

New heuristic to choose a cylindrical algebraic decomposition variable ordering motivated by complexity analysis

Tereso del Río, Matthew England

Coventry University, UK

delriot@uni.coventry.ac.uk, Matthew.England@coventry.ac.uk

Abstract. It is well known that the variable ordering can be critical to the efficiency or even tractability of the cylindrical algebraic decomposition (CAD) algorithm. We propose new heuristics inspired by complexity analysis of CAD to choose the variable ordering. These heuristics are evaluated against existing heuristics with experiments on the SMT-LIB benchmarks using both existing performance metrics and a new metric we propose for the problem at hand. The best of these new heuristics chooses orderings that lead to timings on average 17% slower than the virtual-best: an improvement compared to the prior state-of-the-art which achieved timings 25% slower.

1 Introduction

1.1 Cylindrical algebraic decomposition

A Cylindrical Algebraic Decomposition (CAD) of \mathbb{R}^n is a decomposition of \mathbb{R}^n into semi-algebraic cells that are cylindrically arranged. A cell being semi-algebraic means that it can be described by polynomial constraints. CADs are defined relative to a variable ordering, for example, $x_n \succ x_{n-1} \succ \dots \succ x_1$. Then the cylindrical property means that the projections of any two cells in \mathbb{R}^n onto a subspace \mathbb{R}^i , $i < n$ with respect to this variable ordering, are either equal or disjoint. I.e. the cells in \mathbb{R}^n are arranged into cylinders above cells in \mathbb{R}^{n-1} , which are themselves arranged into cylinders above \mathbb{R}^{n-2} and so on.

It can be very useful to find such decompositions satisfying a property such as sign-invariance for an input set of polynomials (i.e. each polynomial has constant sign in each cell). The principle is that given an infinite space, a sign-invariant decomposition gives a finite set of regions on each of which our system of study has invariant behaviour, and thus can be analyzed by testing a single sample point. When such a decomposition is also cylindrical and semi-algebraic we can use it to perform tasks like quantifier elimination.

Collins in 1975 [13] was the first to propose a feasible algorithm to build such sign-invariant decompositions for a given set of polynomials. This algorithm has two phases, projection and lifting, each of them consisting of n steps, where n is the number of variables in the given set of polynomials S_n .

In the first step of the projection phase the given set of polynomials, S_n is passed to a CAD projection operator to obtain a set of polynomials S_{n-1} in $n - 1$ variables (without the biggest variable x_n). This process is iterated until a set of polynomials S_1 only in the variable x_1 is left: at this point the projection phase ends.

In the first step of the lifting phase a CAD of \mathbb{R}^1 is created by computing the t ordered roots of S_1 , denoted r_1, \dots, r_t , and building the CAD of \mathbb{R}^1 out of the cells $(-\infty, r_1), [r_1], (r_1, r_2), \dots, (r_{n-1}, r_n), [r_n], (r_n, \infty)$. Note that a sample point can be taken from each of those cells.

For each of those cells, the sample point is substituted into the set of polynomials S_2 to obtain a set of polynomials in one variable. Then using this set a stack of cells is built on top of each cell by following the instructions in the previous paragraph. These stacks are combined later into a CAD of \mathbb{R}^2 and by iterating this process a CAD of \mathbb{R}^n is eventually built, concluding the lifting phase and the algorithm.

The proof of correctness of CAD (allowing the conclusion of sign-invariance) relies on proving that the decompositions built over the sample point are representative of the behaviour over the entire cell: to conclude this the projection operator must produce polynomials whose zeros indicate where the behaviour would change. One representation of a CAD is as a tree of cells of increasing dimension; whose leaves are the cells in \mathbb{R}^n , nodes the cells in lower dimension, and branches representing the cylinders over projections.

Since the introduction of CAD by Collins, many improvements have been made to the algorithm. We do not detail them all here but refer the reader to the overview of the first 20 years in [14] and to e.g. the introduction of [3] for some of the more recent advances. We note in particular the recent developments in CAD projection [26] and the recent application of CAD technology within SMT and verification technology e.g. [24], [1] which has inspired new adaptations of the CAD algorithm such as [6]. The work of this paper is presented for traditional CAD, but we expect it would transfer easily to these recent contexts.

