Exploring Alternative Machine Learning Models for Variable Ordering in Cylindrical Algebraic Decomposition

Rohit John[0009−0005−3045−725X] and James Davenport[0000−0002−3982−7545]

University of Bath, Bath, United Kingdom

Abstract. Cylindrical Algebraic Decomposition is a computer algebra tool with many applications, from robotics to biochemistry. But it can be very sensitive to the ordering of the variables, which may be partially prescribed by the problem, but generally has at least some freedom. While various algorithmic heuristics to choose the best variable order exist, we are looking at using new machine learning models to pick the variable order directly. Of those machine learning methods we have currently implemented, Feed-forward networks seem the most successful (though some others are nearly as good), and much better than a traditional hand crafted heuristic such as Brown's. We also explore an implementation of Graph Neural Networks as well as possible data pollution in current CAD datasets.

Keywords: Machine Learning · Computer Algebra · Cylindrical Algebraic Decomposition · Variable Ordering.

1 Introduction

Cylindrical Algebraic Decomposition (CAD) is a method introduced by George Collins [5] for solving real polynomial systems and performing real quantifier elimination (QE). It segments multi-dimensional spaces into simpler regions, or cells and while introduced as a tool for quantifier elimination, it now has many uses in the resolution of complex algebraic challenges. CAD's utility spans various scientific disciplines, from robotics to epidemic modeling and the validation of economic hypotheses.

CAD comprises two primary phases: projection and lifting. The projection phase reduces the problem's dimensionality by eliminating variables in a specific order, while the lifting phase constructs the decomposition based on this variable ordering. If we are using CAD for QE, the structure of the quantifiers imposes constraints on the choice of variable ordering, but generally there are multiple options for the variable ordering. The choice of variable ordering significantly affects the CAD process's performance and complexity with [3] highlighting how different orderings can lead to vastly different computational complexities, ranging from doubly exponential to constant size. For example, CAD applied to $\exists x.ax^2 + bx + c = 0$ yields 27 cells using the ordering $a \prec b \prec c$ and 115 using

the opposite (note that we have no choice over x: it must be the first variable eliminated).

There is no known method, short of trying all possibilities, for deciding what is the absolute best variable ordering for a given problem. Various algorithmic suggestions, effectively heuristics, exist: commonly used ones are Brown's [2], sotd [6] and ndrr [1].

In such a situation, we can consider using Machine Learning, either to choose which of the algorithmic suggestions to use, or to choose the variable ordering directly. This paper will explore various models in directly selecting optimal variable orderings, as well as a new implementation of Graph Neural Networks (GNN). We will also briefly cover improvements to existing datasets of CAD problems. Fuller details are in the first author's dissertation [12][1]

2 Background

2.1 CAD and Variable Ordering

Cylindrical Algebraic Decomposition (CAD) is a foundational algorithm in computational algebraic geometry, specifically designed for decomposing multi-dimensional space into distinct, non-overlapping regions, or cells. These cells are described through semi-algebraic conditions, which means they are defined by a finite set of polynomial equations and inequalities.

The cylindrical projection property of CAD, which ensures that the projection of any two cells is either identical or disjoint, is defined with respect to a specific ordering, which determines the hierarchical projection of cells onto lower-dimensional spaces.

Let us say there are n cells in $\mathbf{R}^1$ (arranged in increasing order), and above the ith such cell there are n_i cells in $\mathbf{R}^2$ (arranged in increasing order of the new coordinate), and above the jth such cell there are $n_{i,j}$ cells in $\mathbf{R}^3$ (arranged in increasing order of the new coordinate), and so on.

Definition 1 (Implicit in [15]). *The sequence* $[n, n_1, \ldots, n_n, n_{1,1}, \ldots, n_{1,n_1}, n_{2,1}, \ldots, n_{n,n_n}, n_{1,1,1}, \ldots]$ *is called the* tree structure *of the CAD.*

If two problems P_1 *and* P_2 *are such that, for every variable ordering, the CADs for* P_1 *and* P_2 *have the same tree structures, then [15] say that* P_1 *and* P_2 *are* duplicates.

The primary function of CAD is to facilitate Quantifier Elimination (QE) in the real number domain. It does this by producing a decomposition where the truth value of a given logical formula remains consistent within each cell. This consistency allows for the transformation of complex, quantified formulas into simpler, quantifier-free versions by examining a finite number of sample points within the cells.

