Math.Comput.Sci. (2024) 18:17
https://doi.org/10.1007/s11786-024-00591-0

Mathematics in Computer Science

Lessons on Datasets and Paradigms in Machine Learning for Symbolic Computation: A Case Study on CAD

Tereso del Río · Matthew England

Received: 20 February 2024 / Accepted: 19 June 2024 / Published online: 11 September 2024

Abstract Symbolic Computation algorithms and their implementation in computer algebra systems often contain choices which do not affect the correctness of the output but can significantly impact the resources required: such choices can benefit from having them made separately for each problem via a machine learning model. This study reports lessons on such use of machine learning in symbolic computation, in particular on the importance of analysing datasets prior to machine learning and on the different machine learning paradigms that may be utilised. We present results for a particular case study, the selection of variable ordering for cylindrical algebraic decomposition, but expect that the lessons learned are applicable to other decisions in symbolic computation. We utilise an existing dataset of examples derived from applications which was found to be imbalanced with respect to the variable ordering decision. We introduce an augmentation technique for polynomial systems problems that allows us to balance and further augment the dataset, improving the machine learning results by 28% and 38% on average, respectively. We then demonstrate how the existing machine learning methodology used for the problem—classification—might be recast into the regression paradigm. While this does not have a radical change on the performance, it does widen the scope in which the methodology can be applied to make choices.

Keywords Symbolic computation · Machine learning · Data augmentation · Classification · Regression · Cylindrical algebraic decomposition

Mathematics Subject Classification 68W30 · 68T05

1 Introduction

Symbolic computation algorithms, including those commonly used within theory solvers for SMT, often have within them a variety of choices to be made: choices that do not affect the correctness of the outputs, but can still have a significant effect upon the resources needed for such algorithms to return an output and on the form that such outputs take. For example, when running Buchberger's algorithm, we may choose any S-pair of polynomials from the list

M. England (✉) · T. del Río
Coventry University, Coventry, UK
e-mail: Matthew.England@coventry.ac.uk

T. del Río
e-mail: delriot@coventry.ac.uk

of those to process when computing the next S-polynomial: as long as we process all pairs in the list eventually, then we obtain a Gröbner basis of the original polynomials. However, it is well observed that the strategy utilised to make such choices can significantly impact the cost of running the algorithm, and also how the final output is presented. Historically, these choices have been made using heuristics developed by experts, such as the sugar heuristic developed to choose pairs of polynomials in Buchberger's algorithm [22]. However, not all choices are well-documented or evaluated openly in the literature.

There has been a trend in recent years for programmers to train machine learning models to make these choices. *Machine Learning* (ML) refers to statistical techniques that learn rules from data. It has been well documented that ML has outperformed expert humans in a wide range of fields, driven forward by growth in computing power, new techniques, and methods, and an explosion in available training data. It was first suggested a decade ago that ML might enhance the effectiveness of symbolic computation algorithms by replacing human-designed heuristics [26]. This could allow non-expert users of symbolic computation to optimise algorithms to their application domain, and for mathematicians to focus on the theory with less distraction from such implementation details. Further, the emergence of Explainable AI techniques may allow for ML to provide suggestions and guidance on algorithm development, even for those who do not wish to incorporate ML into their final software [34].

We will investigate two aspects of ML methodology, as introduced in the next two subsections, in the context of choices for symbolic computation. We will present experimental results for our case study problem of choosing the variable ordering for cylindrical algebraic decomposition, introduced shortly, but emphasise that the lessons drawn should be applicable more widely.

1.1 Machine Learning Paradigms

Many of these choices in symbolic computation fall naturally into the ML *classification* paradigm: where we train the ML model to choose one option from a set of discrete possibilities. An alternative paradigm is ML *regression*: where we train the ML model to estimate a continuous real numbered variable. These are often presented as distinct approaches for tackling different types of problem, but this paper notes that either may be used for optimisation of symbolic computation algorithms. Given our heuristic choice from a set of distinct possibilities, we identify the optimal one by the evaluation of some metric, e.g. the time taken for the algorithm to run with that choice, or the size of the output produced with that choice. This metric is used to identify the best choice in the classification problem; but it could also be the variable we seek to estimate in the regression problem, with the set of estimations for the different possibilities then used subsequently to make that discrete choice.

This raises the question of which paradigm is better to view such problems. Regression is certainly a more difficult problem than classification; however, for our purposes, the learning needs only to be sufficient to *rank* the discrete possibilities allowing scope for significant error in the estimations of the continuous variable without loss of performance in the symbolic computation. With that in mind, our hypothesis is that regression should be superior since it exposes more information to the ML model during training. Given a choice between n discrete possibilities, the classification paradigm treats each example as a single instance for the model to learn from, while the regression paradigm treats each example as n different instances for the model to learn from. The regression paradigm should thus allow the ML model to obverse more information: not just which choice was the best, but what effect all of the possible choices have.

1.2 Data Augmentation

Data augmentation consists of generating new data instances from existing ones. It is a widely-used technique in ML applications where a larger dataset can help tackle overfitting and increase the accuracy of the resulting model. Moreover, it can be used to mitigate biases in the dataset and to reduce the cost of labelling [36].

Data augmentation is commonly used to generate new images in Computer Vision ML applications in particular. For example, in a vision classification problem we can rotate and reflect the image without changing its label. In

our work later, we will demonstrate how these ideas of data augmentation may be used in the context of choices for symbolic computation through the permutation of the variables. In our case, the label of these new instances will change but can be easily identified from the original label: important since the labelling of data with symbolic computation will often be the most expensive part of training a ML model to make such choices.

1.3 Our Case Study Problem: Variable Ordering for CAD

This paper will focus on the case study of choosing the variable ordering for Cylindrical Algebraic Decomposition (CAD): an algorithm that decomposes the n-dimensional real space into distinct cells (connected regions of the space). It is traditionally applied on sets of polynomials to produce cells in which each of those polynomials is sign-invariant (either positive, negative, or zero throughout the cell). Such a CAD is a powerful tool for analysing and understanding the behaviour of polynomial systems, enabling us to solve tasks such as real quantifier elimination. However, the doubly exponential complexity of CAD with respect to the number of variables strongly limits its practical application. The careful study of its optimisation can only add new applications to its scope.

One particular area in need of optimisation is the variable order. This may be free or constrained, depending on the intended use of the CAD after it is created. When free, it has been observed that the choice of variable ordering can significantly impact both the theoretical complexity of CAD [9] and the practical tractability of CAD implementations [18]. Much of the experimental methodology used in this paper is replicated from our prior work on this topic, with the changes under study being the paradigm used to train the ML models and the augmenting of the dataset.

1.4 Paper Outline

Section 2 contains a literature review: we start with previous work on the use of ML in symbolic computation, then describe the CAD algorithm used as a case study in this paper and the current state-of-the-art approaches to choose its variable ordering. Section 3 then covers some common methodology used in all our experiments.

Section 4 considers the classification paradigm, replicating previous work and identifying weaknesses that arise from the imbalance of the dataset. We describe how data augmentation fixes both these issues and improves the performance of ML models. This contribution was originally presented in a paper of the SC-Square 2023 workshop [16] which this present article extends.

In Sect. 5 we move on to reframe this problem in the regression paradigm, comparing to classification, and in Sect. 6 we present an alternative formulation that is more widely applicable. Finally, in Sect. 7 we share our conclusions obtained with these experiments.

2 Literature Review

ML has been improving multiple fields over the past decades. In some safety-critical areas, such as automatic driving and medicine, ML models are not fully trusted, as they are not 100% accurate and an error could be fatal. However, they are still used to assist humans, such as when your car applies lane assist technology, or when a doctor's attention is drawn to particular cases. Similarly in mathematics, and thus symbolic computation, we usually require exact answers and so on the surface it may seem unlikely to use ML in computer algebra systems; except perhaps for drawing human attention to things of potential interest, as in [15], or where an ML-produced output can be easily checked for correctness, as in [30]. However, as discussed in the introduction, symbolic computation algorithms often contain choices that do not affect correctness but which have a significant impact on the resources needed: such choices may be safely addressed by ML and are the focus of our study. We start by reviewing the prior literature for examples of this, paying particular attention to the learning paradigm utilised, before reviewing the literature for our case study problem.

2.1 Machine Learning in Symbolic Computation

2.1.1 Traditional Classification Paradigm

Most of the examples in the literature of ML to optimise symbolic computation take the form of supervised learning with the classification paradigm: we will survey those here. *Supervised* ML means that we start with a dataset of labelled instances that we use to train the ML models. For example, a dataset of images where each image is labelled as either a cat or a dog is used to train a binary classifier to make that distinction. It is important that the data structure has some fixed features across the instances in the dataset, e.g. images of the same resolution or a representation as a fixed feature vector.

This paradigm was first used for symbolic computation by Huang et al. in 2014 [26]. They trained a support vector machine to choose the variable ordering for three-variable CAD problems (meaning six possible variable orderings); and later the same authors used this methodology to decide when to precondition a CAD input with computation of a Groebner basis [24], a binary classification. Florescu and England built on this, experimenting with models [19] and features [20].

There are further examples of supervised ML classification for computer algebra away from CAD in the literature. An early example was by Simpson et al. who used ML classifiers to pick from different algorithms to compute the resultant [37], substantially improving the runtimes in both Maple and Mathematica. Recently, Barket et al. have experimented with ML to select the order in which to try 12 possible sub-algorithms for symbolic integration in Maple [2]. They experimented with both traditional LSTMs (Long-Short Term Memory neural networks) which process sequential data and a variant, Tree LSTMs, which processed the input as expression trees. They found the latter structure beneficial, and outperformed the standard Maple mechanism to select the sub-algorithm.

ML classifiers to select the sub-algorithm for a task have also proved powerful in computational logic: where SATZilla was an early example to select a SAT-solver for a problem [40], and recently MachSMT did this for the far more diverse world of SMT [35].

Other recent examples of ML classification in computer algebra include the use of classifiers to predict features of amoebae in algebraic geometry [1], and the number of real solutions to polynomial systems [4]. However, we draw a distinction between these and the prior examples since here ML is directly predicting the output, and thus inaccuracy in the ML predictions does not just lead to inefficiencies, but to incorrect answers.

2.1.2 Modified Classification Paradigms

Brown and Daves used ML to choose a polynomial ordering for the NuCAD algorithm in [10]. This required them to be able to choose between an *a priori* unspecified number of options: i.e., it is not just that there are multiple options but that the number of options may differ at each selection point. Brown tackled this by training a binary classifier to choose between two polynomials and then having the options compete with each other under this classifier until one stands as a winner.

