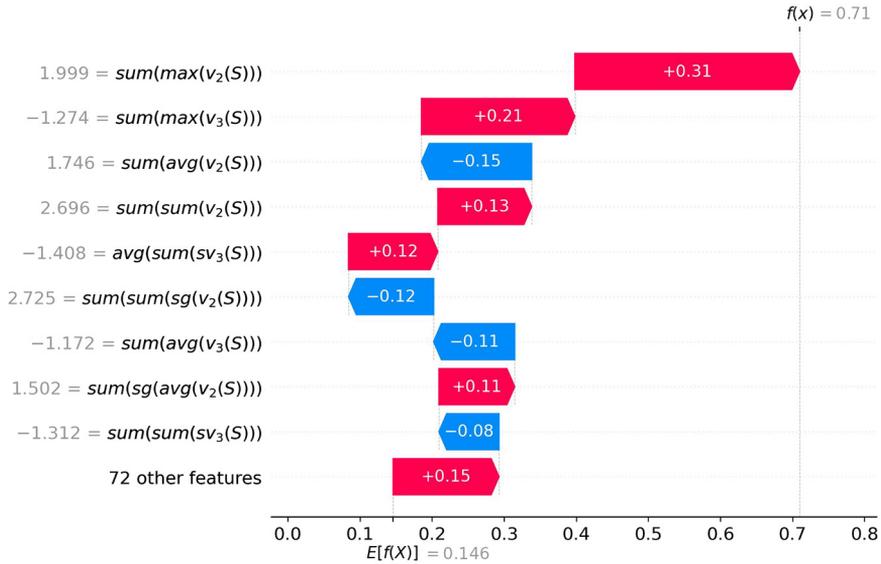
Fig. 3. An explanation of a decision made by the MLP model on an example CAD problem instance, for the selected output ordering, ordering 5: $x_3 \succ x_1 \succ x_2$.

Plots of the relative importance of features to each model by class are given in Fig. 4. These plots give a good insight into not only that feature's impact but also how a model trains and learns from the data. The KNN model, for example, creates little distinction between the impacts of the features. This is what we would expect from a KNN, because the model is looking for those instances which are most similar to the instance being evaluated to make a decision. It is the aggregate of the number of instances similar to a certain test point that will decide the output. This applies similarly to the SVM. For both models, this continues past the top five features. Specifically, the difference from the feature with the highest average impact on the model output magnitude to the twentieth feature is about 0.012 for the KNN, and 0.005 for the SVM. In contrast, the MLP and DT models show more distinction in the importance of these features: decision trees in particular are known to place importance on a few features. To compare, the difference from the top to the fifth feature is about 0.28 for the ML.

Upon close examination of Fig. 4, some interesting phenomena may be observed. We see that often the impact on the output class for features of a certain variable is largest if that variable is last in the ordering of that output class. For example, the feature with the greatest impact on the KNN model is an x_3 feature, and Orderings 1 and 3 have the highest proportional impact which (referring back to Table 1) are the orderings with x_3 last. We see this for the x_2 features and the SVM model as well. But the MLP model shows different behaviour. Here, only x_2 and x_3 features are seen, and both Orderings 3 and 5 have the greatest impact.

4.2.3. Features in human-designed heuristics

The `Brown` heuristic (Brown, 2004) uses three simple metrics to choose a variable ordering, all of which may be described in the language of Section 4.1.3:

- $max(max(v_i(S)))$,
- $max(max(sv_i(S)))$, and
- $sum(sum(sg(v_i(S))))$.

We cannot compare directly to the features of the other human-designed heuristics in Section 2.3 as they all make use of expensive algebraic operations rather than using features from the input polynomials alone.

Table 4

The top five features that each model found to have the highest average impact on model output magnitude.

	KNN	MLP
1	$avg(max(v_3(S)))$	$sum(max(v_3(S)))$
2	$sum(max(v_3(S)))$	$sum(max(v_2(S)))$
3	$avg(max(v_2(S)))$	$sum(avg(v_3(S)))$
4	$sum(max(v_2(S)))$	$avg(sum(sv_3(S)))$
5	$max(max(v_1(S)))$	$sum(avg(v_2(S)))$

	SVM	DT
1	$avg(max(v_1(S)))$	$avg(avg(v_2(S)))$
2	$avg(max(v_2(S)))$	$avg(avg(sg(v_1(S))))$
3	$sum(max(v_2(S)))$	$max(sum(v_3(S)))$
4	$avg(max(v_3(S)))$	$avg(avg(v_3(S)))$
5	$sum(max(v_3(S)))$	$avg(max(v_1(S)))$

5. Post SHAP analysis to identify the most relevant features

SHAP produced for us a global analysis explaining the features that are important to each of the four ML models we experiment with. We now work to create a unified ranking of the features across the four models for the purpose of creating new human-level heuristics in Section 6.

### 5.1. Merging features that generate the same heuristic

We aim for heuristics independent of particular variable names which are present in the ML features. I.e. the prominence of $sum(sum(v_1(S)))$ in ML will lead us to the same heuristic as $sum(sum(v_3(S)))$. So we now aggregate the contribution of the features across the three variables, resulting in rankings like the one in Table 5 for each model. This reduces the original 81 features to 27. We use $v_i(S)$ as a short notation to denote the combination across the three variables, i.e. $sum(sum(v_i(S))) := \sum_{i=1}^{3} sum(sum(v_i(S)))$.

Moreover, some features, even though they produce different values for the instances, share some relations. In particular, some of these features are proportional to each other (e.g. $sum(max(v_1(S)))$ is going to be a multiple of $(avg(max(v_1(S))))$ for every set of polynomials and therefore they can be seen theoretically to induce the same heuristics. This additional merging reduces the number of features further to 18. The ranking of such features for the MLP model is given in Table 6. Similar rankings are available for the other three models.