[1] The code for all of these models alongside datasets which will be released shortly are available on https://github.com/rohitpj/New_ML_Models_for_CAD.

The algorithm operates in two main phases: the projection phase, which reduces the dimensionality of the polynomials, and the lifting phase, which incrementally reconstructs the higher-dimensional decompositions based on the outcomes of the projection phase. This process ensures that each cell is appropriately invariant for the initial set of polynomials, maintaining constant signs, truth values or other invariants across it.

The choice of variable ordering in CAD is not merely procedural but can significantly influence the complexity and efficiency of the decomposition. This ordering affects both the theoretical and practical performance of CAD, impacting the number of cells produced and, consequently, the computational resources required for QE tasks. Over the years, various heuristics have been developed to optimize variable ordering.

There is no known algorithm (short of brute-force enumeration) for finding the absolute best ordering for a given problem, hence we need a heuristic method. Traditional heuristics like sotd [6], ndrr [1], and Brown's [2] have been developed to guide variable ordering. Although sotd and ndrr might provide more accurate results by utilizing the projection phase, their real-world efficiency is debatable due to the computational resources required. Brown's heuristic[2], based on simple statistics derived from the polynomials, strikes a balance between accuracy and resource efficiency. Additionally it has been demonstrated that most (if not all) heuristics developed can be misled, and so the need for a computationally efficient and accurate heuristic has developed. As noted by the referees, [13] and [15] introduce new heuristics which seem to outperform Brown.

2.2 Machine Learning in CAD

Machine learning (ML) encompasses a diverse set of algorithms and methods designed for creating models that learn patterns and rules from data, rather than relying on explicit programming for every decision. This approach is particularly valuable in scenarios where the relationships within the data are complex or not well-understood in advance. When we say "learn from data", what we often mean is "learn from certain developer-selected features of the data", and the choice of features is often critical.

1. Prior research by the second author's team, culminating in [9], considered the input as a set of polynomials, and had features such as "the maximum degree of x_1 among the variables" (Table 1). The dataset was problems in three variables taken from the `nlsat` dataset[3]. This was looking for a meta-heuristic, i.e. should we use sotd, ndrr or Brown's? Both sotd and ndrr need substantial additional computations to decide their recommended ordering, so we should not use them if Brown's is adequate.
2. [7] used the same dataset and set of features, but looked at direct choice of the ordering.

[2] [11] classifies "triangular" [4] with Brown's, this this seems to be a misapprehension: [4] describes a different method from that of projection/lifting.

[3] Originally at https://cs.nyu.edu/ dejan/nonlinear/.

N.B. Both the above were essentially limited to a fixed number of variables, i.e. we could train, and ask questions, about three variables (as these did), but four variables would require a fresh set of features and a fresh training set. Conversely the next approach is suited to sparser problems in any number of variables.

3. A very different approach was taken in [11], which abstracted a problem as a graph whose nodes were the variables, with an edge between x_i and x_j if there is a polynomial in the problem which contains both x_i and x_j. Graph Neural Networks (GNN) have been applied in conjunction with Reinforcement Learning in [11] to choose the variable ordering, however the authors are not aware of an approach that does not use Reinforcement Learning for solving this problem. In this GNN-based method, each variable becomes a node in the graph, and we therefore need per-variable features (see Table 2).

3 Our Methodology

We implemented several Machine Learning based models, among them Linear Regression (LR), K-Nearest Neighbours (KNN), Decision Trees (DT), Extreme Gradient-Boosting (XGB) and Feed-Forward Networks (FFN) which used both regression and classification.

Additionally we developed a new method of using Graph Neural Networks to directly select optimal orderings, and while our results were comparable to implementations that partially used GNNs, they did not hold up to other models. All of these models were used to directly select the most likely ordering.

3.1 Datasets

The issue of insufficient training data has been explored in [8] and [16]: both using similar methods to balance and augment the MetiTarski and QF_NRA datasets respectively. We will be comparing our data to [8] in this paper. By permuting the variables names in each polynomial set 6 new sets can be formed (assuming 3 variables) each with a different label.

