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

Rohit John

Master of Science in Computer Science
The University of Bath
2023/24

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

Submitted by: Rohit John

Copyright

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Bachelor of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

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. There are various algorithmic heuristics to choose the best variable order, some cheap (e.g. Brown's) and some expensive (e.g. ndrr). Much previous research has concentrated on using machine learning to select the most suitable heuristic, as well as generating the data required to train these models. 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 Brown's. We also explore an implementation of Graph Neural Networks as well as possible data pollution in current CAD datasets.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank the late Dr Alessio Guglielmi for his support and advice while he was my second marker, as well as the encouragement he gave for me to publish my work.

I am grateful to Matthew England, Tereso del Río (Coventry University) and Alicia Dickenstein for their useful and encouraging comments, as well as Yuhang Dong and Fuqi Jia for access to their dataset.

Finally I thank my family and friends for their constant encouragement and support in the duration of this project.

Chapter 1

Introduction

Cylindrical Algebraic Decomposition (CAD) is a method introduced by George Collins Collins (1975) 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 Brown and Davenport (2007) highlighting how different orderings can lead to vastly different computational complexities, ranging from doubly exponential to constant size. For example, a CAD application to $\exists$ x. $ax^2+bx+c=0$ yields 27 cells using ordering $a \prec b \prec c$ and 115 using the opposite.

There is no known method, short of trying all possibilities, for deciding what is the best variable ordering for a given problem. Various algorithmic suggestions, effectively heuristics, exist: commonly used ones are Brown's Brown (2004), sotd Dolzmann, Seidl and Sturm (2004) and ndrr Bradford et al. (2013).

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. While machine learning based heuristics exist, they are trained on polynomial sets with only 3 variables and so are limited in their capability. This paper will explore various models in directly selecting optimal variable orderings, as well as a new implementation of Graph Neural Networks (GNN) which may solve the problem mentioned above. We will also cover the possible existence of data pollution in current datasets of CAD computations as well as improvements to them.

Chapter 2

Literature and Technology Survey

2.1 Development of CAD

Cylindrical Algebraic Decomposition (CAD) is a foundational algorithm in computational algebraic geometry initially developed by George Collins in 1975 and used for solving polynomial systems and quantifier elimination. More specifically, it decomposes multidimensional 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 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 a constant sign, truth value or other invariant across it.

More formally, CAD could be defined as an algorithm that produces the following:

- A **decomposition of $\mathbf{R^n}$** is a set of cells C_i, such that $\bigcup_i C_i = \mathbf{R^n}$, and $C_i \cap C_j = \emptyset$ if $i \neq j$.
- The cells are **semi-algebraic** meaning they may be described by a finite sequence of polynomial constraints.
- The cells are **cylindrical** meaning the projection of any two cells to a lower coordinate space, in the variable ordering, are identical or disjoint. I.e., the cells in $\mathbf{R^m}$ stack up in cylinders over cells from CAD in $\mathbf{R^{m-1}}$; can project via cell description.

CAD as shown by Collins is generally built relative to a set of input polynomials such that each polynomial has a constant sign in each cell: this is called **sign-invariance**.

The primary function of CAD is to facilitate Quantifier Elimination (QE) in the real number domain. We define QE as the following -

Let $Q_i \in \{\exists, \forall\}$ be quantifiers and φ be some quantifier-free formula. Then given

$$\Phi(x_1, \dots, x_k) := Q_{k+1}x_{k+1} \dots Q_n x_n \varphi(x_1, \dots, x_n), \tag{2.1}$$

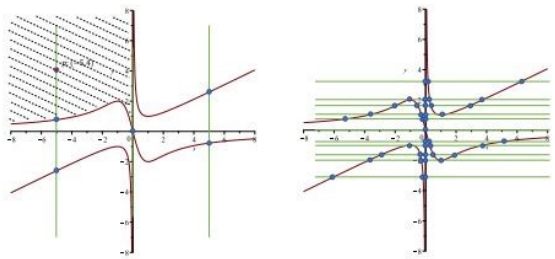
Quantifier Elimination is the problem of producing a quantifier-free formula $\psi(x_1, \dots, x_k)$ equivalent to Φ. Defined by Huang et al. (2014)

CAD efficiently implements QE 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.