5.2. Creating a unified ranking

We now wish to create a combined feature ranking across the four models. We will create this by viewing it as a voting problem, where each of the models has a different ranking for the features and we have to get a combined ranking. A usual method to combine such rankings is the Borda Count: where for each ranking we assign a penalisation of 1 to the first, 2 to the second, and so on; at the end ranking the options from lowest penalization to highest. However, this method strongly penalises being very badly ranked for just one of the models. Meaning that a feature that is the most relevant for three of the models while being the most irrelevant for one of the models would not end up in a good position in the final ranking.

Thus we use a modification, the Dowdall System, that assigns the first feature a reward of 1, the second a reward of 1/2, the third receives 1/3, and so on until, at the end ranking the options from higher reward to lowest. See for example the work of Fraenkel and Grofman (2014) on these different
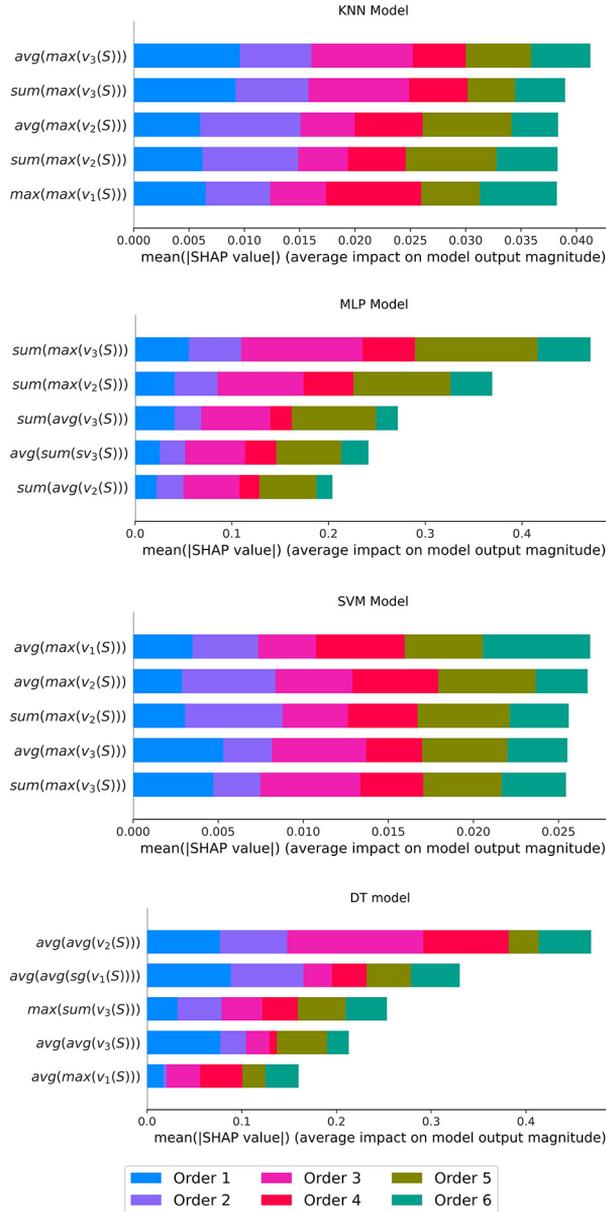
Fig. 4. The top five features for each model from aggregating over the 100 test points. Ordered by average impact on model output magnitude, and coloured by output class. (For interpretation of the colours in the figure(s), the reader is referred to the web version of this article.)

voting systems. We chose the Dowdall system because it allows merging the preferences without a big penalisation for being a very distant preference for one of the models, contrary to the Borda Count.

Table 7 shows the features and their scores in our final ranking and their rewards from the Dowdall System. The most voted feature is the sum of the degrees of the polynomials in the set, which is precisely the feature used in `gmods` (del Río and England, 2022), found through the study of the

Table 5
Features important for the Multi Layer Perceptron after merging across the variables.

Feature Name	Summed SHAP Value
$sum(max(v_i(S)))$	90.716
$sum(avg(v_i(S)))$	57.566
$sum(max(sv_i(S)))$	41.69
$sum(sum(v_i(S)))$	36.67
$avg(avg(sg(v_i(S))))$	35.676
$avg(sum(sv_i(S)))$	34.758
$avg(max(sv_i(S)))$	30.981
$avg(avg(v_i(S)))$	29.248
$sum(sum(sv_i(S)))$	28.757
$avg(avg(sv_i(S)))$	26.053
$sum(sum(sg(v_i(S))))$	25.164
$sum(sg(avg(v_i(S))))$	23.751
$sum(avg(sv_i(S)))$	21.86
$avg(sum(sg(v_i(S))))$	21.495
$sum(avg(sg(v_i(S))))$	18.058
$avg(sg(sum(v_i(S))))$	16.486
$max(max(v_i(S)))$	15.033
$max(avg(v_i(S)))$	13.687
$max(sum(sv_i(S)))$	13.443
$max(max(sv_i(S)))$	12.029
$avg(max(v_i(S)))$	11.864
$max(max(sg(v_i(S))))$	11.586
$max(avg(sv_i(S)))$	11.559
$max(sum(v_i(S)))$	11.389
$max(sum(sg(v_i(S))))$	10.451
$avg(sum(v_i(S)))$	9.318
$max(avg(sg(v_i(S))))$	7.493

complexity analysis of CAD. In this ranking: the eleventh, eighteenth (bottom) and eighth most voted features are those used by the Brown heuristic (Brown, 2004).

The second most voted feature, the average across the polynomials of the average degree of the variable across the monomials, is a feature that has been never considered relevant in any prior work.

The top six features are chosen for further experimentation in the next section to find new human-level heuristics. We draw the line here as there is a big drop off and the lower features score more similarly to each other, i.e. the feature scores stabilize after the sixth score. This stabilization of the scores is illustrated in Fig. 5.