The work in the literature that comes closest to our ideas in Sect. 5 is that of Florescu and England [21]. They also started from the recognition that the classification paradigm was not revealing as much information to the ML models as there was available. They edited the procedure that performs cross-validation for hyperparameter selection on the classifiers so that the hyperparameters were selected based, not on classification accuracy of choices, but on the runtime achieved by those choices. In fact, this improved performance motivated our hypothesis. However, after the selection of hyperparameters was completed, the actual classifiers in [21] were still trained on accuracy as normal and so this does not represent a full use of the regression paradigm.

2.1.3 Regression and Reinforcement Learning

There are fewer examples of regression for the optimisation of computer algebra in the literature. We were particularly motivated by the work of Peifer et al. which used regression to choose S-pairs of polynomials in Buchberger's

algorithm [33]. Classification could not be used here since the number of possible S-pairs to choose from differed from one problem instance to another. They instead trained a regressor to predict the number of polynomial additions that are required after choosing a given pair, in an iteration of the main loop of Buchberger's algorithm. In this way, instead of informing the algorithm about the best option, the algorithm is informed about how much work the chosen option required.

As well as a move to regression instead of classification, this work also represents a shift from supervised to unsupervised learning. Peifer et al. did not measure the number of additions required for every possible S-pair in advance to create a dataset to learn from. Instead, their model made choices, got scored (the score being the number of additions), and amended its parameters in each iteration of the algorithm. This approach is usually known as *reinforcement learning*.[1]

We note that [33] aimed to minimise the cost of each step, in the hope that if the cheapest decision is taken at each step, then the cost of the overall algorithm will be small. There is a risk with this approach that it leads to more S-pairs being generated overall, but [33] reported strong performance, with their reinforcement learning agent outperforming all human-designed S-pair selection strategies. This motivates our own hypothesis that regression can allow for deeper learning than classification. However, unlike [33], we will not switch to reinforcement learning. We will directly compare supervised learning with classification and supervised learning with regression to see what, if any, performance difference there is.

Lastly, as the present paper was being finalised, [27] was published with the first use of reinforcement learning to choose the variable ordering for CAD. Their methodology applied this approach with graph neural networks (using the same graph embedding discussed later in Sect. 2.3.2).

2.2 Cylindrical Algebraic Decomposition

2.2.1 Definition

Cylindrical Algebraic Decomposition (CAD) is both a mathematical object and an algorithm to produce such objects, first proposed by Collins in 1975 [13]. It takes a set of polynomials $S_n \in \mathbb{R}[x_1, \ldots, x_n]$ as input, along with a variable ordering, and then builds a decomposition of $\mathbb{R}^n$ into connected regions called cells that are each sign-invariant for each of the input polynomials. Further, each cell is semi-algebraic, meaning that it may be described using polynomial constraints; and the cells are arranged in cylinders, meaning the projection of any two cells onto a lower-dimensional space (with respect to the variable ordering) is either equal or distinct. The latter condition makes membership testing, projection, and negation easy which all aid the use of CAD for quantifier elimination; the former condition allows for easy solution formula creation.

2.2.2 Algorithm

CAD algorithms usually proceed in two phases: projection and lifting. The first step of projection takes the set of polynomials S_n and returns a set of polynomials S_{n-1} without the largest variable in the ordering, x_n. In the next step of the projection phase, a set of polynomials without x_{n-1} is created. This phase continues until we reach a set of univariate polynomials in x_1.

In the first step of the lifting phase, a CAD sign-invariant for the set of polynomials S_1 is built. This may be done by using real root isolation and forming cells from the roots and intervals inbetween. In the next step, this first CAD of $\mathbb{R}$ is extended to a CAD of $\mathbb{R}^2$ sign-invariant for the set of polynomials S_2. This is achieved by taking each cell $C \in \mathbb{R}$ and choosing a single sample point $s \in C$ that we substitute into S_2 to produce univariate polynomials upon which we can apply real root isolation. We are able to conclude the sign-invariant decomposition we form for $\mathbb{R} \times s$ as representative for $\mathbb{R} \times C$, meaning the same split into cells according to roots of polynomials and the

[1] Another example of reinforcement learning is [28] which used it to learn the best pivot to use in Gaussian Elimination.

intervals inbetween throughout. The conclusion may be drawn because the information captured in the projection phase encodes the places where the behaviour of those roots changes (the notion of *delineability*). This lifting phase continues until a CAD sign-invariant decomposition of $\mathbb{R}^n$ is built.

2.2.3 Complexity

Davenport and Heinz proved that CAD generates a doubly exponential number of cells (2^{2^n}) in the worst case with respect to the number of variables (n) [14]. Brown and Davenport in [7] have shown that there is a family of sets of polynomials for which the right choice of variable ordering would yield a constant complexity and the wrong choice of variable ordering would result in a doubly exponential complexity. Numerous practical experiments have also shown the significant effect of the variable ordering choice in practice, e.g. [18,26]. This is the choice that we seek to optimise.

2.3 Heuristics for Choosing the CAD Variable Ordering

We use *heuristic* to refer to a rule we may apply to choose a CAD variable ordering. Heuristics are not guaranteed to produce an optimal answer, but they should produce an answer quickly. They are usually based on a simple feature (metric) or combination of features of the input. Although commonly designed by humans, we may also think of an ML model as a heuristic (noting that while ML models may be expensive to train, they are, once trained, quick to make a prediction on a single instance). In this section, we will present the best non-ML heuristics that have been found for our problem so far.

2.3.1 The Brown Heuristic

For many years, a well-known and widely used heuristic is the one of Brown introduced in his ISSAC 2004 tutorial notes [8]. Denoted `Brown`, this uses three simple criteria in turn, breaking ties in previous ones with the subsequent ones. For a given set of polynomials S, it projects first:

- x_i s.t. $\max_{p\in S}\left(\max_{monom\in p}\left(degree_{x_i}(monom)\right)\right)$ is minimal, (i.e. using the highest degree with which the variable appears in the polynomials); breaking ties with,
- x_i s.t. $\max_{p\in S}\left(\max_{monom\in p}\left(totaldegree_{x_i}(monom)\right)\right)$ is minimal,
  (i.e. using the highest total degree of a monomial in which the variable appears); breaking ties with
- x_i s.t. $\max_{p\in S}\left(\max_{monom\in p}\left(sign(degree_{x_i}(monom))\right)\right)$ is minimal,
  (i.e. using the number of monomials in which the variable appears).

It is not specified in [8] what to do if there is a third tie: our implementation randomly picks between the tied variables. Furthermore, whether the entire ordering of variables is chosen at once or one variable at a time taking into account intermediate projections is also not specified in [8].

2.3.2 Other Human-Designed Heuristics

There have been other human-designed heuristics since then that used more expensive algebraic information (see [5,18,39]). However, given our emphasis on heuristics that give answers quickly and the fact that they do not produce results much better than `Brown` [25], we do not consider them further here.

More recently, the authors of this article proposed the heuristic `gmods` in [17], motivated by the complexity analysis of CAD. This heuristic projects first the variable with the lowest degree in the product of the polynomials. Again, if there are ties, our implementation picks randomly.

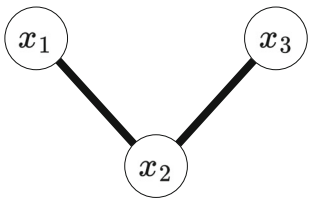

Fig. 1 Graph associated to S_G

Another recent heuristic was proposed by Li et al. (2021) [31]. This involved a graph structure associated with a polynomial set to capture sparsity information: the nodes of the graph are the variables in the polynomial set, with two nodes connected if they appear in the same polynomial. For example, the set of polynomials $S_G = \{x_1^3 x_2 - x_1 + 2, x_2^4 - x_3\}$ would be associated with the graph in Fig. 1. Their heuristic uses operations upon this graph to identify a variable ordering that seeks to preserve variable sparsity.

2.3.3 Human-Level Heuristics

The state-of-the-art human-level heuristic in the current literature is `T1`, from [34]. We call this *human-level* because although it was informed by ML techniques, it can be expressed independently of an ML model in a similar quantity of text as the human-designed heuristics above. In fact, it uses three simple features in a chain just like `Brown`. The features were those found most impactful after a SHAP analysis to interpret decisions of ML classifiers trained to choose the best variable ordering [34]. For a given set of polynomials S, this heuristic projects first:

- x_i s.t. $\sum_{p \in S} \left(\max_{monom \in p} \left(degree_{x_i}(monom) \right) \right)$ is minimal, (i.e. using the degree of the variable in the product of the polynomials); breaking ties with,
- x_i s.t. $\operatorname{aveg}_{p \in S} \left(\operatorname{aveg}_{monom \in p} \left(degree_{x_i}(monom) \right) \right)$ is minimal, (i.e. using the average of the average degree of the variable in the polynomials); breaking ties with
- x_i s.t. $\sum_{p \in S} \left(\sum_{monom \in p} \left(degree_{x_i}(monom) \right) \right)$ is minimal, (i.e. using the sum of all degrees of the variable in the polynomials).

Once again, if there is a third tie our implementation picks randomly.

3 Our Methodology for Machine Learning

In this section, we present the core of the ML methodology we will follow in all the experiments of the following sections.

3.1 Dataset

To train and test supervised ML models, one needs a dataset of labelled instances to train upon. For our purpose, each instance is a set of polynomials in three variables, labelled with the best variable ordering for CAD (the one for which a CAD could be built in the least time). We build this as follows.

- **Acquisition of sets of polynomials:** We select the 5942 three-variable problems from the `QF_NRA` category of the SMT-LIB [3], and extract the set of polynomials used from each. We acknowledge that these are all satisfiability problems and so do not represent the full application range of CAD (e.g. the more general problem of quantifier elimination). However, they do (mostly) arise from applications, making performance upon them

particularly meaningful. Most problems are generated by the theorem prover MetiTarski [32], but there are also examples coming from, e.g., biology [6] and the GeoGebra dynamic geometry tool [38].

- **CAD timings:** For each set of polynomials, `CylindricalAlgebraicDecomposition` from Maple's Regular Chains Library [11] is timed to build the CAD for all six possible variable orderings. A time limit of 30 s is used and for the cases in which no variable ordering finished within the limit, the process was repeated with a time limit of 60 s.
- **Removal of over-complicated problems:** Problems in which the construction of a CAD took more than 60 s for all six orderings are discarded, leaving 5599 problems.
- **Elimination of duplicates:** Each problem has associated with it six CADs (one for each ordering) and thus a vector of six CAD cell counts. When we identify two problems for whom these vectors are identical then we discard one problem. This happens extensively in the dataset where there are many problems that differ only by a single coefficient, having little effect on the geometry/topology of the problem. After this step, only 1019 problems are left. We note that this step is the main difference between our work and [23]: despite the latter using six times more data, they reported similar performance of the ML models trained. Aside from efficiency, this should avoid data leakage between testing and training.