Our main dataset used polynomial sets from [8] who we thank for their work. As stated in [8], the original MetiTarski dataset is heavily imbalanced, with the ordering (x_3,x_2,x_1) being 4 times more likely to be the optimal ordering than any of the others.

One of the minor limitations we encountered was the dataset only labelling one ordering as correct while other orderings share identical computation times and total cells. This may be leading to issues in training where correctly selected orderings may be erroneously discarded and depending on the tie-break method between orderings, can lead to to over fitting. We identify 15% of the dataset with duplicate lowest computation times.

In our experiments in data pollution we trained our models on 4 iterations of the original dataset, Unbalanced, Balanced, Augmented and Shuffled.

Let us suppose there are $5N$ problems $P_1, \ldots, P_{5N}$. We can consider these problems in various ways.

Unbalanced The original data set, where (as pointed out in [8]) (x_3, x_2, x_1) is nearly four times as likely to be the right choice, compared with (x_1, x_2, x_3). Split 80:20, i.e. $P_1, \ldots, P_{4N}/P_{4N+1}, \ldots, P_{5N}$, for training/testing.

Balanced The original data set, but each problem P_i is replaced by P'_i: the same problem under a random permutation of variables. Split 80:20 i.e. $P'_1, \ldots, P'_{4N}/P'_{4N+1}, \ldots, P'_{5N}$, for training/testing.

Augmented The original data set, but each problem P_i is replaced by six problems $P_{i,1}, \ldots, P_{i,6}$, i.e. all six permutations of the three variables. Split 80:20 i.e. $P_{1,1}, P_{1,2} \ldots, P'_{4N,6}/P'_{4N+1,1}, \ldots, P'_{5N,6}$,for training/testing.

Shuffled As Augmented, except that the data set is shuffled before being split, so each P_i would typically have 4 or 5 representatives in training, and 1 or 2 in testing.

We would echo the statement in [11]: "Besides, the lack of sufficiently large datasets is also a matter of urgency". A lot of their data was generated by use of Maple `randpoly` function, which generates, by default (which appears to be what was used), polynomials with five terms, irrespective of the number of variables. Obviously one can generate large amounts of random data but these may not be representative of actual problems.

3.2 Training and Features

Two feature sets were compared for these models, the first were used in [8] and [10] with 11 features based on the degrees and proportion of each variable in the equations.

Table 1. General Feature Set, from [10,7,8].

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

The second feature set was adapted from the one used in [11] with the 5th and 6th features being removed after performing variable importance analysis. Each variables features were concatenated for 36 total features.

Table 2. Per Variable Feature Set, from [11].

Number	Description
1	Number of other variables occurring in the same polynomials
2	Number of polynomials containing the variable
3	Maximum degree of the variable among all polynomials
4	Sum of degree of the variable among all polynomials
5	Maximum degree of all terms containing the variable
6	Sum of degree of all terms containing the variable
7	Sum of degree of leading coefficient of the variable
8	Sum of number of terms containing the variable
9	Proportion of the variable occurring in polynomials
10	Proportion of the variable occurring in terms
11	Maximum number of other variables occurring in the same term
12	Maximum number of other variables occurring in the same polynomial

Training took place using the Google Colab resource and Hex, the University of Bath's GPU cloud. In line with previous work we additionally performed hyperparameter tuning using grid-search with 5-fold cross validation.

3.3 Implementation of general models

The implementation of the commonly used Classification models (LR, KNN, DT, GBM, etc.) used the Python sklearn libraries and were relatively simple to implement. We implemented two Feed-Forward Networks for this problem, one aiming to directly classify the most optimal ranking in the same manner as the previously described models, and the other ranking the possible orderings using regression and selecting the highest ranked: a similar model was described in [14]. These models were based on TensorFlow and had identical architectures apart from the final activation functions, with softmax being used for the first model and a linear function for the second. Categorical cross-entropy was used as the loss function for the first model and mean-squared-error for the second.

3.4 Implementation of GNN models

We began by converting our polynomial sets into graph form, which could then be used to train our GNN. The approach we took has been described in [11], which viewed each variable in the set as a node, and an edge existing between two nodes if the corresponding variables appeared in the same polynomial. We experimented with two forms of GNN, mirroring the style of our FFN.