6. New human-level heuristics motivated by XAI

In this section, we will first introduce the methodology we use to evaluate a heuristic, and then experiment with new human-level heuristics motivated by Section 5. We will compare these against the current state-of-the-art heuristics for choosing the variable ordering for CAD.

To evaluate the performance of these heuristics two things are needed: a set of meaningful benchmark examples and metrics to quantify the performance of the heuristics on them. We will use the benchmarks and metrics described recently by del Río and England (2022) for this purpose.

Table 6
Features in Multi Layer Perceptron after merging those that would generate the same heuristic.

Feature Name	Summed SHAP Value
$sum(max(v_i(S)))$	102.58
$avg(avg(v_i(S)))$	86.814
$sum(max(sv_i(S)))$	72.671
$sum(sum(sv_i(S)))$	63.515
$avg(avg(sg(v_i(S))))$	53.735
$avg(avg(sv_i(S)))$	47.913
$sum(sum(sg(v_i(S))))$	46.66
$sum(sum(v_i(S)))$	45.988
$avg(sg(sum(v_i(S))))$	40.236
$max(max(v_i(S)))$	15.033
$max(avg(v_i(S)))$	13.687
$max(sum(sv_i(S)))$	13.443
$max(max(sv_i(S)))$	12.029
$max(max(sg(v_i(S))))$	11.586
$max(avg(sv_i(S)))$	11.559
$max(sum(v_i(S)))$	11.389
$max(sum(sg(v_i(S))))$	10.451
$max(avg(sg(v_i(S))))$	7.493

Table 7
Voted score of merged and aggregated features across all models.

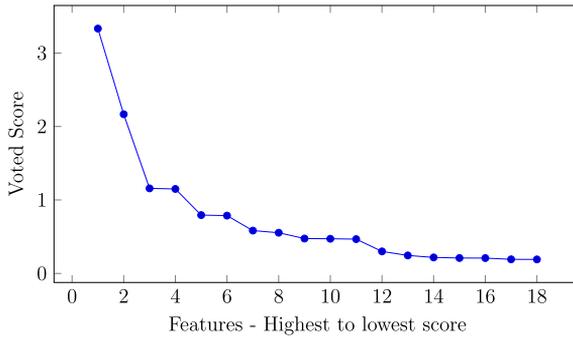| Feature Name	Voted Score
$sum(max(v_i(S)))$	3.333
$avg(avg(v_i(S)))$	2.167
$sum(sum(v_i(S)))$	1.158
$avg(avg(sg(v_i(S))))$	1.15
$sum(sg(sum(v_i(S))))$	0.794
$sum(max(sv_i(S)))$	0.787
$avg(avg(sv_i(S)))$	0.583
$sum(sum(sg(v_i(S))))$	0.554
$max(sum(v_i(S)))$	0.475
$sum(sum(sv_i(S)))$	0.472
$max(max(v_i(S)))$	0.467
$max(sum(sg(v_i(S))))$	0.3
$max(avg(v_i(S)))$	0.245
$max(max(sg(v_i(S))))$	0.218
$max(avg(sg(v_i(S))))$	0.21
$max(sum(sv_i(S)))$	0.209
$max(avg(sv_i(S)))$	0.192
$max(max(sv_i(S)))$	0.191

Fig. 5. Plot of feature scores from Table 7.

6.1. Benchmarks

We source benchmarks from the three-variable problems in the QF_NRA category of the SMT-LIB (Barrett et al., 2016). These examples are all satisfiability problems and so do not represent the full application range of CAD which can also address quantifier elimination. However, they do mostly emit from real applications making performance upon them particularly meaningful. Common sources are problems emitting from the theorem prover MetiTarski (Paulson, 2012), attempts to prove termination of term-rewrite systems, verification conditions from Keymaera (Platzer et al., 2009), and parametrized generalizations of geometric problems, as well as problems emitting from economics (Mulligan et al., 2018) and biology (Bradford et al., 2020).

A sign-invariant CAD is built using Maple (Chen et al., 2009) for each example in each variable ordering: 5599 problems finished for at least one ordering before the time limit (60 seconds). Of these, the authors select 1019 "*unique*" problems where, as described in (del Río and England, 2022, Section 4.1), unique means the CAD tree structures built are not identical to another example for every ordering. This benchmark merging is needed since some application sources in the SMT-LIB produce examples that differ from each other only slightly, leading to identical CAD computations, and thus potentially unfair evaluation. The benchmarks used here are identical to those of del Río and England (2022), which are stored in an open source dataset released for that paper.[8]

The problems selected have a range of difficulties, as shown by the histogram in Fig. 6 which plots the number of problems against the CAD computation time of their optimal ordering. The vertical axis (number of problems) has a logarithmic scale, showing that the dataset has much more easy problems. There is a need to enlarge the SMT-LIB with more difficult problems.

6.2. Evaluation metrics

For evaluation metrics, we again follow del Río and England (2022) and use the three selected there as follows.

- `Accuracy:` Percentage of benchmarks in which the optimal ordering (quickest in runtime) coincides with that chosen by the heuristic.

This is the most commonly used tool in ML classification and was used to train the ML models earlier.[9] However, it is not ideally suited to our problem: it considers as equally wrong a terrible ordering and an ordering that is slightly sub-optimal, even if the difference compared to the optimal ordering is so small that it could be due to computational noise.