Collin's initial algorithm can often results in decompositions that are significantly more complicated than necessary Wilson (2014), and a significant amount of work has been done in improving it, with notable ones including various projection operators Hong (1991), McCallum (1985), the concept of a partial and truth invariant CAD Brown (1998), the usage of equational constraints McCallum (1999) and the usage of Grobner bases for preconditioning Buchberger and Hong (1991).

2.2 Variable Ordering in CAD

The choice of variable ordering in CAD is not merely procedural but significantly influences the complexity and efficiency of the decomposition, which Brown and Davenport demonstrated using a class of problems where the use of one ordering resulted in a constant number of cells, and another a with doubly exponential number of cells Brown and Davenport (2007). 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, which we will briefly cover.



(b) $x \prec y$ (c) $y \prec x$

Figure 2.1: The difference correct variable ordering can make: Credit to Jia et al. (2023)

Brown: Browns heuristic was developed by Christopher Brown in 2004, and presented a simple yet effective method for determining optimal variable ordering Brown (2004). It chooses which variable to project first using the following where the last variable in the order is the first to be eliminated.

1. Descending order by degree of variable
2. Descending order by highest total-degree term in which the variable appears
3. Descending order by number of terms containing the variable

Sum of Total Degrees & Number of Distinct Real Roots: We will cover these together due to their shared characteristics. Both heuristics construct the full set of projection polynomials for each ordering, sotd selects the ordering with the lowest sum of total degrees for each of the monomials in each of the polynomials (Dolzmann, Seidl and Sturm (2004)) while ndrr selects the one with the lowest number of distinct real roots of the univariate polynomials Bradford et al. (2013).

There is no known algorithm (short of brute-force enumeration) for finding the best ordering for a given problem, hence we need a heuristic method. 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, based on polynomial-derived information, 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.

2.3 Machine Learning

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.

2.3.1 Classification Models

While the various ML models are suited for different tasks, in the field of classification several models have stood out as more efficient and accurate compared to the rest. This however doesn't mean certain models can't be adapted, and as we'll see this can offer certain additional advantages not immediately obvious. The following models have been commonly used, and will form part of our experiments.

- **Linear Regression:** Models the relationship between dependent and independent variables to predict outcomes, seeking the line that best fits the data by minimizing squared differences.
- **Support Vector Machine (SVM):** An algorithm that identifies the best hyperplane to separate different classes, utilizing kernel functions for linear and nonlinear data.
- **K-Nearest Neighbors (KNN):** Classifies data points based on the classifications of their k nearest neighbors, using distance metrics like Euclidean distance.
- **Decision Tree (DT):** Uses a flowchart-like structure for classification or regression, splitting data at nodes based on feature values to reach decision outcomes.
- **Gradient Boosting Machine (GBM):** Builds decision trees sequentially to correct previous errors, optimizing loss functions for regression and classification.
- **Feedforward Neural Network (FFN):** Consists of multilayer nodes connected by edges, using backpropagation to adjust weights and minimize errors.

2.3.2 Graph Neural Networks

The models described above have been implemented in previous literature as well as being commonly used for classification tasks. Graph Neural Networks (GNN) aren't as regularly used due to their less intuitive data structure (a graph), which many problems and datasets do not fit into. However, its ability to process data with variation in input size can prove to be an advantage in our case since equations are clearly not bounded to less than 3 variables. We will briefly cover the structure of GNNs and how they can be used for classification.

GNNs operate on graphs, which consist of nodes (or vertices) and edges. Each node in the graph represents an entity, and each edge represents a relationship between two nodes. The basic workflow of a GNN model involves the following steps:

- **Node Embedding Initialization**: Each node is initialized with features. These can be raw features based on the specific problem domain, such as the chemical properties of atoms in a molecule or the textual features of a document in a citation network.
- **Message Passing (Aggregation and Combination)**: For each node, gather and aggregate information from its neighbors. This aggregation function can be a sum, mean, or more complex operations like LSTM. After aggregation, the node combines the aggregated message with its own features, often using neural networks to update its embedding.
- **Readout or Pooling**: After several iterations of message passing, a readout or pooling function is applied to aggregate node features into a single graph-level representation. This is particularly used in graph classification tasks, where a single representation of the entire graph is needed.
- **Graph Representation**: The final graph representation can then be used for tasks such as classification, regression, or clustering.

Application in Graph Classification

In graph classification, the goal is to predict a label or property for entire graphs (rather than individual nodes). Since the entire graph needs to be represented in one structure, global pooling is particularly important. This step can be done in several ways:

Sum Pooling: Summing up the node embeddings.