Our initial implementation aimed to classify the whole graph into one of the 6 possible orderings, however this failed to utilise the most appealing aspect of GNNs, their ability to process graphs of different sizes which would allow our model to function on polynomial sets with larger variable sizes.

We additionally explored node classification as an alternative. This method assigned each node one of the possible positions in the ordering instead of

classifying the graph as a whole, and this method performed better than the initial implementation.

3.5 Exploration of data pollution

Our initial implementation was trained and tested using the balanced_processed file from [8], however this dataset was shuffled. Simply splitting this dataset into training and testing sets would lead to permutations of polynomial sets being spread across the dataset, possibly leading to data pollution.

To correct for this issue and test the possible effects of data pollution we first sorted the dataset by file id and then split it, resulting in train/test sets with no mixing of permutations. We then compared models trained on the shuffled dataset (with testing instances removed) with models trained on our sorted dataset and a balanced dataset, which was the result of removing all but one of the permutations of every unique polynomial set in the sorted dataset. The original, unbalanced MetiTarski dataset was also trained as a comparison, however this performed very poorly.

Unfortunately this issue was discovered relatively late in the timeline of our work, and so was only tested on a subsection of our models.

4 Results

We compare our models with a random choice approach, and with Brown's heuristic. Brown's heuristic only requires information from the initial polynomials, making it a fair comparison with our heuristics, whereas sotd and ndrr perform substantial additional computations which were not published in the dataset we used. We measure the proportion of times the models predict the optimal variable ordering. GNN models were not trained on the original feature sets since they require one feature set for each node

Table 3. Comparison of Model Accuracy Across Different Feature Sets

Model	Original Features	Extended Features
Random	16.67%	16.67%
Brown	32.15%	32.15%
LR	42.12%	49.09%
KNN	54.12%	56.56%
DT	54.44%	58.48%
XGB	56.55%	59.08%
FFN Classifying	57.60%	**59.95**%
FFN Ranking	**57.67**%	59.91%
Graph GNN	—-	23.20%
Node GNN	—-	36.78%

Feed-Forward Networks performed the best on both feature sets, with both classification and regression methods achieving very similar levels of accuracy.

While every model saw an increase in accuracy when trained on the extended feature set, Linear Regression improved the most but still falls short of the more sophisticated models.

Table 4. Comparison of Model Accuracy Across Different Training Sets

Model	Augmented	Balanced	Unbalanced	Shuffled
LR	43.22	**44.67**	23.94	47.28
KNN	26.11	25.96	15.01	61.68
DT	36.48	35.35	22.41	59.10
XGB	**47.43**	40.54	**37.18**	61.49
FFN	42.64	32.7	30.09	**62.00**

In Table 4 we can clearly see the effect of training our model on a shuffled dataset since the test set contains permutations of polynomial sets which it already been trained on. When the models are trained on a sorted dataset with no mixing of permutations there is a large drop in accuracy compared to both the Shuffled model and the original models shown in Table 3. For certain models data augmentation may not be useful, and potentially even harmful. In light of the issues demonstrated in Table 4, Table 3 should be viewed as a comparison of the two feature sets.

While our implementation of GNNs were not as accurate as our other models with a test accuracy of 38%, we aim to explore this model further to ensure our implementation is not fundamentally flawed. This may be due to the training data's small graph size, possible making it difficult for the model to distinguish between variables. We will also explore the feature set used to ensure our conversion from polynomial sets to node features were correct. Their ability of GNNs to process polynomials with different numbers of variables makes them more practical for implementations as opposed to models which would have to be trained separately for each and so this an area we hope to explore further.

5 Conclusion and Future Work

In this paper we apply Machine Learning to directly select optimal variable orderings in Cylindrical Algebraic Decomposition which greatly outperform the manual heuristics, as well as develop a new GNN model to directly select orderings. We additionally demonstrate potential data pollution when training models using current datasets. Our results corroborate previous literature along with our own thesis, however more work needs to be done in turning GNNs into a viable model for selecting variable orderings.

We have obtained access to the dataset underlying [11], and intend to experiment with that as well. [16, §5.2] also points to a further problem with the MetiTarski datatset beyond that in [8]: different problems can be duplicates in the sense that they always have the same tree structure (see Definition 1). [15,

§4.1] made this point and state that it can lead to unequal valuation of selection mechanisms if one does well on a problem with many duplicates.