---

[8] https://doi.org/10.5281/zenodo.6750528.
[9] See (Florescu and England, 2020a) for an attempt to train ML partially with a different metric.
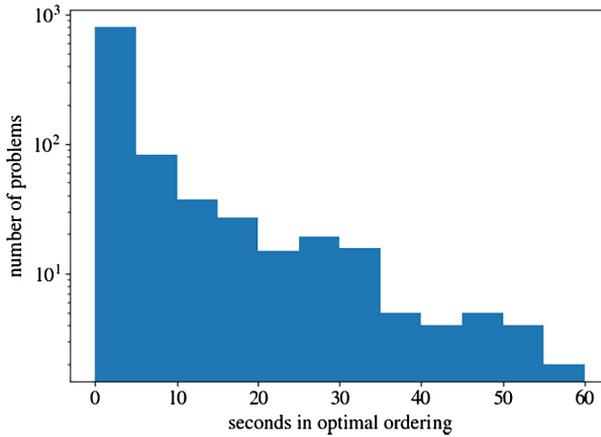
Fig. 6. Histogram showing how the 1019 problems selected from the SMT-LIB for our experiments range in difficulty. The horizontal axis has 5 second time segments and the vertical axis shows the number of problems whose optimal ordering took that level of computation time (in a logarithmic scale).

- `Total time:` Amount of time needed to create a CAD for each benchmark when choosing the variable ordering given by the heuristic.

This metric corresponds most closely to the absolute performance achieved and would be most commonly used to rank a computer algebra implementation. However, it will not take into account the chosen orderings for the smaller problems in our dataset and could allow a small number of large problems to distort findings. Hence del Río and England (2022) proposed the next metric to address the shortcomings of the above: a metric that does not ignore the simple problems, but at the same time distinguishes between different non-optimal orderings.

- `Markup:` The markup of the time needed by the heuristic ($t_{heuristic}$) with respect to the time using the optimal choice ($t_{optimal}$). Defined as the average across all benchmarks of

$$\text{markup} = \frac{t_{heuristic} - t_{optimal}}{t_{optimal} + 1}.$$

  The $+1$ is included in the denominator to reduce the impact that the computational noise can have on problems that only require tenths of seconds.

When a CAD could not be created within the time limit using the ordering chosen by a heuristic, the time of the heuristic was set in the dataset to twice the time limit. These runtimes are also in the Zendodo data release for (del Río and England, 2022).

6.3. Existing heuristics

The prior state-of-the-art heuristic for choosing CAD variable ordering is `gmods`, as presented by del Río and England (2022). This is based on a metric motivated by the complexity analysis of CAD. The analysis found that given a set of polynomials S_n and the ordering $x_{i_n} \succ \cdots \succ x_{i_1}$, the maximum number of cells that can be generated is proportional to

$$\prod_{j=1}^{n} sum(max(v_{i_j}(S_j))), \tag{1}$$

where S_{j-1} is the projection of S_j with respect to x_{i_j}.

The heuristic `mods` was first proposed by del Río and England (2022) as a heuristic that computes the sets $S_1, \ldots, S_{n-1}$ for each variable ordering and then chooses the ordering that minimises the worst case complexity, i.e. that minimises (1). This performed very well, in the sense that it chooses good orderings. However, it is an expensive heuristic: the cost of computing the CAD projection phase for all the orderings was not compensated by the improved quality of the ordering it chooses, at least not for the dataset in question. This is illustrated later by the experimental results in Table 8, where `mods` took into account the additional cost of the heuristic while `free-mods` does not, we report on the performance of the hypothetical heuristic `free-gmods` for comparison purposes later.

To solve this issue del Río and England (2022) decided to avoid doing any projections for choosing the variable ordering except those utilised by the actual CAD. They proposed to pick the first variable of the ordering x_{i_n} to be the one with minimal $sum(max(v_i(S_n)))$, choosing the one with the lowest index in case of a tie. Only then do they take a CAD projection of S_n with respect to x_{i_n} to obtain S_{n-1}, which is then studied to pick the second variable $x_{i_{n-1}}$ as the one with minimal $sum(max(v_i(S_{n-1})))$. This allows the heuristic to gather as much information as possible before choosing the next variable, continuing this process to determine the whole variable ordering.

This heuristic does not require any more CAD projections than those needed to run the CAD algorithm. In this paper we refer to heuristics that follow this structure as "*greedy*" (in the traditional algorithmic sense that they make a choice based on the local information). All the heuristics introduced below are greedy.[10]

Since the source benchmarks contain a bias towards certain orderings, the authors choose a variable randomly whenever there is a tie, instead of choosing the one with the lowest index or the first lexicographically which could allow that bias to effect the results.

For many years, a well-known and widely used heuristic is that of Brown (2004). Denoted `Brown`, this uses three criteria, in turn, breaking ties with the subsequent ones. It projects first the variable that minimizes:

- $max(max(v_i(S)))$, (i.e. variable with lowest overall degree);
  breaking ties with
- $max(max(sv_i(S)))$, (i.e. variable with the lowest total degree in the monomials containing it);
  breaking ties with
- $sum(sum(sg(v_i(S))))$, (i.e. number of terms containing the variable).

It is not specified what to do if there is a third tie: our implementation simply picks between the tied variables randomly. Moreover, it is not specified whether the whole variable ordering is chosen at once or one variable at a time. In this paper, we implement the greedy variant: i.e. we choose the first variable to be projected, and then use the projected polynomials to choose the next variable in the ordering.

Table 8 evaluates these existing heuristics on our benchmark set. We include also the performance of a virtual best heuristic (the hypothetical heuristic which always picks the optimal ordering) to show the best possible figures for the dataset, and a heuristic that picks at random. As in (del Río and England, 2022) we find `gmods` to be the best of all metrics except accuracy: where it is beaten by `mods` but at a cost not worth paying for this dataset.

Note that for some heuristics the number of problems completed is not a natural number. This is because the heuristic is not deterministic (due to the random choices to break ties) and the experiments are repeated 5000 times to obtain an average to account for this.

6.4. Creating new greedy heuristics from XAI identified features

We will now build new heuristics solely based on a single feature from the earlier ML models. We aim to see how powerful these features are when taking decisions on their own.

---

[10] We note that `greedy-sotd` presented in (Dolzmann et al., 2004) is not greedy in the sense of this paper because it makes a decision based on all the possible single projections for the next step (still less than the original `sotd` which computed all full projections).