3.2 Data Embedding Via Features

There are various possible embeddings that ML models can take as input (e.g. images, text, numbers), but there is no ML model known to the authors that takes a set of polynomials directly as an input. In line with prior work on ML for the CAD variable ordering problem, we take a feature-engineering approach, representing each set of polynomials by a vector of floats defined from simple-to-compute metrics on the input such as those used by `Brown` and `T1` above [12,20,26].

Florescu and England [20] proposed a semi-automated approach which allowed them to generate 1728 features for each variable in a given set of polynomials in three variables. However, many of these features were copies of each other, and there are 27 per variable which are unique. We will imitate their approach using different, hopefully more accessible, notation. Let us first introduce four different ways to convey information about a polynomial with respect to a variable. We will use the variable x_1 in the polynomial $f = 2x_1^3x_2 + x_1^2x_2x_3 + 2x_1^2x_3^3 - 3x_1 - x_2^3x_3 - 4x_3^2 + 7$ as an example.

1. Extract the degree of the variable of each monomial. E.g. $I_{1,x_1}(f) = [3, 2, 2, 1, 0, 0, 0]$.[2]
2. Extract the total degree of each monomial in which the variable appears. E.g. $I_{2,x_1}(f) = [4, 4, 5, 1]$.
3. Represent with a '1' for each monomial containing the variable and with a '0' from each monomial not containing the variable. E.g. $I_{3,x_1}(f) = [1, 1, 1, 1, 0, 0, 0]$.
4. Extract the total degree of each monomial in which the variable appears and add a zero for each monomial in which the variable does not appear. E.g. $I_{4,x_1}(f) = [4, 4, 5, 1, 0, 0, 0]$.

Information relative to a variable in a set of polynomials is then described with a vector of vectors. For example, using the first option above, the information relative to the variable x_1 in the set of polynomials $S = \{4x_1^3x_3 - x_2x_3 + 5x_3^2 - 1, f\}$ is $I_{1,x_1}(S) = [[3, 0, 0, 0], [3, 2, 2, 1, 0, 0, 0]]$.

To condense this vector of vectors down to a single feature, Florescu and England [20] proposed using the operations of maximum, summation, and average, taking inspiration again from the features in the human-design heuristic of Brown [8]. Taking the example in the previous paragraph, an option would be using summation and then average, to obtain

$$avg(sum(I_{1,x_1}(S))) = avg([sum([3, 0, 0, 0]), sum([3, 2, 2, 1, 0, 0, 0])]) = avg([3, 8]) = 5.5.$$

[2] The order of monomials in a polynomial is not really relevant but is captured by the list data-structure: however, this order does not affect any of the operations performed upon the list later.

The template that Florescu and England [20] proposed to extract features relative to the variable x_i with respect to the set of polynomials S can be simplified to

$$\underset{p\in S}{\text{operation}}\left(\underset{M\in p}{\text{operation}}\left(I_{j,x_i}(S)\right)\right)$$

where operation is either a maximum, a sum, or an average, and where $j \in \{1, 3, 4\}$.

This allowed them to extract nine features per variable for each j. We note that [20] used $I_{1,x_i}(S)$, $I_{3,x_i}(S)$ and $I_{4,x_i}(S)$ while we used $I_{1,x_i}(S)$, $I_{2,x_i}(S)$ and $I_{3,x_i}(S)$ (which we found led to better performance). This obtains 27 features for each variable x_i. The number is reduced to 26 after $max(max(I_{3,x_i}(S)))$ is eliminated (which we do since it was evaluated to 1 for every instance in our dataset: every variable x_i appears in every set of polynomials S at least once).

Finally, recall the work in [31] discussed in Sect. 2.3.2 earlier which utilised a graph embedding of the polynomial set. Inspired by their work, we add the degree of the variable in such a graph as a new feature, bringing the total back to 27 features per variable. In the example from Sect. 2.3.2 this new feature would evaluate to 1 for x_1 and x_3, and to 2 for x_2.

3.3 Dataset Preprocessing

To prepare the data for the ML models, it is common to standardise it by setting the mean of each feature to 0 and the standard deviation to 1. Standardisation of the data helps avoid biases towards certain features and improves the accuracy and effectiveness of the model, as explained by [29]. We use the function `StandardScaler` from the library `sklearn.preprocessing` for this.

3.4 Train/Test Split

We divide our dataset into five separate parts, known as "*folds*". We are careful to ensure that problems from the same source (the same directory of the SMT-LIB) end up in the same fold. This is to prevent "*data leakage*": a situation in which a model is tested using the same or similar data to the data it was trained on. We choose the hyperparameters and train models on four of these folds using the strategy being studied; then the models will be tested on the remaining fold. Repeating this until all folds have been used for testing, and taking the average to get final results. This cross-validation process ensures that our testing is fair and that a model does not benefit from the test/train split, while still ensuring that the models are evaluated on data they have not seen during the selection of its hyperparameters or its training.

3.5 Hyperparameter Tuning

In ML, "*hyperparameters*" are settings that govern the architecture and learning process of the model. Hyperparameters are set prior to training, in contrast to model parameters which are optimised during training. Examples of hyperparameters would include the number of layers in a neural network or the number of branches in a decision tree.

To find the best hyperparameters, we employ a technique called "*Bayesian Optimisation*" through the function `BayesSearchCV` in `scikit-optimize`. This function is more sophisticated than the more commonly used `GridSearchCV` which would evaluate all combinations of values for hyperparameters taken from discrete sets, and `RandomisedSearchCV` which randomly samples from such a grid. `BayesSearchCV` tries a fixed number of parameter samples from specified distributions and intelligently navigates the hyperparameter space to select these.

Additionally, instead of relying on default metrics like accuracy or mean squared error, we utilise a custom scoring function. Our custom scorer evaluates a model based on the time it takes to construct CADs using the chosen variable ordering, rather than, for example, how often the optimal variable ordering is chosen. This aligns the evaluation used to choose the hyperparameters with the specific goals of our research, similarly to [21].

3.6 ML Models Used

During our experiments, we train a variety of models to ensure that the conclusions drawn generalise. The models we use are: K-Nearest Neighbours (KNN), Random Forest (RF), Support Vector Machine (SVM), Multi-Layer Perceptron (MLP), and Extreme Gradient Boosting (XGB). For the latter we use the implementation in the `xgboost` library and for the others the implementations in the `sklearn` library. Models are trained using the `.fit()` method in the `sklearn` library with default settings, maintaining a standardised process to fairly compare the changes under scrutiny.

3.7 Testing the Models

Unless otherwise specified, the dataset used to test the different strategies is the *balanced dataset* described later in Sect. 4.3 which best ensures the fairness of the comparisons.

In this paper, we aim to find a methodology to make good choices in symbolic computation. In particular, we focus on the case study of choosing good variable orderings. But what does "*good*" stand for in this context? To measure the complexity of a CAD we use the time needed to generate it, but as it is impossible to obtain a model that guarantees choosing the optimal ordering every single time we must find ways to compare strategies that may be better in some instances but worse in others. We make use of the following metrics.

3.7.1 Number of Solved Instances

The most basic metric is the number of solved instances, i.e. the number of instances for which the strategy chose a variable ordering that did not timeout.

3.7.2 Time Accuracy

Time accuracy is the percentage of instances for which the strategy selects the fastest variable ordering to build a CAD. Accuracy is very commonly used to measure the performance of ML models in classification problems. However, it does not suit our interests very well. Imagine that the best variable ordering to build a CAD takes 12.43 s. If our strategy chooses a variable ordering that takes 12.47 s, that is still pretty good, but the accuracy measure does not see it that way. It treats this almost-right choice as completely wrong, the same way it would treat a terrible choice, even if that choice timed out. This is even worse when we consider that slight differences in the timings may be the result of computational noise.

3.7.3 Total Time

Perhaps the metric most aligned with user satisfaction is total time: the time needed to build a CAD for each instance in the testing dataset, when choosing the ordering according to the strategy being studied. Unlike accuracy, this metric treats suboptimal choices based on how suboptimal they are. In the example of Sect. 3.7.2, a choice that takes 12.43 s will be evaluated only marginally better than a choice that takes 12.47 s.

However, this metric does not fairly reflect performance in instances where building a CAD took a small amount of time for all orderings. For instance, if two variable orderings take 1 s and 4 s, respectively, to build a CAD, the

impact of this choice is equivalent to the impact of a choice between a variable ordering that took 31 s and one that took 34 s. However, you may argue that selecting an option that costs 4 instead of 1 is a much worse decision than choosing an option that costs 34 instead of one that costs 31.

Moreover, this metric is very dataset-dependent, meaning that a total time of 1000s might be a fabulous time for one dataset while being a terrible time for another dataset. Therefore, it is a poor metric for evaluating a single sample or a dataset for which the performance of other strategies is unknown. This implies that it is not the right metric to use in, say, Reinforcement Learning, where the model should be rewarded or penalised based on its performance in a group of instances.

3.7.4 Time Markup

We desire a metric that does not completely penalise suboptimal orderings while reflecting the effectiveness in fast instances. For this purpose, in [17], we proposed *time markup*. This metric is the percentage of extra time needed to build a CAD using the chosen variable ordering measured against the optimal ordering. Given a chosen suboptimal ordering with time t_{chosen} in a sample where the optimal ordering time is $t_{optimal}$ the markup is defined as

$$\frac{t_{chosen} - t_{optimal}}{t_{optimal} + 1}.$$

We add +1 to the denominator to avoid high markups for small differences in simple problems because these may be caused by computational noise rather than by the orderings themselves.

Over a dataset we define the markup as the average of the markups for all instances.

3.7.5 Metrics Based on Cell Counts

An alternative set of metrics to those in Sects. 3.7.2–3.7.4 could be defined using a count of the number of cells of a CAD rather than the time taken to compute the CAD. In theory, such metrics would be implementation independent (although in practice, all CAD implementations differ on some of the underlying theory they implement). The cell count and the time taken should be lightly correlated (at least before any post-processing of cells): we choose to focus on the time related metrics, as it is the measure of most value to users.

3.7.6 Timeout Penalisation

Note that total time and time markup metrics depend on knowing the time needed to complete a CAD using the variable ordering suggested by the strategy studied. However, as described in Sect. 3.1, CADs in Maple were called with a time limit, after which, if the call has not finished, it is stopped by Python.[3] When this happens, we do not get to know the time needed to build a CAD with such an ordering; we only know it is more than the chosen time limit. However, we need to choose a numerical value to be able to evaluate these choices. This time should be at least the time limit; however, to penalise these bad choices, in our implementation, we set the time to twice the time limit in such cases.