1.2 CAD variable ordering

It is well known that the variable ordering given can have a huge impact on the time and resources needed to build the CAD (see e.g. [15], [19], [23]). We demonstrate this for a very simple example in Figure 1, where one choice leads to three times the number of cells than the other. In fact, [7] shows that the choice of variable ordering can even change the theoretical complexity for certain classes of problems.

Depending on the application the CAD is to be used for, we have a free or constrained choice of variable ordering. For example, to use a CAD for real quantifier elimination it is necessary to project variables in the order they are quantified, but there is freedom to swap the order of variables in quantifier blocks, and also to swap the order of the free (unquantified) variables and parameters. Making use of this freedom is an important optimisation. This paper aims to



Fig. 1: CADs sign-invariant for $x^2 - y$ in the two possible orderings. The ordering $y > x$ generates the CAD on the left with only three cells (the coloured regions and the curve between). The ordering $x > y$ generates the CAD on the right with nine cells (the four coloured regions along with four line segments and the turning point of the curve).

present a new heuristic to pick the variable ordering for the construction of a CAD.

1.3 Plan of the paper

We continue in Section 2 by describing the previous heuristics developed for the problem. Then in Section 3 the proposed heuristics are presented. We then move onto our experimental evaluation: in Section 4 the methodology is detailed and in Section 5 the results obtained are analyzed. We finish with our conclusions and suggestions for future work in Section 6.

2 Previous heuristics

Due to how critical the variable ordering can be, a variety of heuristics have already been proposed for making the choice. We will focus on two of these which are widely considered to constitute the current state-of-the-art: the Brown heuristic, presented in [5], and `sotd`, presented in [15]. We describe these in full over the coming subsections.

We acknowledge there are additional human-designed heuristics in the literature, but these are either even more expensive than `sotd` e.g. [4], [27], or designed relative to a very specific CAD implementation e.g. [16].

We also acknowledge that there exists a family of machine learning methods to take this decisions e.g. [23], [19], [22], [12], [8] which have been shown to outperform the human-designed heuristics. We do not compare against these directly but note that the lessons learnt in this paper could inform another generation of these machine learnt heuristics.

2.1 The Brown heuristic

The Brown heuristic was proposed by Brown in the notes to his ISSAC 2004 tutorial [5, Section 5.2]. It chooses to project the variable with:

1. lowest degree; breaking ties with
2. lowest value of the highest total degree term in which the variable appears; breaking ties with
3. lowest number of terms containing the variable.

Should there remain ties after the third measure then [5] does not specify what to do: we name the variables x_1, x_2, \dots according to the order in which they appear in the description of the problem and our implementation of the Brown heuristic chooses the variable with the lowest subindex first.

The text in [5] is also unclear on whether these measures are applied only to the input polynomials to produce the complete ordering, or whether they are applied to select an ordering one variable at a time, each time being applied to the polynomials obtained from projection with the last variable. Our implementation does the latter: this requires no additional projection computation above that required to build a single CAD, and matches the projection computation used by our new heuristic allowing for a fair comparison later.

For an example, let us consider the set $S_3 = \{x_3^3 + x_2^3 + x_2 - x_1^4, x_2^3 - x_1\}$. In S_3 the variable x_1 has degree 4, but x_3 and x_2 both have degree 3 so they tie in the first feature. However, x_3 reaches this maximum in only one term, while x_2 does so in two of them and hence x_3 will be the first CAD projected variable. The CAD projection of S_3 with respect to x_3 is $S_2 = \{x_2^3 + x_2 - x_1^4, x_2^3 - x_1\}$. In S_2 the variable x_1 has degree 4 while x_2 has degree 3, so x_2 is the second variable to be projected. This determines the variable ordering chosen with our implementation of the Brown heuristic: $x_3 \succ x_2 \succ x_1$.

The motivation of the Brown heuristic is to try to make the next projected set of polynomials as small as possible. The heuristic is “cheap” as the measures it uses require only easy to calculate characteristics of the input polynomials, and no algebraic computations such as projection.

Moreover, the Brown heuristic has been shown to achieve similar accuracy to more complicated heuristics [23], [21], like the one introduced in the next section. This means that when we include the cost of running the heuristics themselves, the Brown heuristic is actually superior.

Nevertheless, as will be seen later, it is possible to propose better heuristics by looking at the bigger picture rather than focusing on the next projected set.