Average Pooling: Computing the average of the node embeddings.

Max Pooling: Taking the maximum across node embeddings in each dimension.

After pooling, the graph-level representation can be fed into a fully connected layer or any other classifier to predict the graph's label.

Application in Node Classification

Node classification on the other hand aims to predict a certain label for every node in the graph. After several rounds of aggregation (message passing), instead of applying pooling, we use each nodes feature vector to make a prediction about the node's class using a simple FFN.

GNNs have been successfully applied in graph and node classification tasks across various domains. They are particularly valuable in fields like bioinformatics, social network analysis, and cheminformatics, where the intrinsic graph structure plays a crucial role in the underlying phenomena. GNN's were also used alongside Reinforcement Learning in Jia et al. (2023) in tackling the problem described above with successful results.

2.4 Machine Learning in CAD

Using these models there have been several approaches to machine learning in the field of optimising CAD. This began with Huang et al. (2014) which selected orderings using a SVM to select between Brown,sotd and ndrr which would then select an ordering. However, the dataset using in training for this paper was heavily imbalanced, and this will be an issue in other papers shown. Both sotd and ndrr need substantial additional computations to decide their recommended ordering, so if adequate, Brown was recommended be used.

Huang et al. (2016) applied another SVM to the similar problem of deciding whether to precondition Cylindrical Algebraic Decomposition with Groebner Bases with positive results.

Other previous literature England and Florescu (2019), Changbo Chen and Chi (2020) have implemented various machine learning models Models such as Support Vector Machines (SVMs), K-Nearest Neighbors (KNNs) and Multi-Layer Perceptrons (MLPs) to directly select the most optimal ordering, almost all of which beat human designed heuristics. This literature has also included the use of various feature sets, and research on algorithmically generating new features Florescu and England (2019) can generate up to 1728 features for a 3 variable problem (although only 78 were used in testing).

A significant amount of this work has used data generated using the MetiTarski theorem prover and the QF_NRA collection of the SMT-LIB library, both of which are heavily imbalanced. To remedy this issue, Hester et al. (2023) and and del Río and England (2022a) both independently proposed a new method of balancing polynomial sets by permuting the variables in each equation, thereby balancing the dataset with no change to the resulting decomposition's and no additional time taken in computing CADs. While this created a balanced dataset, there are concerns that the appearance of duplicate CAD's could lead to data pollution which we will be experimenting with.

All of these models can only process polynomial sets with 3 or less variables. Jia et al. (2023) suggests a solution to this by abstracting the polynomials 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 Jia et al. (2023) to choose the variable ordering, however we are not aware of an approach that does not use Reinforcement Learning for solving this problem. In this GNN method, each variable becomes a node in the graph, and we therefore need per-variable features

Chapter 3

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 GNN's, they did not hold up to other models.

Finally we investigated possible data pollution in the datasets used in Hester et al. (2023) and demonstrate removal of this pollution resulting in significantly lower results than originally expected, with augmenting the whole dataset performing on par with simply balancing labels.

Training took place using the Google Colab resource and Hex, the University of Bath's GPU cloud. The FFN was implemented using TensorFlow, the GNN using PyTorch Geometric library, and all other models used the Python sklearn Library.

3.1 Datasets

3.1.1 General Model Dataset

The issue of insufficient training data has been explored in Hester et al. (2023) and del Río and England (2023) both using similar methods to balance and augment the MetiTarski and QF_NRA datasets respectively. We will be comparing our data to Hester et al. (2023) in this project. As stated in Hester et al. (2023), the original MetiTarski dataset used in many papers is heavily imbalanced, with the ordering (x3,x2,x1) being 4 times more likely to be the optimal ordering than any of the others. To remedy this the authors simply permuted the variable names in each polynomial set, thereby producing a perfectly balanced dataset with each label appearing an identical number of times. This dataset consisted of 41369 polynomial sets which were based off 7200 original polynomial sets. Hester's work was accompanied with a CSV file with pre-processed features and the fastest ordering labelled from 0-5, which we used for our initial implementations, however modifications were made by generating additional features using the raw polynomial sets for better performance and data analysis.

One of the minor limitations we encountered was the dataset only labelling one ordering as most optimal (having the lowest computation time) while other permutations shared identical

computation times and total resulting 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.