4 Improvements When Using the Classification Paradigm

In this section, we work with the supervised ML classification paradigm, most commonly used in the literature. First. in Sect. 4.1, the usual approach to the creation of inputs for the ML model is described. In Sect. 4.2 some

[3] We make each CAD call in a separate Maple session launched and timed by Python to avoid any instance having the benefit of Maple's previous caching of intermediate results. For example, the same resultant may be needed for two different variable orderings, and we want to ensure the cost of its computation is included in both instances.

Table 1 Hypothetical timings to build a CAD for a set of polynomials with six possible variable orderings

Ordering	Timing
$\succ_{123}$	22.16s
$\succ_{132}$	17.14s
$\succ_{213}$	Timeout 30s
$\succ_{231}$	24.87
$\succ_{312}$	**16.06**
$\succ_{321}$	22.58

problems in the created classification dataset are identified, and in Sect. 4.3 some tools are introduced to solve these problems. In Sect. 4.4 the misleading results that can be obtained if these issues are not detected are presented, and in the following subsections a series of improvements to the classification model are presented to overcome these difficulties. The first improvement solves the imbalance problem in the training dataset, the second increases the size of the training dataset, and the last reduces the number of features describing an instance while preserving most of the information these features carry. These contributions were largely presented in the SC-Square 2023 workshop paper [16].

4.1 Creating a Classification Instance

A classification instance in supervised learning consists of a pair: an input and its label. In our case, from each set of polynomials in three variables, we must find its optimal variable ordering label. We also need to create an embedding that can be taken as input by a ML model.

4.1.1 Creating the Inputs

The task here is to represent a set of polynomials in three variables as a list of features. In Sect. 3.2 we described a methodology for creating a list of features, which we will denote by $f_x(S)$ for a given variable x and a given set of polynomials S.

To represent a three-variable set of polynomials $S \in \mathbb{R}[x_1, x_2, x_3]$ to the classification model, we will therefore use the lists of features created with each variable appended to each other, placing the features related to the variable x_1 first, then those related to x_2, and finally those related to x_3.

4.1.2 Creating the Labels

The labels correspond to the possible variable orderings that a CAD can use for a set of polynomials $S \in \mathbb{R}[x_1, x_2, x_3]$. We will use as a label the variable ordering for which building a CAD took the shortest time. We will use the notation $\succ_{ijk}$ to refer to ordering $x_i \succ x_j \succ x_k$.

4.1.3 Illustrative Example

For illustration purposes, we will consider a hypothetical example that produces the timings to build a three-variable CAD indicated in Table 1. Figure 2 then illustrates how we create ML classification instances from our working example. This will serve as a comparison for the alternative methodologies presented later.

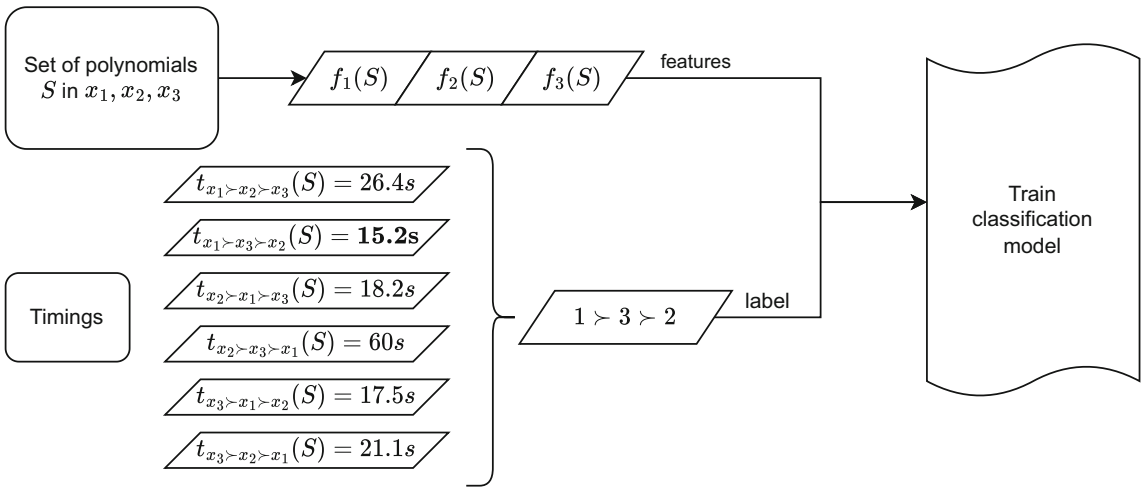

Fig. 2 Classification training workflow for the example of Sect. 4.1.3

4.1.4 Reflection on the Number of Variables

Note that the length of the feature array describing the set of polynomials and the number of labels (possible variable orderings) depends on the number of variables. Therefore, a model trained on three variable sets will not be useful for sets of polynomials with a higher number of variables. In Sect. 5 we will eliminate the dependence the output has on the number of variables, and in Sect. 6 we will further eliminate the dependence of the feature array with respect to the number of variables, creating a strategy that can be used to choose a variable ordering to build a CAD for sets of polynomials of an arbitrary number of variables.

4.2 Imbalance in the Dataset

As observed recently in [16,23] an analysis on the classification dataset described in Sect. 4.1 reveals that the dataset obtained is imbalanced: there are far more pairs of labels with ordering $\succ_{123}$ than any of the others, as can be observed in Fig. 4. Such an imbalance is a problem for ML, because models trained on an imbalanced dataset will be biased towards the majority class. Moreover, this problem may go unnoticed if the testing dataset is also imbalanced in a similar way, reporting misleading results that will not generalise when testing on a balanced dataset.

There is no reason to expect real-world data to have such an imbalance and so it is essential to balance the data prior to ML to avoid model bias. There are various approaches: e.g., we could oversample the minority classes (giving instances in the minority classes multiple times to the model), or undersample the majority classes (not giving all instances in the majority classes to the model). Instead, we use a data augmentation technique to generate more instances of the minority classes.

4.3 Improving the Datasets with Data Augmentation

As introduced in Sect. 1.2, data augmentation consists of generating new data instances from existing ones and is a common technique in computer vision.

As a motivating example, let us imagine that we have a dataset of arrows: 56 arrows pointing up, 35 left, 4 downward, and 175 right. Like our CAD dataset, this dataset is very imbalanced; any model trained on it would likely have a bias towards predicting that the arrow points to the right and against predicting that the arrow points downward.

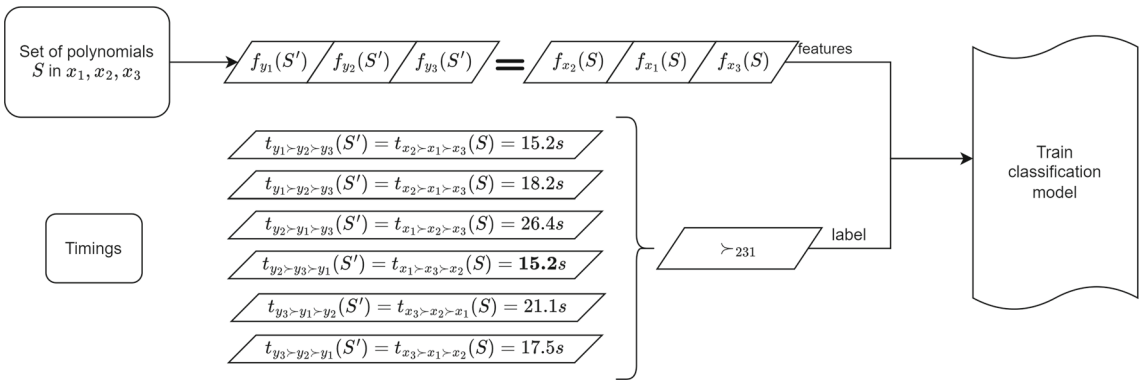

Fig. 3 Classification training workflow with data augmentation: S' has been created by renaming x_1 to y_2, x_2 to y_1 and x_3 to y_3 in S

It is clear to any human that a picture of an arrow pointing to the right that is rotated 90 degrees clockwise results gives a picture of an arrow pointing downward. Similar to this we could rotate by 180° or 270° clockwise to give arrows facing left and up, respectively. Editing images in this way can be very useful because it allows us to obtain a balanced dataset from the imbalanced dataset: simply randomly rotating each image by 0°, 90°, 180° or 270°.

Our instances consist of sets of polynomials, for example, $\{x_1^2 - x_2, x_3^3 - 1\}$ for which we can determine, by computing and comparing CADs, that the optimal variable ordering to compute a CAD for this set is $x_2 \succ x_1 \succ x_3$. Now observe that simply by swapping the names of the variables x_1 and x_2 we may obtain the new set of polynomials $\{x_2^2 - x_1, x_3^3 - 1\}$, in which we know, without any further CAD computation, that the optimal variable ordering is $x_1 \succ x_2 \succ x_3$.

This process of how an instance with a different label can be obtained from an existing instance is illustrated in Fig. 3 for our example of Sect. 4.1.3.

4.3.1 Three Datasets

We will refer to the dataset described in Sect. 3.1 as the "*imbalanced dataset*". Similar to how the arrow image dataset can be balanced by randomly rotating, our dataset can be balanced by randomly permuting the variable names: we refer to the dataset obtained this was as the "*balanced dataset*". We then go further and fully augment the dataset by including all possible variable permutations, obtaining six instances from each original one: creating the "*augmented dataset*". The distribution of instances to optimal orderings is illustrated in Figs. 4, 5 and 6.

Note that in these two new datasets, instances are first separated in folds and then either balanced or augmented. This ensures that even when augmenting the dataset there is no data leakage.

4.3.2 Other Ideas to Augment the Dataset

In computer vision, not only rotations are used as a tool to balance and augment datasets, but also other operations such as flipping or zooming.

The equivalent of flipping an image for a set of polynomials would be making changes of variables of the form $y_i = (-1)^{j_i} x_i$, where j_i is an integer for all i. This change of variables would not change the computation of the CAD and therefore the label would remain the same; however, it does not affect the features used to describe the set of polynomials; so the instances obtained by this change of variables would be just copies of existing instances.