Table 8

Evaluation of the existing heuristics to choose the variable orderings for CAD. In bold, the best measure of the metric out of all the heuristics. Note that after the double line, heuristics are not.

Name	Accuracy	Total time	Markup	# Completed
gmods	0.563	**7192.2**	**0.212**	**982.6**
Brown	0.553	7842.6	0.278	968.9
mods	**0.639**	8137	0.127	979
random	0.167	20797.3	4.034	262.5
free-mods	0.639	6637	0.566	990
virtual-best	1	4822.7	0	1019

The explainability analysis in Section 5 found $sum(max(v_i(S)))$ to be the most impactful feature. A greedy heuristic SumMaxV based on this feature will select the variable for which this feature is minimal for each projection (and in the case of a tie one of the tied variables will be chosen randomly).

Consider for example the set $S_3 = \{x_1 x_2 x_3 - 1, x_1^2 - x_2^2 x_3\}$; then SumMaxV will work as follows. First, the feature will be computed for each variable:

$$sum(max(v_1(S_3))) = sum([1, 2]) = 3,$$

$$sum(max(v_2(S_3))) = sum([1, 2]) = 3,$$

$$sum(max(v_3(S_3))) = sum([1, 1]) = 2.$$

So x_3 is projected first because it has the minimal value. The Lazard projection (McCallum and Hong, 2016) of S_3 with respect to x_3 gives $\{x_1^3 x_2 - x_2^2, x_1 x_2, x_2^2, x_1^2\}$. Factorizing and removing repeated factors gives $S_2 = \{x_1^3 - x_2, x_1, x_2\}$ and again we compute this feature for each of the remaining variables:

$$sum(max(v_1(S_2))) = sum([3, 1, 0]) = 4 \text{ and,}$$

$$sum(max(v_2(S_2))) = sum([1, 0, 1]) = 2.$$

So x_2 is projected next because it has the minimal value. Thus heuristic SumMaxV chooses the ordering $x_3 \succ x_2 \succ x_1$.

6.5. Performance of new single feature heuristics

We now evaluate the new heuristics based upon the features that ranked in the top six in Table 7 (selected as discussed at the end of Section 5.2). AvgAvgSgV is based upon feature $avg(avg(sg(v_i(S))))$, SumMaxSV is based upon $sum(max(sv_i(S)))$ and so on. The results are shown in Table 9.

It is not a coincidence that the statistics for SumMaxV in Table 9 are the same as for gmods in Table 8: gmods can be viewed as single feature heuristic based on $sum(max(v_i(S)))$.

The table shows there are two heuristics features that stand out: SumMaxV (gmods) and AvgAvgV. These achieve similar results, and to choose the best feature would depend on the metric used: neither dominates the other.

6.6. New heuristics of triples of features identified by SHAP

We have now seen a number of single-feature heuristics doing well, without a clear winner. We note that even the best of these (gmods) still relies on a random tie-breaker for 25% of the problem instances, indicating it does not alone have access to enough information. So we will next combine these single features into triples, using subsequent ones as tie-breakers. This approach is the same as that used by the heuristic of Brown (2004). This should give a more informed choice than selecting a random variable and is still certainly at a human level in size and complexity.

Table 9

Evaluation metrics for the new heuristics to choose the variable order-ings for CAD. In bold, the best measure of the metric out of all the heuristics.

Name	Accuracy	Total time	Markup	# Completed
SumMaxV	**0.563**	7192.2	**0.212**	982.6
AvgAvgV	0.544	**7138.7**	0.224	**983.5**
SumSumV	0.549	7524.8	0.261	975.3
AvgAvgSgV	0.535	8682.6	0.559	956.3
SumSgSumV	0.45	10836.7	1.223	922.5
SumMaxSV	0.509	8771.7	0.563	956.5

Table 10

Evaluation metrics for the different heuristics to choose the vari-able orderings for CAD. In bold, the best measure of the metric out of all the heuristics. T1=SumMaxV>AvgAvgV>SumSumV and T2=SumMaxV>SumSumSgV>SumSumV.

Name	Accuracy	Total time	Markup	# Completed
Brown	0.553	7842.6	0.278	968.9
T1	0.567	**6896.3**	0.193	**985.7**
T2	**0.583**	6896.7	**0.188**	984.8
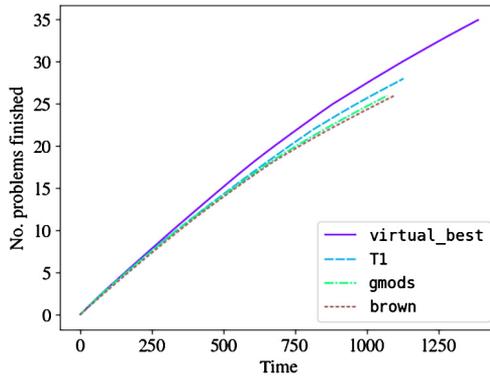
Fig. 7. Survival plot for best-existing heuristics and new heuristics proposed in the problems that take more than 30 seconds for all the orderings.

We will denote these heuristics based on multiple features by concatenating their names with a >, where smaller indicates that it is used as a tie-breaker for the bigger features. Following this notation the Brown heuristic would be denoted as MaxMaxV>MaxMaxSV>SumSumSgV.

There are 120 possible ordered triples that can be created from the top six features of Table 7. For each we create a heuristic that chooses the variable that minimises the first feature, breaking ties using the second feature and the third if necessary, and finally breaking ties by choosing a random variable if there is a tie for all three features.

Performance statistics for the two triples that are best in one of the metrics are given in Table 10, along with Brown that makes choices with a similar quantity of information.