Additionally in the original dataset any occasions where the the CAD computation timed out were left blank. Since we used average time taken and average cells produced as a new metric alongside accuracy, this had to be adjusted since removing any instances of time out would result in a large number of CAD's with high times and cell counts being removed, greatly skewing the results. Considering these are the CAD's that we're particularly aiming to optimise, we decided that for each instance of time-out in a particular variable ordering, to take the largest time and cell count within the 6 that didn't time out, and add 30s and 1000 cells respectively. We acknowledge that this is a relatively arbitrary measurement, however it allowed us to compare our models using more difficult CAD problems, thereby judging their efficacy more reliably.

3.1.2 Data Pollution Dataset

This began with the initial randomised dataset described in Hester et al. (2023), with shuffled permutations. We sorted our dataset by input_file, effectively reversing the shuffling and leaving a dataset where permutations of a polynomial are grouped together. We then applied train-test splits using a standard 80-20 split. To provide a better comparison to using augmentation we additionally created a dataset consisting of only one of each of the possible permutation, leaving a balanced but not augmented dataset. Finally we also used the original, unbalanced dataset to act as a baseline. In our experiments in data pollution we trained our models on 4 iterations of the original dataset, Unbalanced, Balanced, Augmented and Shuffled. We tested all of these models on a balanced version of our test set, however results are identical when testing on the original augmented test set without data pollution. We additionally removed any instances of copies between our test sets and Shuffled.

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

Unbalanced The original data set, where (as pointed out in Hester et al. (2023)) (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. $P_1, \dots, P_{4N}/P_{4N+1}, \dots, P_{5N}$, 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. $P'_1, \dots, P'_{4N}/P'_{4N+1}, \dots, P'_{5N}$, for training/testing.

Augmented The original data set, but each problem P_i is replaced by six problems $P_{i,1}, \dots, P_{i,6}$, i.e. all six permutations of the three variables. Split 80:20 i.e. $P_{1,1}, P_{1,2} \dots, P'_{4N,6}/P'_{4N+1,1}, \dots, P'_{5N,6}$,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 thank Prof. Davenport for helping us explain the dataset variations as well as formalising the definitions in writing.

3.1.3 GNN Dataset

This dataset was also based off Hester et al. (2023) but due to the need to generate graphs for each polynomial set as well as the different features used we generated the feature sets and adjacency matrices from scratch. Polynomials sets are represented in graph form with each variable being its own node, and an edge between two nodes existing where they appear in the same polynomial, first suggested in Jia et al. (2023).

3.1.4 Features used

Two feature sets were compared for these models, the first were used in Hester et al. (2023) and Huang et al. (2014) with 11 features based on the degrees and proportion of each variable in the equations.

Table 3.1: General Feature Set, from Huang et al. (2014); England and Florescu (2019); Hester et al. (2023).

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 Jia et al. (2023) with the 5th and 6th features being removed after performing feature importance analysis. Each variables features were concatenated for 36 total features.

3.2 Model Implementation & Training

We explored multiple approaches to this problem, and determining order using multi-class classification proved to be most successful in accuracy. We additionally explored using regression to predict the normalised times and selecting the ordering with the lowest which performed very well in minimising time taken, as well as a novel method of directly selecting orderings using GNNs however this proved unsatisfactory.

3.2.1 Ordering Classification

This approach has been taken in previous literature, aiming to select a single ordering as the most optimal. Being a classification problem we were able to use various commonly used models such as LR, KNN and DT using the Python sklearn library as well as the XGBoost

Table 3.2: Per Variable Feature Set, from Jia et al. (2023).

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

library. Every model described above was trained on the 2 feature sets described in Table 3.1 and 3.2 and returned 1 of the 6 possible orderings.

Our FFN was implemented using TensorFlow, with a structure consisting of 7 layers with a ReLU activation function between each one, the full structure as well as hyperparameters for all models are detailed in the appendix. A softmax function before the final layer to produce 6 probabilities for each of the 6 variable orderings, Adam was used as our optimiser and we trained the model for 50 epochs, beyond which overfitting started to apply.

3.2.2 Computation Time Prediction

Using regression for this problem was recently described in del Río and England (2024), but was also independently hypothesised and developed by the author. This method aims to assign a rank for each variable ordering based on computation time taken and selects the lowest one. Since the method does not greatly differentiate between orderings with very little time difference, any of the lowest orderings could be chosen with little impact on actual time taken, whereas a classification model when faced with a problem without a clear solution may choose randomly.