The equivalent of zooming an image for a set of polynomials could be making changes of variables of the form $y_i = j_i x_i + k_i$, where $j_i, k_i \in \mathbb{R}$ for all i. These changes of variables do affect the features used to describe the set of polynomials, mainly because they tend to change the sparsity of the variables, one of the features used, so they would definitely create new instances. However, these changes in sparsity also affect the computations of resultants

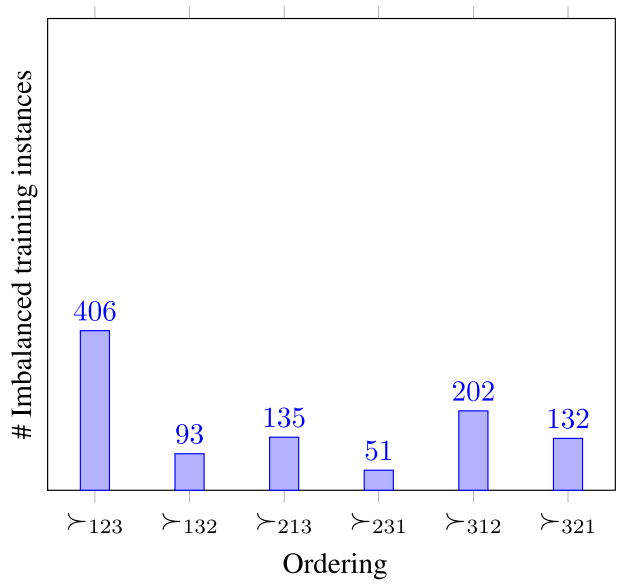

Fig. 4 Instances in the classes in the imbalanced classification dataset

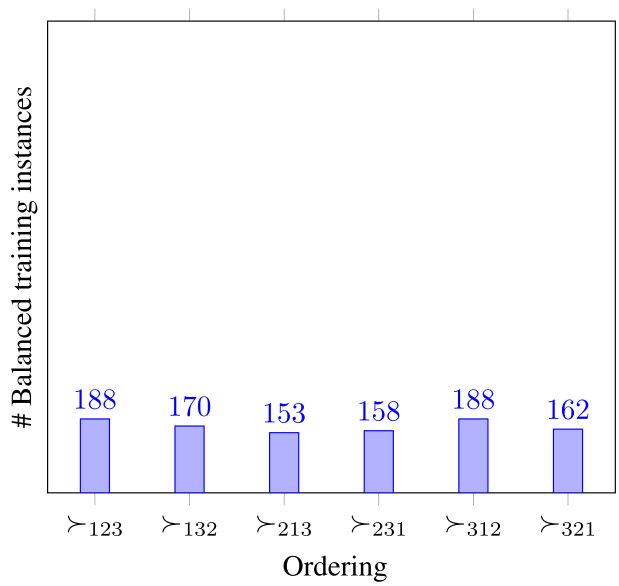

Fig. 5 Instances in the classes in the balanced classification dataset

and discriminants needed to build a CAD, so the label of the instance may change; meaning that constructing CADs for all possible orderings would be necessary to find the real label of the instance.

4.4 Replicating prior work

To replicate the work done by Florescu and England [21] we used the biased training dataset to train ML classifiers, and we tested it in the biased testing dataset. As can be seen in Fig. 7, these models perform similarly to existing heuristics, some outperforming them.

However, these models are trained on a biased dataset, and so are likely to learn to replicate these biases. The biased dataset contains a large percentage of instances with label $\succ_{123}$, and therefore, the models may simply

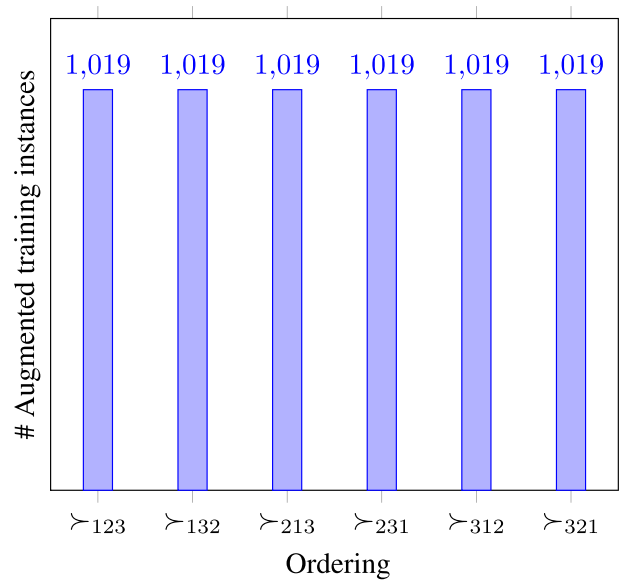

Fig. 6 Instances in the classes in the augmented classification dataset

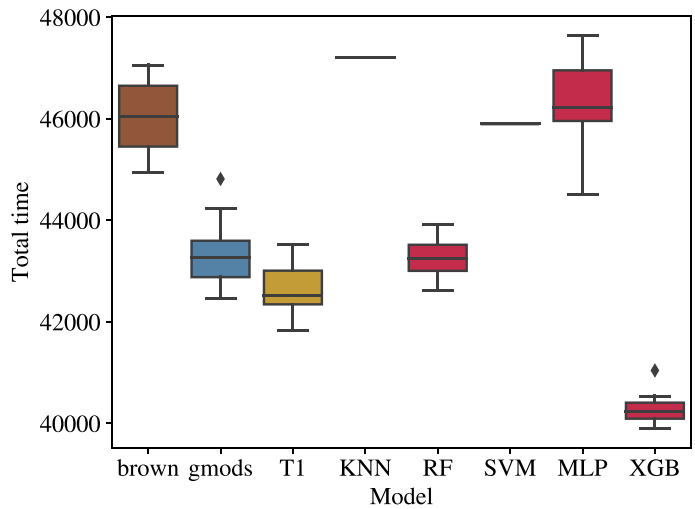

Fig. 7 Existing heuristics and ML models trained on the biased dataset, evaluated over the biased dataset. See Sect. 2.3 for the definitions of `gmods`, `brown` and `T1`, and see Sect. 3.6 for definitions of the other acronyms

learn to predict this class more often. So when these models are tested on a dataset that has the same bias, their performance may be misleading.

To identify if this is the case, we may evaluate those models trained using the biased training dataset on the balanced testing dataset. As expected, all models suffer a very significant performance drop, as illustrated in Fig. 8.

4.5 ML classifiers obtained when balancing and augmenting the training dataset

So models trained on an imbalanced dataset perform poorly on a balanced one. We will test whether using the balanced dataset for training the models will improve the performance: this should be the case since the models will no longer learn the biases they got when trained on an imbalanced dataset. We will also evaluate if using the

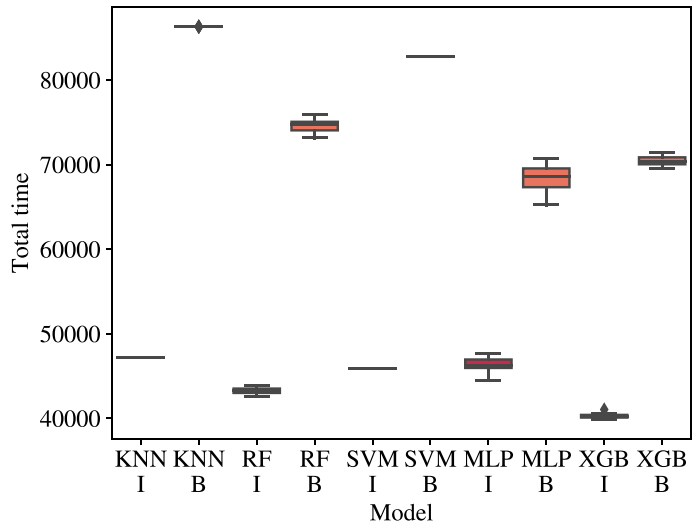

Fig. 8 ML models trained on a biased dataset tested on both imbalanced (I) and balanced (B) datasets

Table 2 Metrics over the balanced testing dataset for the KNN classifier

Training dataset	Total time	Completed	Accuracy	Markup
Imbalanced	86301	5166	0.321	1.827
Balanced	55105	5675	0.424	0.728
Augmented	51809	5751	0.509	0.614

Table 3 Metrics over the balanced testing dataset for the MLP classifier

Training dataset	Total time	Completed	Accuracy	Markup
Imbalanced	68386	5480	0.389	1.136
Balanced	48737	5793	0.478	0.481
Augmented	46066	5834	0.492	0.312

Table 4 Metrics over the balanced testing dataset for the RF classifier

Training dataset	Total time	Completed	Accuracy	Markup
Imbalanced	74551	5375	0.386	1.29
Balanced	52384	5720	0.458	0.584
Augmented	43436	5884	0.538	0.31

augmented dataset described in Sect. 4.3 offers further benefits: it has six times the number of instances of the imbalanced and balanced datasets.

The results in Tables 2, 3, 4, 5 and 6 show the performance of models trained on our three datasets when evaluated on the balanced dataset and confirm our hypotheses. Balancing the data avoided the biases and the additional data from further augmentation had a further positive effect.

Table 5 Metrics over the balanced testing dataset for the SVM classifier

Training dataset	Total time	Completed	Accuracy	Markup
Imbalanced	82743	5244	0.336	2.014
Balanced	62780	5581	0.396	1.283
Augmented	45108	5845	0.508	0.445

Table 6 Metrics over the balanced testing dataset for the XGB classifier

Training dataset	Total time	Completed	Accuracy	Markup
Imbalanced	70413	5445	0.393	1.14
Balanced	54241	5699	0.444	0.703
Augmented	47604	5816	0.496	0.433

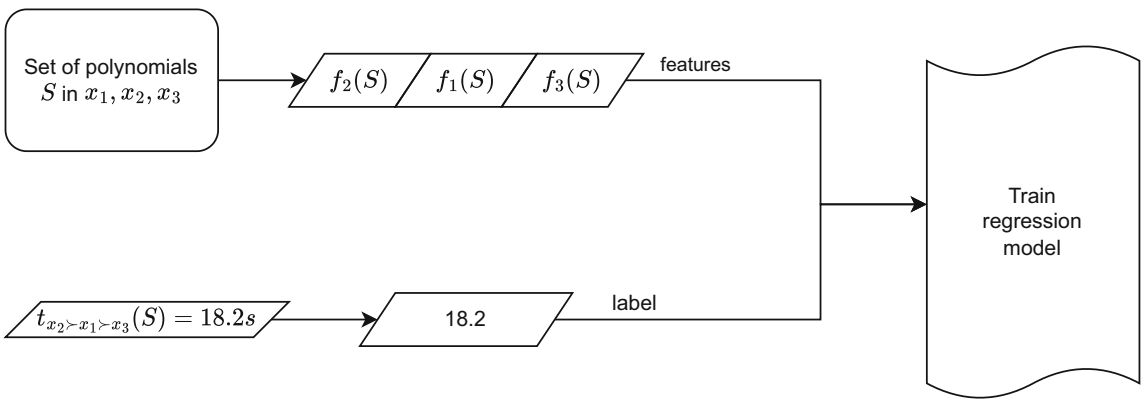

Fig. 9 Regression training workflow showing how a single regression instance is created from just 1 of the 6 orderings for the example of Sect. 4.1.3; 5 further instances may be created similarly

5 Changing from the Classification to the Regression Paradigm

As we described in Sect. 1.1, although prior work on ML to make choices for our problem has employed the classification paradigm, since the choices seek to minimise a continuous metric the problem may also be conceived as ML regression. We explore the effects of this paradigm change in this section: first describing how regression instances are created in Sect. 5.1, then presenting an experimental comparison in Sect. 5.2.