Table 10 shows that the heuristic T1=SumMaxV>AvgAvgV>SumSumV wins in two of the four metrics: total time and the number of problems completed within the time limit. T1 relies on the random tie-breaker for only 5% of the problem instances, far less than any of the single feature heuris-

tics, as expected. Note that this heuristic is created from the top three features XAI suggested features, in the order they appear in the voting system.

- $sum(max(v_i(S)))$, (i.e. variable with the lowest degree in the product of the polynomials); breaking ties with
- $avg(avg(v_i(S)))$, (i.e. variable with the lowest average of average degree in the polynomials); breaking ties with
- $sum(sum(v_i(S)))$, (i.e. variable with the lowest sum of all its degrees).

This triple also outperforms all the singleton heuristics, and its performance gets really close to the performance of the hypothetical heuristic `free-gmods`, shown in Table 8.

In Fig. 7 we give a survival plot comparing the triple T1 to the prior state-of-the-art and the virtual best. We do this just for the hardest problems in the dataset (those taking more than 30 seconds in all orderings) to allow for a clearer visualisation of the differences. The figure shows the leap that existed between the prior state-of-the-art and the best possible in theory, and that the new XAI informed triple bridges a good deal of that gap.

7. Conclusions

7.1. Summary

In this paper, we have devised a new state-of-the-art human-level heuristic for choosing a CAD variable ordering, at least in three variable problems. Previous approaches to the problem were of two types: either an expert handpicking features to create a human-designed heuristic; or an ML model computing with many features under the hood. We presented an intermediate approach where we start with the ML model but then use XAI to select important features, which may then be used later independently of any ML technology.

We note that SHAP was able to find the feature identified recently by del Río and England (2022) that was used to form the prior state-of-the-art heuristic; and identified also the tools of the heuristic in Brown (2004). But it also found an important feature not previously considered by experts.

7.2. Future work

There is of course scope for future work. Using multiple features clearly improved the results, and there are certainly others ways we could consider combining them. Perhaps this combination itself could also be ML informed? It would also be interesting to explore alternative explainable AI methods, especially ML models that are inherently explainable, rather than the post-hoc explainability analysis of SHAP. We also acknowledge the need to experiment with other datasets including problems with more variables.

The experiments presented in this paper are for selecting CAD variable ordering. However, the methodology could be applied almost directly to variable ordering choices for other algorithms, and could be adapted to many algorithm optimisation problems in symbolic computation and beyond. We have shown that ML has a role to play in symbolic computation, even for developers who prefer not to include a reliance upon it in their final code. It can be used as a tool to guide and develop algorithms at a human level. We thus hope and expect to see other applications of XAI to inform decisions in CASs and optimise symbolic computation algorithms in the coming years.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The code and data supporting the research in this paper are freely available from Zenodo: https://doi.org/10.5281/zenodo.8229298.

Acknowledgements

References

Barrett, C., Fontaine, P., Tinelli, C., 2016. The satisfiability modulo theories library (SMT-LIB). Online resource. https://smtlib.cs.uiowa.edu/.

Bernal, E.A., Hauenstein, J.D., Mehta, D., Regan, M.H., Tang, T., 2023. Machine learning the real discriminant locus. J. Symb. Comput. 115, 409–426. https://doi.org/10.1016/j.jsc.2022.08.001.

Bishop, C.M., 2006. Pattern Recognition and Machine Learning. Springer. https://link.springer.com/book/9780387310732.

Bradford, R., Davenport, J.H., England, M., Errami, H., Gerdt, V., Grigoriev, D., Hoyt, C., Košta, M., Radulescu, O., Sturm, T., Weber, A., 2020. Identifying the parametric occurrence of multiple steady states for some biological networks. J. Symb. Comput. 98, 84–119. https://doi.org/10.1016/j.jsc.2019.07.008.

Bradford, R., Davenport, J.H., England, M., Wilson, D., 2013. Optimising problem formulations for cylindrical algebraic decomposition. In: Carette, J., Aspinall, D., Lange, C., Sojka, P., Windsteiger, W. (Eds.), Intelligent Computer Mathematics. In: Lecture Notes in Computer Science, vol. 7961. Springer Berlin Heidelberg, pp. 19–34. https://doi.org/10.1007/978-3-642-39320-4_2.

Brown, C.W., 2004. Companion to the tutorial "Cylindrical algebraic decomposition" presented at ISSAC '04. https://www.usna.edu/Users/cs/wcbrown/research/ISSAC04/handout.pdf.

Brown, C.W., 2015. Open non-uniform cylindrical algebraic decompositions. In: Proceedings of the 2015 International Symposium on Symbolic and Algebraic Computation. ACM, pp. 85–92. https://doi.org/10.1145/2755996.2756654.

Brown, C.W., Davenport, J.H., 2007. The complexity of quantifier elimination and cylindrical algebraic decomposition. In: Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation. ACM, pp. 54–60. https://doi.org/10.1145/1277548.1277557.

Buchberger, B., 2006. Bruno Buchberger's PhD thesis (1965): an algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal. J. Symb. Comput. 41, 475–511. https://doi.org/10.1016/j.jsc.2005.09.007.

Carette, J., 2004. Understanding expression simplification. In: Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation. ACM, pp. 72–79. https://doi.org/10.1145/1005285.1005298.

Chen, C., Moreno Maza, M., Xia, B., Yang, L., 2009. Computing cylindrical algebraic decomposition via triangular decomposition. In: Proceedings of the 2009 International Symposium on Symbolic and Algebraic Computation. ACM, pp. 95–102. https://doi.org/10.1145/1576702.1576718.

Chen, C., Zhu, Z., Chi, H., 2020. Variable ordering selection for cylindrical algebraic decomposition with artificial neural networks. In: Bigatti, A., Carette, J., Davenport, J.H., Joswig, M., de Wolff, T. (Eds.), Mathematical Software – ICMS 2020. Springer International Publishing, pp. 281–291. https://doi.org/10.1007/978-3-030-52200-1_28.