We converted our times into rankings, tie-breaking based on the corresponding index of the orderings 0-5. We chose to use the same FFN structure as described above, however a linear activation function was used for the final layer. Mean Squared Error loss was used for comparing the predictions to ground truth.

3.2.3 Graph Neural Networks

We took two approaches in implementing GNNs to this problem, classifying the graphs directly and classifying the nodes in the order in which the variables should be decomposed.

Graph Classification

This was our initial implementation and aimed to predict the whole graph into one of 6 categories which corresponded to the 6 variable orderings.

We use 3 Graph Convolutional Layers to transform our node features, applying batch normalisation in between to stabilise learning. Dropout was applied to prevent overfitting, and the nodes were aggregated to generate a single vector for the entire graph. Finally two linear layers were used to classify the graph into the six potential classes.

Node Classification

This was a new approach which aimed to classify nodes into classes which determined the order in which they should be decomposed. The neural networks structure was similar to the Graph Classification method, however instead of 6 binary outputs for each ordering, the model assigned each node a value from 0-2, where the "0" node is decomposed first and "2" last.

These two models were implemented using the PyTorch Geometric library. Graphs from the dataset were converted into the 'Data' object from pytorch_geometric by calculating the edges from the adjacency matrix, and with standard test-train splits being applied to the entire dataset.

Metrics used

We used 3 main metrics for determining the efficacy of our models.

- **Accuracy:** This simply measures the number of times the model predicted the optimal ordering divided by the total number of predictions. While this metric was easy to implement and standard among previous literature, it suffered certain drawbacks where it did not account for the difference in efficiency between orderings, i.e given resulting decomposition times of 10,11,12 and 500 the model views any ordering except from the first as equally bad. To solve this we used the following two metrics as well.
- **Average time taken:** This metric measured the average time taken to decompose a polynomial set using the chosen variable ordering, thereby avoiding the issue detailed above.
- **Average cell count:** This metric was included alongside average time taken to act as an additional comparison alongside average time taken. While more CAD permutations have duplicate lowest resulting cells compared to using time, this metric is independent of hardware limitations and so may serve as a more future-proof metric.

Chapter 4

Results

4.1 Model Comparison

We compare our models with a random choice approach and Brown's heuristic. Brown's heuristic only requires information from the initial polynomials, making it a fair comparison with our models, 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 as well as the average computation time and resulting cell count for the orderings selected. Unfortunately at the time of our conversion from raw polynomials to graphs the use of average time and average cells was not considered, and so our GNN dataset contained only graphs and labels without identifying file information, which prevented us from applying our other metrics.

Table 4.1: Model performance using original feature set

Model	Accuracy (%)	Average Time	Average Cells
LR	42.12	3.78	1760.81
KNN	54.12	2.56	1604.21
DT	54.54	2.64	1608.44
XGB	56.55	2.54	1596.80
FFN Regression	54.63	2.47	1626.87
FFN Classification	57.53	2.53	1593.68

Table 4.2: Model performance using extended feature set

Model	Accuracy (%)	Average Time	Average Cells
LR	49.09	3.01	1679.37
KNN	56.56	2.49	1591.08
DT	58.48	2.45	1588.15
XGB	59.08	2.45	1584.52
FFN Regression	54.88	2.44	1620.73
FFN Classification	59.67	2.49	1589.22
Graph Classification	23.26%	—	—
Node Classification	36.78%	—	—

Feed-Forward Networks performed the best on both feature sets, with classification achieving slightly higher than regression. However, in both feature sets using FFN for regression achieves the lowest average time and the highest average cells with the exception of LR. This is somewhat to be expected since the regression model used a time based ranking to form its predictions, and will perform better on a metric directly based on it.

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. We additionally see Decision Tree, Extreme Gradient Boosting and K-Nearest Neighbour models performing very well across both original and extended feature sets.

While our implementation of GNN's were not as accurate as our other models with a test accuracy of 36.8%, 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 GNN's 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.