5.1 Creating a Regression Instance

A regression instance consists of a pair: an input and a target. In our case, the input is formed from the set of polynomials and variable ordering, and the target will be the time taken to build a CAD for these. The embedding that we will use to encode the input will be formed by the same three vectors of features introduced previously in Sect. 4.1.1 for classification. However here, as illustrated in Fig. 9, the order in which we append the vectors will be determined by the input variable ordering.

This embedding eliminates all existing symmetries in the classification embedding, so it is not possible to augment this dataset as we did in Sect. 4.3 for classification. However, the number of instances in the regression dataset is already as large as that of the augmented classification dataset since each classification instance becomes six

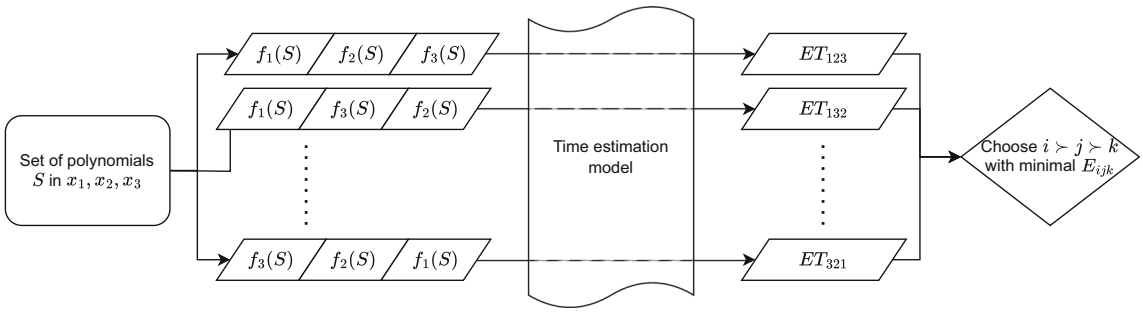

Fig. 10 Overview of how the regression model is used

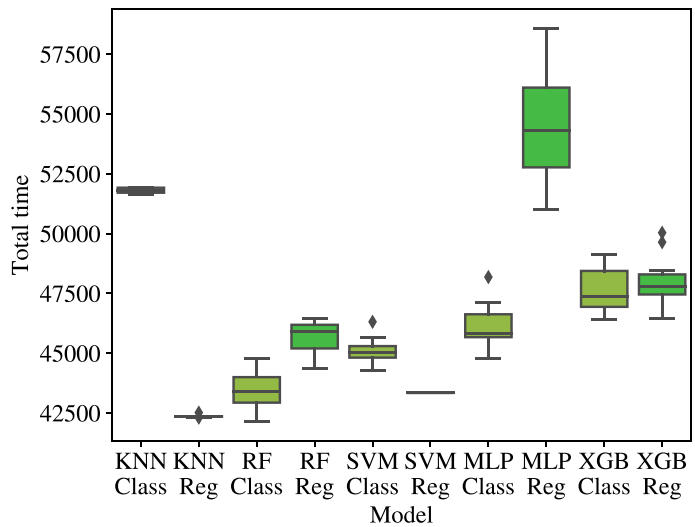

Fig. 11 Comparing the total times to build CADs for the balanced dataset with the variable ordering suggested by classifiers (Class) and regressors (Reg)

regression instances. Furthermore, the regression dataset encodes more information than the augmented classification dataset. Although instances in both datasets share the same features, their labels differ: the augmented classification dataset labels are binary, indicating only which ordering is the fastest, while the regression dataset labels contain numerical information about the exact time each ordering took.

Our ML regression models will predict how long it takes to build a CAD for a given set of polynomials using a given variable ordering. However, this is not our final goal: we are interested in choosing variable orderings to build a CAD for a given set of polynomials as fast as possible. Thus, for a given set of polynomials in the testing dataset, we must ask the regression model to estimate how long it would take to build a CAD for each of the variable orderings, and then we will choose the ordering that was estimated to take the shortest time, as illustrated in Fig. 10. This final choice can then be evaluated using the same metrics introduced in Sect. 3.7.

5.2 Results Comparing Classification with Regression

Figure 11 compares the total time required to build the CADs in our test dataset using the variable ordering suggested by each model. We see that the KNN and SVM models perform better under the regression paradigm, while the RF, MLP, and XGBoost models perform worse. Overall, the best performance on the dataset is achieved by a model under the regression paradigm: KNN. So the results partially validate our hypothesis that regression can use extra information to improve performance on the choice, but with the substantial caveat that it is model dependent.

We also reflect that the scope of choices that can be made using regression is broader than those that can be made using classification. For example, when using CAD for quantifier elimination, we must project variables in the order of quantification but still have freedom to change order within quantification blocks (and the free variables). In this case, if a classifier returns an option that does not satisfy these restrictions the recommendation is useless; however, using the regression paradigm, one can simply take the best choice that meets the restrictions.

5.3 Limitations with Respect to Number of Variables

The number of possible outputs the classification models have to choose from is factorial of the number of variables. A classification model trained for n variables cannot study a problem with more variables; and although the methodology could generalise (i.e., we may train new classifiers with a dataset of problems in more variables), the factorial growth means this will get harder quickly.

In comparison the regression models presented in this section always produced a single real number regardless of the number of variables. The difficulty in generalising to more variables will be the embedding used to input sets of polynomials into both the regression and classification models: this will still grow with the number of variables, but only linearly (27 times the number of variables).

If we want to choose variable orderings for an arbitrary number of variables, then ideally we should find an embedding that has an invariant size with respect to the number of variables. We consider one approach to this in Sect. 6, in which our model takes always a fixed-size input of only 27 features.

6 Using Regression to Rank Variables

In Sect. 5 we trained regression models to estimate how long it would take to build a CAD for a problem with a given variable ordering. In this section we will use ML to rank the variables themselves instead of the variable orderings, allowing for an easier generalisation to more variables.

This is similar to how the human-designed heuristics in Sect. 2.3 choose a variable ordering. Human heuristics assign some measure of complexity to the variables and pick the simpler variables first in the ordering, but their measure of complexity is defined by the creator of the heuristic and, therefore, is fixed. Instead, we will train ML models so that they learn their own measures of complexity, and we will pick the variables that the model considers to be simpler first in the ordering. This shares a similarity with Chen et al. (2020) [12] who also proposed to train models to choose variables instead of orderings. However, they still trained classification models and thus suffer from the limitations discussed above, being limited in their approach to sets of polynomials with less than ten variables.

Moreover, instead of choosing the whole variable ordering using only the original polynomials, once the first variable has been chosen, we will use the projected set to choose the next variable. This allows us to make our choices using the most recent information.

6.1 Creating Instances for Regression on Variables

Our instances now concern the polynomials and a single variable. For an embedding we use just the features for that variable, i.e. $f_i(S)$: the features describing variable x_i in a set of polynomials S.

In Sect. 5 we trained ML models to learn the complexity of a variable ordering and there was a clear output to use: the time necessary to build a CAD using such variable ordering. In this section, for estimating the complexity of a variable, we have available the timings of the orderings that start with that variable: in our three-variable case study there are two possible orderings that begin with each variable. We use the quickest of these to label the instance, as illustrated in Fig. 12.

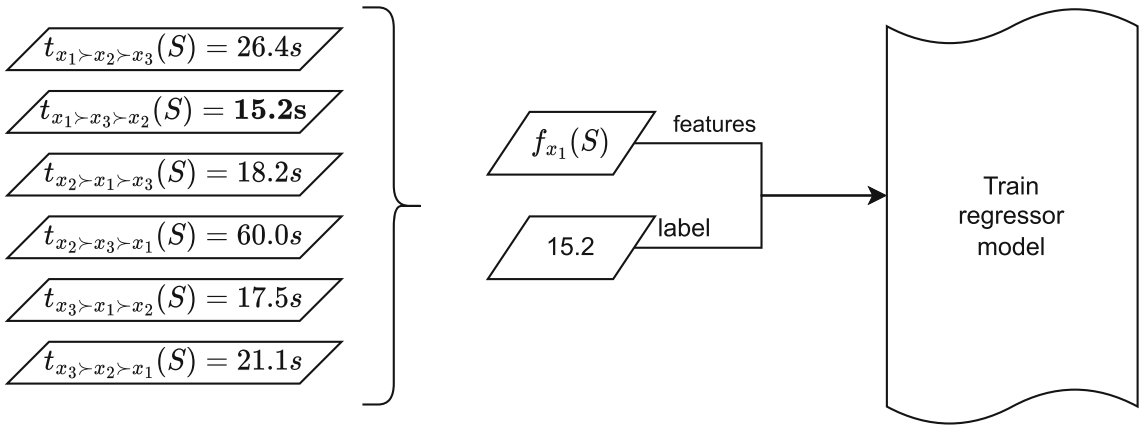

Fig. 12 Illustration of how the instance for variable x_1 is encoded for the example of Sect. 4.1.3. Note that two further instances for the other variables will be created similarly

6.2 Results

Tables 7, 8, 9, 10 and 11 compare the classification paradigm (trained with the augmented dataset), the regression paradigm on the orderings, and the regression paradigm on the variables. Interestingly, we see that the two models which did poorest under the regression paradigm to pick an ordering, MLP and XGBoost, do better under this alternative regression paradigm (while the others do worse). The lowest timing overall remains the KNN under the previous regression paradigm, but given the wider applicability of this alternative paradigm we think it worthy of consideration.

In Fig. 13 we show the spread of timings for all the key strategies presented in this paper, represented by letters as below. They show that for all models the original approach of training on an unbalanced dataset does not generalise to a balanced dataset (A vs. B); but that this performance is regained and sometimes exceeded by the new approaches of this paper (C–F). The optimal paradigm does seem to depend on the ML model type being used.