Acknowledgments. Davenport was partially funded by the UK's EPSRC under grant EP/T015748/1. We are grateful to Matthew England and Tereso del Río (Coventry University) for useful comments, as well as Yuhang Dong and Fuqi Jia for access to their dataset. We are grateful to the referees for many comments and drawing our attention to [13].

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Bradford, R., Davenport, J., England, M., Wilson, D.: Optimising Problem Formulation for Cylindrical Algebraic Decomposition. In: J. Carette *et al.* (ed.) Proceedings CICM 2013. pp. 19–34 (2013). https://doi.org/10.1007/978-3-642-39320-4_2
2. Brown, C.: Tutorial handout at ISSAC 2004. http://www.cs.usna.edu/ wcbrown/research/ISSAC04/handout.pdf (2004)
3. Brown, C., Davenport, J.: The Complexity of Quantifier Elimination and Cylindrical Algebraic Decomposition. In: Brown, C. (ed.) Proceedings ISSAC 2007. pp. 54–60 (2007). https://doi.org/10.1145/1277548.1277557
4. Chen, C., Davenport, J., Lemaire, F., Moreno Maza, M., Xia, B., Xiao, R., Xie, Y.: Computing the real solutions of polynomial systems with the RegularChains library in MAPLE: ISSAC 2011 Software Demo. Communications in Computer Algebra 3 **45**, 166–168 (2011). https://doi.org/10.1145/2110170.2110174
5. Collins, G.: Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition. In: Brakhage, H. (ed.) Proceedings 2nd. GI Conference Automata Theory & Formal Languages. Springer Lecture Notes in Computer Science, vol. 33, pp. 134–183 (1975). https://doi.org/10.1007/3-540-07407-4_17
6. Dolzmann, A., Seidl, A., Sturm, T.: Efficient Projection Orders for CAD. In: Gutierrez, J. (ed.) Proceedings ISSAC 2004. pp. 111–118 (2004). https://doi.org/10.1145/1005285.1005303
7. 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.) Proceedings CICM 2019. pp. 93–108 (2019). https://doi.org/10.1007/978-3-030-23250-4_7
8. 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.) Proceedings CICM 2023. Springer Lecture Notes in Computer Science, vol. 14101, pp. 297–302 (2023). https://doi.org/10.1007/978-3-031-42753-4_21
9. Huang, Z., England, M., Wilson, D., Davenport, J., Paulson, L.: Using Machine Learning to Improve Cylindrical Algebraic Decomposition. Mathematics in Computer Science **13**, 461–488 (2019). https://doi.org/10.1007/s11786-019-00394-8

10. Huang, Z., England, M., Wilson, D., Davenport, J., Paulson, L., Bridge, J.: Applying machine learning to the problem of choosing a heuristic to select the variable ordering for cylindrical algebraic decomposition. In: S.M.Watt *et al.* (ed.) Proceedings CICM 2014. pp. 92–107 (2014). https://doi.org/10.1007/978-3-319-08434-3_8
11. Jia, F., Dong, Y., Liu, M., Huang, P., Ma, F., Zhang, J.: Suggesting Variable Order for Cylindrical Algebraic Decomposition via Reinforcement Learning. NIPS **36**, 76098–76119 (2023)
12. John, R.: Exploring Alternative Machine Learning Models for Variable Ordering in Cylindrical Algebraic Decomposition (2024)
13. 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. Journal of Symbolic Computation Article 102276 **123** (2024). https://doi.org/10.1016/j.jsc.2023.102276
14. del Río, T., England, M.: Lessons on Datasets and Paradigms in Machine Learning for Symbolic Computation: A Case Study on CAD. https://arxiv.org/abs/2401.13343 (2024)
15. 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 CASC 2022. Lecture Notes in Computer Science, vol. 13366, pp. 300–317 (2022). https://doi.org/10.1007/978-3-031-14788-3_17
16. del Río, T., England, M.: Data augmentation for mathematical objects. In: Ábrahám, E., Sturm, T. (eds.) Proceedings of the 8th SC-Square Workshop, CEUR-WS Proceedings. vol. 3455, pp. 29–38 (2023), https://arxiv.org/abs/2307.06984