4.2 Data Pollution

Table 4.3: Comparison of Model Accuracy Across Different Training Sets

Model	Augmented	Balanced	Unbalanced	Shuffled
LR	43.00	43.73	26.03	**48.30**
KNN	26.98	29.22	15.66	**60.70**
DT	34.52	33.58	23.86	**60.55**
XGB	47.50	41.84	38.80	**61.06**
FFN	39.45	31.40	29.73	**62.51**

As previously described, training our models on an unbalanced dataset results in very poor performance, which is to be expected considering the large bias in labels. Balancing the labels leads to significantly higher accuracies for certain models, however the expected increase in accuracy from Balanced to Augmented is not observed when data pollution is removed as opposed to the results shown in Hester et al. (2023) which had a 7-12 % increase for every model except from multi-layer perceptrons.

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 an augmented 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 it appears 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.

Chapter 5

Conclusions

In this paper we apply Machine Learning to directly select optimal variable orderings in Cylindrical Algebraic Decomposition, and greatly outperform 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.

As demonstrated in Table 4.3 using the Augmented MetiTarski dataset in its raw form can lead to misleading results due to data pollution. Modifications have additionally been made to the Augmented MetiTarski dataset described in Hester et al. (2023) using functions to give users the ability to easily sort it and remove data pollution, as well as fixing the issue of multiple lowest orderings not being accounted for using another function to remove instances of multiple lowest computation times or cells.

We have made this dataset publicly available at https://github.com/rohitpj/New_ML_Models_for_CAD, as well as the code for generating our datasets and code for the models described above.

5.0.1 Future Work

We have obtained access to the dataset underlying Jia et al. (2023), and intend to experiment with that as well, to ensure our GNN implementation is not limited by the problems described above, as well as the small variable sizes. Additionally we hypothesise a potential GNN model which selects a single node which is then decomposed, leaving an updated graph with the node removed. The model would then predict the next node to decompose based on the updated graph until only one node is left. While this model has the obvious drawback of requiring a pass through the GNN at every decomposition as well as the updating of the graph, it would be be able to process polynomial sets of any variable size, and the task of only having to select one node instead of classifying every node in one pass may be more suitable for a GNN.

(del Río and England, 2023, §5.2) also points to a further problem with the MetiTarski dataset beyond that in Hester et al. (2023): different problems can be duplicates in the sense that they always have the same tree structure. (del Río and England, 2022b, §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. We may explore this problem in the same manner as our original data pollution experiment.

Number of words until this point, excluding front matter: 5346.

Bibliography

Bradford, R., Davenport, J., England, M. and Wilson, D., 2013. Optimising Problem Formulation for Cylindrical Algebraic Decomposition. In: J. Carette *et al.*, ed. *Proceedings cicm 2013*. pp.19–34.

Brown, C., 1998. Simplification of truth-invariant cylindrical algebraic decomposition. In: O. Gloor, ed. *Proceedings issac '98*. pp.295–301.

Brown, C., 2004. Tutorial handout [Online]. Available from: `http://www.cs.usna.edu/~wcbrown/research/ISSAC04/handout.pdf`.

Brown, C. and Davenport, J., 2007. The Complexity of Quantifier Elimination and Cylindrical Algebraic Decomposition [Online]. In: C. Brown, ed. *Proceedings issac 2007*. pp.54–60. Available from: `https://doi.org/10.1145/1277548.1277557`.

Buchberger, B. and Hong, H., 1991. *Speeding-up Quantifier Elimination by Gröbner Bases*. (91-06).

Changbo Chen, Z.Z. and Chi, H., 2020. Variable Ordering Selection for Cylindrical Algebraic Decomposition with Artificial Neural Networks. In: A. Bigatti, J. Carette, J. Davenport, M. Joswig and T. de Wolff, eds. *Mathematical software — icms 2020*. Springer, *Springer Lecture Notes in Computer Science*, vol. 12097, pp.281–291.

Collins, G., 1975. Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition [Online]. In: H. Brakhage, ed. *Proceedings 2nd. gi conference automata theory & formal languages. Springer Lecture Notes in Computer Science*, vol. 33, pp.134–183. Available from: `https://doi.org/10.1007/3-540-07407-4_17`.

Dolzmann, A., Seidl, A. and Sturm, T., 2004. Efficient Projection Orders for CAD. In: J. Gutierrez, ed. *Proceedings issac 2004*. pp.111–118.

England, M. and Florescu, D., 2019. Comparing machine learning models to choose the variable ordering for cylindrical algebraic decomposition. *Proceedings cicm 2019*. pp.93–108.