Collins, G.E., 1975. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. Lect. Notes Comput. Sci. 33, 134–183. https://doi.org/10.1007/3-540-07407-4_17.

Collins, G.E., Hong, H., 1991. Partial cylindrical algebraic decomposition for quantifier elimination. J. Symb. Comput. 12, 299–328. https://doi.org/10.1016/S0747-7171(08)80152-6.

Davenport, J.H., Heintz, J., 1988. Real quantifier elimination is doubly exponential. J. Symb. Comput. 5, 29–35. https://doi.org/10.1016/S0747-7171(88)80004-X.

Davies, A., Veličković, P., Buesing, L., Blackwell, S., Zheng, D., Tomašev, N., Tanburn, R., Battaglia, P., Blundell, C., Juhász, A., Lackenby, M., Williamson, G., Hassabis, D., Kohli, P., 2021. Advancing mathematics by guiding human intuition with AI. Nature 600, 70–74. https://doi.org/10.1038/s41586-021-04086-x.

Dolzmann, A., Seidl, A., Sturm, T., 2004. Efficient projection orders for CAD. In: Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation. ACM, pp. 111–118. https://doi.org/10.1145/1005285.1005303.

Du, M., Liu, N., Hu, X., 2019. Techniques for interpretable machine learning. Commun. ACM 63, 68–77. https://doi.org/10.1145/3359786.

England, M., Bradford, R., Davenport, J.H., 2020. Cylindrical algebraic decomposition with equational constraints. J. Symb. Comput. 100, 38–71. https://doi.org/10.1016/j.jsc.2019.07.019.

England, M., Florescu, D., 2019. Comparing machine learning models to choose the variable ordering for cylindrical algebraic decomposition. In: Kaliszyk, C., Brady, E., Kohlhase, A., Sacerdoti, C.C. (Eds.), Intelligent Computer Mathematics. Springer International Publishing, pp. 93–108. https://doi.org/10.1007/978-3-030-23250-4_7.

Florescu, D., England, M., 2019. Algorithmically generating new algebraic features of polynomial systems for machine learning. In: Abbott, J., Griggio, A. (Eds.), Proceedings of the 4th Workshop on Satisfiability Checking and Symbolic Computation (SC^2 2019), p. 12. http://ceur-ws.org/Vol-2460/.

Florescu, D., England, M., 2020a. Improved cross-validation for classifiers that make algorithmic choices to minimise runtime without compromising output correctness. In: Slamanig, D., Tsigaridas, E., Zafeirakopoulos, Z. (Eds.), Mathematical Aspects of Computer and Information Sciences (Proc. MACIS '19). Springer International Publishing, pp. 341–356. https://doi.org/10.1007/978-3-030-43120-4_27.

Florescu, D., England, M., 2020b. A machine learning based software pipeline to pick the variable ordering for algorithms with polynomial inputs. In: Bigatti, A., Carette, J., Davenport, J.H., Joswig, M., de Wolff, T. (Eds.), Mathematical Software – ICMS 2020. Springer International Publishing, pp. 302–322. https://doi.org/10.1007/978-3-030-52200-1_30.

Fraenkel, J., Grofman, B., 2014. The Borda Count and its real-world alternatives: comparing scoring rules in Nauru and Slovenia. Aust. J. Polit. Sci. 49, 186–205. https://doi.org/10.1080/10361146.2014.900530.

Giovini, A., Mora, T., Niesi, G., Robbiano, L., Traverso, C., 1991. One sugar cube, please; or selection strategies in the Buchberger algorithm. In: Proceedings of the 1991 International Symposium on Symbolic and Algebraic Computation. ACM, pp. 49–54. https://doi.org/10.1145/120694.120701.

Gunning, D., Aha, D., 2019. DARPA's explainable artificial intelligence (XAI) program. AI Mag. 40 (2), 44–58. https://doi.org/10.1609/aimag.v40i2.2850.

Hauenstein, J.D., He, Y.H., Kotsireas, I., Mehta, D., Tang, T., 2023. Special issue on algebraic geometry and machine learning. J. Symb. Comput. 118, 93–94. https://doi.org/10.1016/j.jsc.2022.10.003.

He, Y.H., 2022. Machine-learning mathematical structures. Int. J. Data Sci. Math. Sci. 1, 1–25. https://doi.org/10.1142/S2810939222500010.

Herm, L.V., Heinrich, K., Wanner, J., Janiesch, C., 2022. Stop ordering machine learning algorithms by their explainability! A user-centered investigation of performance and explainability. Int. J. Inf. Manag. 69, 102538. https://doi.org/10.1016/j.ijinfomgt.2022.102538.

Huang, Z., England, M., Davenport, J.H., Paulson, L., 2016. 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). IEEE, pp. 45–52. https://doi.org/10.1109/SYNASC.2016.020.

Huang, Z., England, M., Wilson, D., Davenport, J.H., Paulson, L., Bridge, J., 2014. 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. In: Lecture Notes in Artificial Intelligence, vol. 8543. Springer International, pp. 92–107. https://doi.org/10.1007/978-3-319-08434-3_8.

Huang, Z., England, M., Wilson, D.J., Bridge, J., Davenport, J.H., Paulson, L.C., 2019. Using machine learning to improve cylindrical algebraic decomposition. Math. Comput. Sci. 13, 461–488. https://doi.org/10.1007/s11786-019-00394-8.

Jovanovic, D., de Moura, L., 2012. Solving non-linear arithmetic. In: Gramlich, B., Miller, D., Sattler, U. (Eds.), Automated Reasoning: 6th International Joint Conference (IJCAR). In: Lecture Notes in Computer Science, vol. 7364. Springer, pp. 339–354. https://doi.org/10.1007/978-3-642-31365-3_27.