Short name	Training dataset	Testing dataset	Paradigm
A	Imbalanced (see Sect. 4.2)	Imbalanced	Classification
B	Imbalanced	Balanced	Classification
C	Balanced (see Sect. 4.3)	Balanced	Classification
D	Augmented (see Sect. 4.3)	Balanced	Classification
E	Regression orderings (see Sect. 5.1)	Balanced	Regression
F	Regression variables (see Sect. 6)	Balanced	Regression

Table 7 Metrics for strategies using a KNN model over the balanced dataset

Strategy	Total time	Completed	Accuracy	Markup
ClassAugmented	51809.2	5751	0.509	0.614
RegOrderings	42363.4	5897.7	0.519	0.312
RegVariables	52692.2	5722.5	0.481	0.554

Table 8 Metrics for strategies using a RF model over the balanced dataset

Strategy	Total time	Completed	Accuracy	Markup
ClassAugmented	43435.9	5884.2	0.538	0.31
RegOrderings	45671.5	5840.7	0.514	0.369
RegVariables	46942.2	5810.1	0.537	0.311

Table 9 Metrics for strategies using an SVM model over the balanced dataset

Strategy	Total time	Completed	Accuracy	Markup
ClassAugmented	45107.6	5844.9	0.508	0.445
RegOrderings	43343.7	5874.0	0.511	0.282
RegVariables	45568.1	5830.8	0.577	0.293

Table 10 Metrics for strategies using a MLP model over the balanced dataset

Strategy	Total time	Completed	Accuracy	Markup
ClassAugmented	46066.1	5833.5	0.492	0.312
RegOrderings	54580.0	5709.3	0.382	0.739
RegVariables	44993.0	5858.7	0.545	0.26

Table 11 Metrics for strategies using an XGB model over the balanced dataset

Strategy	Total time	Completed	Accuracy	Markup
ClassAugmented	47603.6	5816.1	0.496	0.433
RegOrderings	47916.8	5798.4	0.486	0.469
RegVariables	44462.5	5849.1	0.503	0.31

7 Final Thoughts

7.1 Conclusions on Data Balance and Augmentation

7.1.1 The Importance of Data Exploration

We found in Sect. 4 that this well known dataset was imbalanced for our problem and that if we trained classifiers with such imbalanced data they could not perform well on balanced data. This demonstrates the importance of exploring the data before moving to ML. Imbalance is not always inappropriate for ML: some applications will naturally have imbalanced data, and ML models should be aware of this. However, in our case, we seek heuristics for choosing variable orderings for CAD applied in general, and there is little rationale to suppose that general CAD applications favour one ordering over another.[4] Thus, our advice is to ensure that ML models for such applications are trained on balanced data.

[4] Except perhaps the existence of the SMT-LIB data!

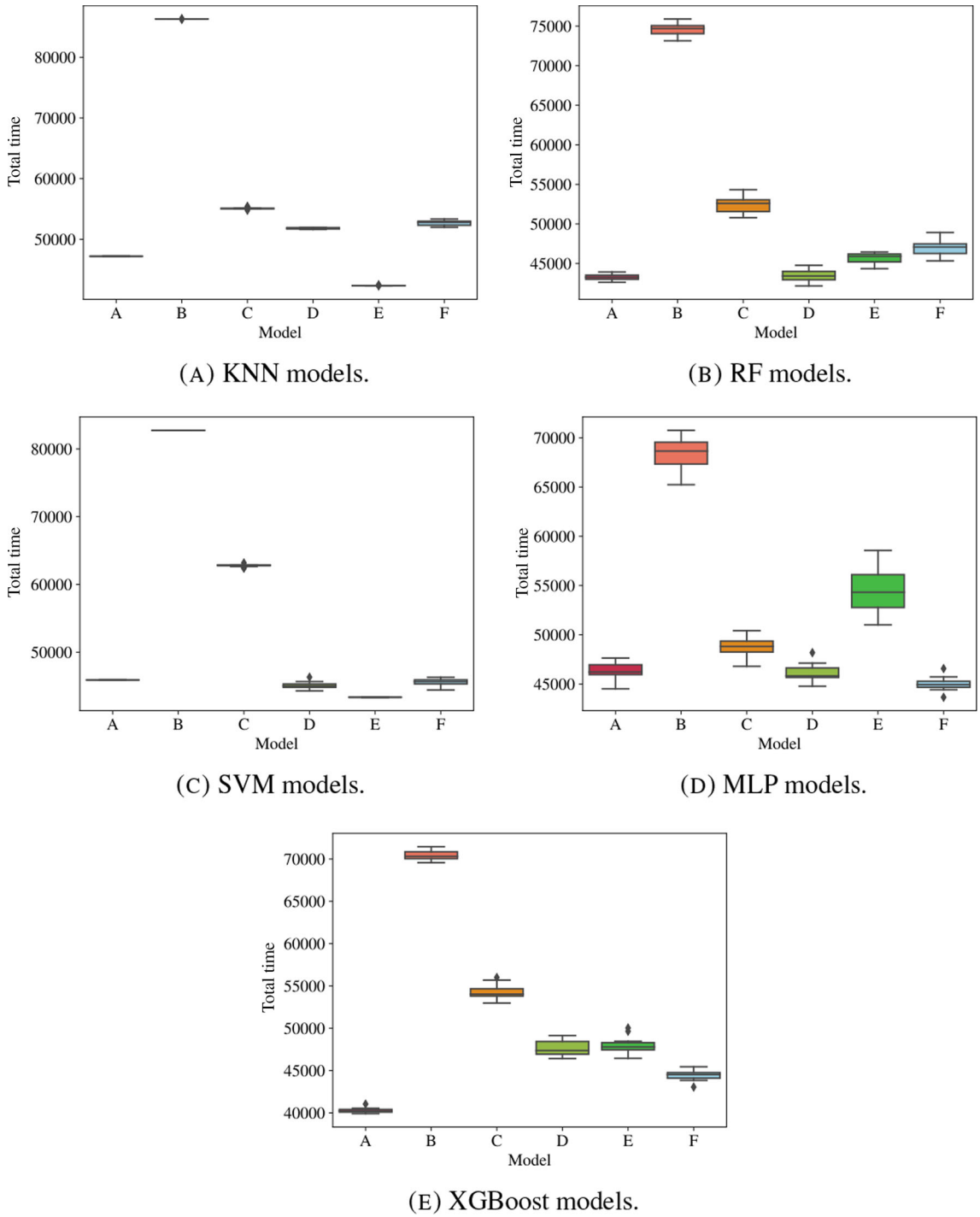

Fig. 13 The total times achieved by models on the balanced testing dataset under different strategies

7.1.2 Recovering Performance Through Balancing and Augmentation

The good news is that ML performance can be improved simply by training on balanced data, reducing the total time by 28% on average, re-validating the value of the data science-led approach. A key conclusion here is that it is beneficial to go further and use maximum data augmentation: all models benefited from this beyond just balancing the data, no matter which dataset they are tested on. Training in an augmented dataset of six times the size of the original dataset reflected in an increase in accuracy of 40% on average and a reduction of 38% in the total time on average. In fact, the performance lost from the original imbalanced case is almost fully recovered this way.

7.1.3 Comparison with the Work of Hester et al. (2023)

Some of these conclusions were also drawn in a recent paper [23]. Table 2 in [23] presents the accuracies of trained models on different datasets: note that both their "*Training Set 2*" and "*Dataset 1*" contain instances in which the models have been trained, meaning that the former is the most appropriate column for evaluation in that table: that column shows results similar to ours.

We note that the original dataset in [23] contained 6895 instances, while in our paper the initial dataset only contained 1019 instances. This is because, even though both datasets have the same ultimate source (the SMT-LIB), our dataset had been stripped of potential duplicate instances (sets of polynomials whose CAD tree structures are potentially identical for every variable ordering), as described in detail in Sect. 3.1. We view this as a necessary step to make meaningful use of the QF_NRA section of the SMT-LIB where there are many *very* similar problems.

This comparison with [23] shows that the size alone of the dataset is not what matters (since [23] has a similar accuracy to the models presented here despite training with much more data). Rather, it is the *number of qualitatively different problems within the dataset*. That is, there is little benefit to including multiple very similar problems. It may seem that data augmentation adds no new information, but since the ML models are not aware of these symmetries by exposing them with augmentation, we actually give them access to this information.

7.2 Conclusions on ML Paradigms

Despite the fact that the regression dataset contains more information than the classification one, such as the specific timings of each ordering, this did not always lead to improvement in the performance of regression models compared to classification models: it seemed that some models were better able to do this than others and this choice of paradigm should be optimised along with model and hyperparameter selection in the ML workflow.

The overall lowest timings were achieved by one of the new regression approaches. The differences in performance were not huge: as argued in Sect. 5.2, we suggest that more important than performance statistics is the wider scope of choices that can be made in symbolic computation using the new regression methodologies. Consider for example an application to S-pair selection in Buchberger's algorithm: we do not know in advance of a decision how many pairs there will be to choose from, but this may be tackled using the paradigm presented in Sect. 6, and still under a supervised learning paradigm rather than using reinforcement learning as in [33].

7.3 Future Work

A key area for future work could be the embeddings: those we use still ignore much information from the polynomials such as all coefficient information. Jia et al. (2023) [27] already proposed a graph embedding that allows inputting a lot of information about a set of polynomials into a GNN model, and they experience a performance similar to that of the heuristic gmods in the SMT-LIB dataset and even beat this heuristic in a randomly generated dataset.

Given the success of this data augmentation, an obvious area for future work is to look for additional augmentation techniques. Returning to the computer vision analogy: rotations are not the only augmentation tool; there are others

also, e.g. mirror reflections. Regarding mathematical objects, a corresponding augmentation technique may be substituting a variable with its negative, which would create a new instance without the need for any further labelling. We could also consider more involved variable transformations; however, these would most likely require additional CAD computations for data labelling, which is the most expensive part of this whole process.

Finally, we note that all these ideas would generalise easily to variable ordering choice for the other decision procedures of non-linear real arithmetic commonly found in the wider toolchains of the SC^2 community. Further, the lessons we illustrated on the importance of exploratory data analysis and paradigm choice would be useful to many other choices available in symbolic computation.

Acknowledgements Tereso del Río is supported by Coventry University and a travel grant from the London Mathematical Society (LMS). Matthew England is supported by UKRI EPSRC Grant EP/T015748/1, *Pushing Back the Doubly-Exponential Wall of Cylindrical Algebraic Decomposition* (the DEWCAD Project).

Data Availability This work in this paper formed part of Tereso del Río's PhD thesis. All code used for the experiments reported on in this paper (including to generate the results images) can be found within the data release for that thesis here: https://zenodo.org/doi/10.5281/zenodo.10834972.