Florescu, D. and England, M., 2019. Algorithmically generating new algebraic features of polynomial systems for machine learning. *Proc. satisfiability checking and symbolic computation 2019 ceur-ws*, 2460, pp.4:1–4:12.

Hester, J., Hitaj, B., Passmore, G., Owre, S., Shankar, N. and Yeh, E., 2023. An Augmented MetiTarski Dataset for Real Quantifier Elimination Using Machine Learning. In: C. Dubois and M. Kerber, eds. *Proceedings cicm 2023. Springer Lecture Notes in Computer Science*, vol. 14101, pp.297–302.

Hong, H., 1991. *Comparison of several decision algorithms for the existential theory of the reals*. (91-41).

Huang, Z., England, M., Davenport, J. and Paulson, L., 2016. Using Machine Learning to Decide When to Precondition Cylindrical Algebraic Decomposition With Groebner Bases. *Proceedings synasc 2016*. pp.45–52.

Huang, Z., England, M., Wilson, D., Davenport, J., Paulson, L. and Bridge, J., 2014. 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.

Jia, F., Dong, Y., Liu, M., Huang, P., Ma, F. and Zhang, J., 2023. Suggesting Variable Order for Cylindrical Algebraic Decomposition via Reinforcement Learning. *Nips*, 36, pp.76098–76119.

McCallum, S., 1985. *An Improved Projection Operation for Cylindrical Algebraic Decomposition* [Online]. (578). Computer Science University Wisconsin at Madison. Available from: `https://minds.wisconsin.edu/bitstream/handle/1793/58594/TR578.pdf?sequence=1`.

McCallum, S., 1999. On Projection in CAD-Based Quantifier Elimination with Equational Constraints. In: S. Dooley, ed. *Proceedings issac '99*. pp.145–149.

Río, T. del and England, M., 2022a. Data augmentation in mathematical objects. `https://europroofnet.github.io/_pages/WG5/Prague23/DelRio.pdf`.

Río, T. del and England, M., 2022b. New Heuristic to Choose a Cylindrical Algebraic Decomposition Variable Ordering Motivated by Complexity Analysis [Online]. In: F. Boulier, M. England, T.M. Sadykov and E.V. Vorozhtsov, eds. *Computer Algebra in Scientific Computing CASC 2022*. *Lecture Notes in Computer Science*, vol. 13366, pp.300–317. Available from: `https://doi.org/10.1007/978-3-031-14788-3_17`.

Río, T. del and England, M., 2023. Data augmentation for mathematical objects. In: E. Ábrahám and T. Sturm, eds. *Proceedings of the 8th sc-square workshop, ceur-ws proceedings*. vol. 3455, pp.29–38. Available from: `https://arxiv.org/abs/2307.06984`.

Río, T. del and England, M., 2024. Lessons on Datasets and Paradigms in Machine Learning for Symbolic Computation: A Case Study on CAD. `https://arxiv.org/abs/2401.13343`.

Wilson, D., 2014. *Advances in Cylindrical Algebraic Decomposition*. Ph.D. thesis. University of Bath.

Appendix A

Code

Please not this project was mostly developed using Google Colab, and so certain parts of the syntax represents that. To run this, please upload this folder directly onto a Google drive (not inside any folders, since this would require changes to the file paths used).

General_Models_CAD.ipynb - Contains code and tests for LR, KNN, DT, XGB and FFN classification models as well as conversion from a polluted dataset to an augmented one without pollution as well a balanced dataset. Additionally contains helper functions for the following: Remove entries with duplicate lowest times or cells, replace permutations which time out (with the highest non empty entry + 30s if time and +1000 if cells), Remove polynomial sets with any permutation which timed out.

CAD_Regression.ipynb - Contains code and tests for FFN regression models as well as the helper functions described above.

GNN_Graph_Classification.ipynb - Contains code for GNN graph classification and testing, as well as conversion from single graph files to a full graph dataset

GNN_Node_Classification.ipynb - Contains code for GNN node classification as well as conversion from single ordering labels to node labels.

extra_added_timeout_data_.csv - Shuffled dataset of poynomial features and their labels, alongside times and cells for all permutations of the polynomial sets.

metitarski_original_processed.csv - Original unbalanced dataset

extra_features_data - Full shuffled dataset for standard models extended with features derived from Jia et al. (2023)

final_full_dataset - Complete GNN dataset with features derived from Jia et al. (2023)