Kobayashi, M., Iwane, H., Matsuzaki, T., Anai, H., 2016. 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). Springer International Publishing, pp. 236–251. https://doi.org/10.1007/978-3-319-32859-1_21.

Kremer, G., Ábrahám, E., 2020. Fully incremental CAD. J. Symb. Comput. 100, 11–37. https://doi.org/10.1016/j.jsc.2019.07.018.

Kuipers, J., Ueda, T., Vermaseren, J.A.M., 2015. Code optimization in FORM. Comput. Phys. Commun. 189, 1–19. https://doi.org/10.1016/j.cpc.2014.08.008.

Lample, G., Charton, D., 2020. Deep learning for symbolic mathematics. In: Mohamed, S., White, M., Cho, K., Song, D. (Eds.), Eighth International Conference on Learning Representations (ICLR 2020), p. 24. https://iclr.cc/virtual_2020/poster_S1eZYeHFDS.html.

Lundberg, S., 2022. SHAP v1.40.0 (GitHub). https://github.com/slundberg/shap.

Lundberg, S.M., Erion, G., Chen, H., DeGrave, A., Prutkin, J.M., Nair, B., Katz, R., Himmelfarb, J., Bansal, N., Lee, S.I., 2020. From local explanations to global understanding with explainable AI for trees. Nat. Mach. Intell. 2, 2522–5839. https://doi.org/10.1038/s42256-019-0138-9.

Lundberg, S.M., Lee, S.I., 2017. A unified approach to interpreting model predictions. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. Curran Associates Inc., Red Hook, NY, USA, pp. 4768–4777. https://dl.acm.org/doi/10.5555/3295222.3295230.

Manubens, M., Moroz, G., Chablat, D., Rouillier, F., Wenger, P., 2012. Cusp points in the parameter space of degenerate 3-RPR planar parallel manipulators. J. Mech. Robot. 4, 041003. https://doi.org/10.1115/1.4006921.

McCallum, S., Hong, H., 2016. On using Lazard's projection in cad construction. J. Symb. Comput. 72, 65–81. https://doi.org/10.1016/j.jsc.2015.02.001.

McCallum, S., Parusiñiski, A., Paunescu, L., 2019. Validity proof of Lazard's method for CAD construction. J. Symb. Comput. 92, 52–69. https://doi.org/10.1016/j.jsc.2017.12.002.

Molnar, C., 2023. Interpretable machine learning. https://christophm.github.io/interpretable-ml-book/.

Mulligan, C.B., Davenport, J.H., England, M., 2018. TheoryGuru: a Mathematica package to apply quantifier elimination technology to economics. In: Davenport, J.H., Kauers, M., Labahn, G., Urban, J. (Eds.), Mathematical Software – Proc. ICMS 2018. Springer International Publishing, pp. 369–378. https://doi.org/10.1007/978-3-319-96418-8_44.

Paulson, L.C., 2012. Metitarski: past and future. In: Beringer, L., Felty, A. (Eds.), Interactive Theorem Proving. In: Lecture Notes in Computer Science, vol. 7406. Springer, pp. 1–10. https://doi.org/10.1007/978-3-642-32347-8_1.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., 2011. Scikit-learn: machine learning in Python. J. Mach. Learn. Res. 12, 2825–2830. http://www.jmlr.org/papers/v12/pedregosa11a.html.

Peifer, D., Stillman, M., Halpern-Leistner, D., 2020. Learning selection strategies in Buchberger's algorithm. In: Daumé III, H., Singh, A. (Eds.), Proceedings of the 37th International Conference on Machine Learning (ICML 2020). PMLR, pp. 7575–7585. https://proceedings.mlr.press/v119/peifer20a.html.

Platzer, A., Quesel, J., Rümmer, P., 2009. Real world verification. In: Schmidt, R.A. (Ed.), Automated Deduction (CADE-22). In: Lecture Notes in Computer Science, vol. 5663. Springer Berlin Heidelberg, pp. 485–501. https://doi.org/10.1007/978-3-642-02959-2_35.

del Río, T., England, M., 2022. New heuristic to choose a cylindrical algebraic decomposition variable ordering motivated by complexity analysis. In: Boulier, F., England, M., Sadykov, T.M., Vorozhtsov, E.V. (Eds.), Computer Algebra in Scientific Computing. In: Lecture Notes in Computer Science, vol. 13366. Springer International Publishing, pp. 300–317. https://doi.org/10.1007/978-3-031-14788-3_17.

del Rio, T., England, M., 2023. Data augmentation for mathematical objects. In: Ábrahám, E., Sturm, T. (Eds.), Proceedings of the 8th Workshop on Satisfiability Checking and Symbolic Computation (SC^2 2023), pp. 29–38. http://ceur-ws.org/Vol-3455/.

Röst, G., Sadeghimanesh, A., 2021. Exotic bifurcations in three connected populations with Allee effect. Int. J. Bifurc. Chaos 31. https://doi.org/10.1142/S0218127421502023.

Shapley, L.S., 1953. 17. A value for n-person games. In: Contributions to the Theory of Games (AM-28), vol. II, pp. 307–317. https://doi.org/10.1515/9781400881970-018.

Simpson, M.C., Yi, Q., Kalita, J., 2016. Automatic algorithm selection in computational software using machine learning. In: 15th IEEE International Conference on Machine Learning and Applications (ICMLA 2016), pp. 355–360. https://doi.org/10.1109/ICMLA.2016.0064.

Strzeboński, A., 2006. Cylindrical algebraic decomposition using validated numerics. J. Symb. Comput. 41, 1021–1038. https://doi.org/10.1016/j.jsc.2006.06.004.

Wilson, D., England, M., Bradford, R., Davenport, J.H., 2015. Using the distribution of cells by dimension in a cylindrical algebraic decomposition. In: Proceedings - 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2014, pp. 53–60. https://doi.org/10.1109/SYNASC.2014.15.