References

1. Bao, J., He, Y.-H., Hirst, E.: Neurons on amoebae. J. Symb. Comput. **116**, 1–38 (2023). https://doi.org/10.1016/j.jsc.2022.08.021
2. Barket, R., England, M., Gerhard, J.: Symbolic integration algorithm selection with machine learning: LSTMs versus tree LSTMs. In: Buzzard, K., Dickenstein, A., Eick, B., Leykin, A., Ren, Y. (eds) Mathematical Software (Proc. ICMS 2024). Lecture Notes in Computer Science, vol. 14749, pp. 167–175. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-64529-7_18
3. Barrett, C., Tinelli, C.: Satisfiability modulo theories. In: Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R. (eds.) Handbook of Model Checking, pp. 305–343. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-10575-8_11
4. Bernal, E.A., Hauenstein, J.D., Mehta, D., Regan, M.H., Tang, T.: Machine learning the real discriminant locus. J. Symb. Comput. **115**, 409–426 (2023). https://doi.org/10.1016/j.jsc.2022.08.001
5. Bradford, R., Davenport, J.H., England, M., Wilson, D.: Optimising problem formulations for cylindrical algebraic decomposition. In: Carette, J., Aspinall, D., Lange, C., Sojka, P., Windsteiger, W. (eds.) Intelligent Computer Mathematics. Lecture Notes in Computer Science, vol. 7961, pp. 19–34. Springer, Berlin (2013). https://doi.org/10.1007/978-3-642-39320-4_2
6. Bradford, R., Davenport, J.H., England, M., Errami, H., Gerdt, V., Grigoriev, D., Hoyt, C., Košta, M., Radulescu, O., Sturm, T., Weber, A.: Identifying the parametric occurrence of multiple steady states for some biological networks. J. Symb. Comput. **98**, 84–119 (2020). https://doi.org/10.1016/j.jsc.2019.07.008
7. Brown, C.W.: Improved projection for cylindrical algebraic decomposition. J. Symb. Comput. **32**(5), 447–465 (2001). https://doi.org/10.1006/jsco.2001.0463
8. Brown, C.W.: Companion to the tutorial cylindrical algebraic decomposition. In: International Symposium on Symbolic and Algebraic Computation—ISSAC, pp. 1–14 (2004). https://www.usna.edu/Users/cs/wcbrown/research/ISSAC04/handout.pdf
9. Brown, C.W., Davenport, J.H.: The complexity of quantifier elimination and cylindrical algebraic decomposition. In: Proceedings of the International Symposium on Symbolic and Algebraic Computation, ISSAC, pp. 54–60 (2007). https://doi.org/10.1145/1277548.1277557
10. Brown, C.W., Daves, G.C.: Applying machine learning to heuristics for real polynomial constraint solving. In: Bigatti, A., Carette, J., Davenport, J.H., Joswig, M., de Wolff, T. (eds.) Mathematical Software–ICMS 2020. Lecture Notes in Computer Science, vol. 12097, pp. 292–301. Springer, Berlin (2020). https://doi.org/10.1007/978-3-030-52200-1_29
11. Chen, C., Moreno Maza, M.: Cylindrical algebraic decomposition in the RegularChains library. In: Hong, H., Yap, C. (eds.), Mathematical Software—ICMS 2014, Volume 8592 of Lecture Notes in Computer Science, pp. 425–433. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44199-2_65
12. Chen, C., Zhu, Z., Chi, H.: Variable ordering selection for cylindrical algebraic decomposition with artificial neural networks. In: Lecture Notes in Computer Science, Volume 12097 LNCS, pp. 281–291. Springer (2020). https://doi.org/10.1007/978-3-030-52200-1_28

13. Collins, G.E.: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. Lecture Notes in Computer Science, 33 (Proc. of the 2nd GI Conference on Automata Theory and Formal Languages), pp. 134–183 (1975). https://doi.org/10.1007/3-540-07407-4_17
14. Davenport, J.H., Heintz, J.: Real quantifier elimination is doubly exponential. J. Symb. Comput. **5**(1–2), 29–35 (1988). https://doi.org/10.1016/S0747-7171(88)80004-X
15. 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.: Advancing mathematics by guiding human intuition with AI. Nature **600**, 70–74 (2021). https://doi.org/10.1038/s41586-021-04086-x
16. del Rio, T., England, M.: 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), number 3455 in CEUR Workshop Proceedings, pp. 29–38 (2023). http://ceur-ws.org/Vol-3455/
17. del Río, T., England, M.: 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. Lecture Notes in Computer Science, vol. 13366, pp. 300–317. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-14788-3_17
18. Dolzmann, A., Seidl, A., Sturm, T.: Efficient projection orders for CAD. In: Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation—ISSAC, pp. 111–118, New York, New York, USA (2004). ACM Press. https://doi.org/10.1145/1005285.1005303
19. England, M., Florescu, D.: Comparing machine learning models to choose the variable ordering for cylindrical algebraic decomposition. In: Kaliszyk, C., Brady, E., Kohlhase, A., Sacerdoti Coen, C. (eds.) Intelligent Computer Mathematics. Lecture Notes in Computer Science, vol. 11617, pp. 93–108. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-23250-4_7
20. Florescu, D., England, M.: 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), Number 2460 in CEUR Workshop Proceedings. CEUR-WS (2019). http://ceur-ws.org/Vol-2460/
21. Florescu, D., England, M.: 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). Lecture Notes in Computer Science, vol. 11989, pp. 341–356. Springer, Berlin (2020). https://doi.org/10.1007/978-3-030-43120-4_27
22. Giovini, A., Mora, T., Niesi, G., Robbiano, L., Traverso, C.: "One sugar cube, please" or selection strategies in the Buchberger algorithm. In: Proceedings of the 1991 International Symposium on Symbolic and Algebraic Computation, ISSAC '91, pp. 49–54, New York, NY, USA (June 1991). Association for Computing Machinery. https://doi.org/10.1145/120694.120701
23. Hester, J., Hitaj, B., Passmore, G., Owre, S., Shankar, N., Yeh, E.: An augmented MetiTarski dataset for real quantifier elimination using machine learning. In: Dubois, C., Kerber, M. (eds.) Intelligent Computer Mathematics. Lecture Notes in Computer Science, pp. 297–302. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-42753-4_21
24. Huang, Z., England, M., Davenport, J.H., Paulson, L.: Using machine learning to decide when to precondition cylindrical algebraic decomposition with Groebner bases. In: 18th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC '16), pp. 45–52. IEEE (2016). https://doi.org/10.1109/SYNASC.2016.020
25. Huang, Z., England, M., Wilson, D., Bridge, J., Davenport, J.H., Paulson, L.: Using machine learning to improve cylindrical algebraic decomposition. Math. Comput. Sci. **13**(4), 461–488 (2019). https://doi.org/10.1007/s11786-019-00394-8
26. Huang, Z., England, M., Wilson, D., Davenport, J.H., Paulson, L.C., Bridge, J.: Applying machine learning to the problem of choosing a heuristic to select the variable ordering for cylindrical algebraic decomposition. In: Watt, S.M., Davenport, J.H., Sexton, A.P., Sojka, P., Urban, J. (eds), Lecture Notes in Computer Science, volume 8543 of Lecture Notes in Artificial Intelligence, pp. 92–107. Springer (2014). https://doi.org/10.1007/978-3-319-08434-3_8
27. Jia, F., Dong, Y., Liu, M., Huang, P., Ma, F., Zhang, J.: Suggesting variable order for cylindrical algebraic decomposition via reinforcement learning. In: Thirty-Seventh Conference on Neural Information Processing Systems, November (2023). https://openreview.net/forum?id=vNsdFwjPtL
28. Kauers, M., Moosbauer, J.: Good pivots for small sparse matrices. In: Boulier, F., England, M., Sadykov, T.M., Vorozhtsov, E.V. (eds.) Computer Algebra in Scientific Computing. Lecture Notes in Computer Science, vol. 12291, pp. 358–367. Springer, Berlin (2020). https://doi.org/10.1007/978-3-030-60026-6_20
29. Kuhn, M., Johnson, K.: Applied Predictive Modeling. Springer, New York (2013). https://doi.org/10.1007/978-1-4614-6849-3
30. Lample, G., Charton, F.: Deep learning for symbolic mathematics. In: Proceedings of the International Conference on Learning Representations (2020). https://doi.org/10.48550/ARXIV.1912.01412
31. Li, H., Xia, B., Zhang, H., Zheng, T.: Choosing the variable ordering for cylindrical algebraic decomposition via exploiting chordal structure. In: Proceedings of the International Symposium on Symbolic and Algebraic Computation, ISSAC, pp. 281–288 (2021). https://doi.org/10.1145/3452143.3465520
32. Paulson, L.C.: MetiTarski: past and future. In: Beringer, L., Felty, A. (eds.) Interactive Theorem Proving. Lecture Notes in Computer Science, pp. 1–10. Springer, Berlin (2012). https://doi.org/10.1007/978-3-642-32347-8_1
33. Peifer, D., Stillman, M., Halpern-Leistner, D.: Learning selection strategies in Buchberger's algorithm. In: International Conference on Machine Learning, pp. 7575–7585. PMLR (2020). https://proceedings.mlr.press/v119/peifer20a.html
34. Pickering, L., Del Rio Almajano, T., England, M., Cohen, K.: Explainable AI insights for symbolic computation: a case study on selecting the variable ordering for cylindrical algebraic decomposition. J. Symb. Comput. TBC:TBC (2024). https://doi.org/10.1016/j.jsc.2023.102276

35. Scott, J., Niemetz, A., Preiner, M., Nejati, S., Ganesh, V.: MachSMT: A Machine Learning-based Algorithm Selector for SMT Solvers. TACAS 2021: Tools and Algorithms for the Construction and Analysis of Systems, pp. 303–325 (March 2021). https://doi.org/10.1007/978-3-030-72013-1_16
36. Shorten, C., Khoshgoftaar, T.M.: A survey on image data augmentation for deep learning. J. Big Data **6**(1), 60 (2019). https://doi.org/10.1186/s40537-019-0197-0
37. Simpson, M.C., Yi, Q., Kalita, J.: Automatic algorithm selection in computational software using machine learning. In: 15th IEEE International Conference on Machine Learning and Applications, pp. 355–360 (2016). https://doi.org/10.1109/ICMLA.2016.0064
38. Vajda, R., Kovács, Z.: GeoGebra and the Realgeom reasoning tool. In: Fontaine, P., Korovin, K., Kotsireas, I.S., Rümmer, P., Tourret, S. (eds.), Proceedings of the 5th Workshop on Satisfiability Checking and Symbolic Computation (SC-Square 2020), Volume 2752 of CEUR Workshop Proceedings, pp. 204–219 (November 2020). http://ceur-ws.org/Vol-2752/
39. Wilson, D., England, M., Bradford, R., Davenport, J.H.: Using the distribution of cells by dimension in a cylindrical algebraic decomposition. In: Proceedings of the 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2014, pp. 53–60 (2015). https://doi.org/10.1109/SYNASC.2014.15
40. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: SATzilla: portfolio-based algorithm selection for SAT. J. Artif. Intell. Res. **32**, 565–606 (2008). https://doi.org/10.1613/jair.2490