2.2 The *sotd* heuristics

The acronym *sotd* stands for ‘sum of total degrees’. The heuristic **sotd** consists of computing the whole CAD projection of the input polynomials in every possible variable ordering and choosing the ordering whose projection has the smallest sum of total degrees throughout all the monomials in all the polynomials of the projection [15].

For example, given the set of polynomials S_3 defined above and following variable ordering $x_3 \succ x_1 \succ x_2$, then we find S_2 as defined above and then projection with respect to x_1 gives $S_1 = \{x_2, x_2^2 + 1, x_2^{11} - x_2^2 - 1\}$. The sum of all the degrees in the projected sets for this ordering is 43. It turns out that this is lowest such value possible from any of the six possible orderings. Hence the **sotd** heuristic would choose this variable ordering.

This heuristic is not very desirable at first glance, because $\mathcal{O}(n!)$ projection steps are needed to make the choice, far more than the n projections used to make a single CAD. This problem was spotted by the authors of [15] leading them to develop a “greedy” version of the heuristic in the same paper. This was greedy in the sense that instead of selecting the entire ordering at once using information from the whole projection phase for each possible ordering; it computed the ordering one variable at a time by comparing their metric on the output of a single projection step for each possible variable from the remaining ones. This version only requires $\frac{n(n+1)}{2}$ number of projections, still more than the amount of projection normally used to build a CAD. In our experiments, the choices it makes are much poorer than those made by **sotd** with the full projection information. In fact, some experiments show it even performs worse than the Brown heuristic, despite having more projection information available to make its choice. We see this later in our experiments (Table 1).

The metric *sotd*, i.e. summing all the degrees in a set of polynomials, was originally constructed as a measure of the overall size of a polynomial set. In [15] it was used along with other measures to demonstrate the effect of variable ordering on CAD computation. It was found to have a strong correlation with those other CAD complexity measures, but unlike them, it did not require the computation of the entire CAD. This led to its proposal for using it as a heuristic.

Thus, it seems the **sotd** measure was not designed primarily for use in a CAD variable ordering heuristics, allowing a gap for the new results of the present paper which presents a measure that is designed this way. This lack of tailoring to CAD can be seen in the way **sotd** and its greedy version give the same importance to all degrees found in the projected sets of polynomials regardless of the variable carrying that degree, whereas CAD works iteratively one variable at a time (treating the others as part of the coefficients when projection and substituting them for the sample point when lifting). Thus different variables carry different weights of effect on CAD computation.

In the next section, it will be shown how better heuristics can be proposed which take into account the potential growth in complexity of the polynomials we observe in CAD projection.

3 Our new proposed heuristics

It has been shown in [15] that the number of cells of a CAD is strongly correlated with the time taken to build that CAD. Hence the most recent CAD complexity analyses in e.g. [17], [3], [18], [25] have studied a bound on the maximum number of cells that can be generated.

The idea of the proposed heuristic is to make a corresponding estimate on this maximum number of cells of the final CAD for each of the possible orderings and pick the ordering that minimizes this value.

We explain how this number can be computed or estimated if the whole CAD projection is known for each ordering. However, as CAD projection of polynomials can be an expensive operation, a greedy version of this heuristic that does not require any CAD projections to make the choice is also proposed.

3.1 Heuristic motivated by a complexity analysis: `mods`

Define the *degree sum* of a variable x in a set of polynomials $S = \{p_1, \dots, p_n\}$ as

$$D_x(S) = \sum_{i=1}^n d_x(p_i), \quad (1)$$

where $d_x(p)$ is the degree of x in the polynomial p . Thus the maximum number of unique real roots that the polynomials in S can have with respect to x , is $D_x(S)$.

To compute a CAD with the variable ordering $x_n \succ \dots \succ x_1$ for the set of polynomials S_n , the set S_n must be projected with respect to x_n to obtain the set S_{n-1} ; and in the same fashion the sets $S_{n-2}, S_{n-3}, \dots, S_1$ are computed.

Thus when following the creation of a CAD as described in Section 1, in the first lifting step at most $2D_{x_1}(S_1) + 1$ cells can be created because x_1 will have at most $D_{x_1}(S_1)$ roots in S_1 . Subsequently at most $(2D_{x_1}(S_1) + 1) \cdot (2D_{x_2}(S_2) + 1)$ cells will be built in the second lifting step (a similar limit applied for each stack above a cell from \mathbb{R}^1).

Hence, at the end of the lifting phase, when the CAD is completed, an upper bound on the number of cells is

$$\prod_{i=1}^n (2D_{x_i}(S_i) + 1). \quad (2)$$

As discussed earlier, the number of cells in a CAD is strongly correlated with the time needed to build such CAD. Therefore, choosing the ordering that minimizes the maximum number of cells in the final CAD sounds like a good idea if we want to choose a fast ordering. Hence, we want to choose the ordering that minimizes (2).

We note that the dominant term of (2) is

$$\prod_{i=1}^n D_{x_i}(S_i), \quad (3)$$

which we refer to as the *multiplication of degree sum (mods)*. By minimising (2) we are likely minimising this and so we refer to the heuristic that picks an ordering to minimise (2) as `mods`. As with our implementation of the Brown heuristic, we apply this to choose one variable at a time, projecting with respect

to that variable after the choice and then applying the measure to the projection polynomials to make the next choice. In case where there is a tie on the measure then we pick the variable with the lowest subindex.

Consider our example set of polynomials $S_3 = \{x_3^3 + x_2^3 + x_2 - x_1^4, x_2^3 - x_1\}$. The degree sum of x_1 in S_3 is

$$D_{x_e}(S_3) = d_{x_e}(x_3^3 + x_2^3 + x_2 - x_1^4) + d_{x_e}(x_2^3 - x_1) = 3 + 0 = 3.$$

Suppose we built a CAD using the variable ordering $x_3 \succ x_1 \succ x_2$. Then as before we obtain $S_2 = \{-x_2^3 + x_1, x_1^4 - x_2^3 - x_2\}$ for which $D_{x_1}(S_2) = 5$, and $S_1 = \{x_2, x_2^2 + 1, x_2^{11} - x_2^2 - 1\}$ for which $D_{x_2}(S_1) = 14$. Therefore, for this example CAD the product (2) evaluates to 2233. It turns out that is the lowest value of (2) for all the possible orderings. Hence `mods` would have chosen this ordering.

3.2 Creating a greedy version of `mods`

As with `sotd`, the heuristic `mods` is relatively expensive, requiring the use of CAD projection operations in all different variable orderings. To reduce its cost we present a greedy version of this heuristic, that will simply choose to project the variable with the lowest degree sum (see ((1))) in the set of polynomials¹

This heuristic will be referred to as `gmods`. Note that unlike `greedy-sotd`, `gmods` does not use any projection information beyond that required to build a single CAD. The metric it is based on uses only easily extracted information from the polynomials. It is thus similar in cost to our implementation of the Brown heuristic.

For example, given the set of polynomials S_3 above we have $D_{x_1}(S_3) = 5$, $D_{x_2}(S_3) = 6$ and $D_{x_3}(S_3) = 3$. Thus `gmods` will select x_3 as the first variable for CAD projection. The CAD projection of S_3 with respect to x_3 gives S_2 as above. In S_2 the variable x_1 has degree sum 5 while x_2 has degree sum 6, so x_1 is the second variable to be projected, determining completely the variable ordering that `gmods` chooses: $x_3 \succ x_1 \succ x_2$.

3.3 Heuristic motivated by expected number of cells

Our `mods` heuristic is motivated to reduce the maximum number of cells that could be computed according to a complexity analysis. It is natural to ask whether we could be more accurate and seek take decisions according to an expected value of the number of cells rather than the maximum?

To calculate the maximum number of cells that can be generated, the degree of the polynomials has been used because it is the maximum number of real roots that a polynomial can have, so we may consider the expected number of

¹ We note that this measure applied only to the original polynomials is one of the features that was generated algorithmically to train different machine learning classifiers to take decisions on sets of polynomials in [21].

roots of a polynomial. According to [20], the expected number of real roots for polynomials of small degree is proportional to the logarithm of its degree, at least for their definition of random polynomials. However, for a linear polynomial, this relation would predict zero roots when it should be one, and so to address this we suggest a heuristic following this approach should add one before taking the logarithm.

Thus we hypothesise an expected number of cells in the final CAD as below (following the approach of Section 3.1):

$$\prod_{i=1}^n (2 \log(D_{x_i}(S_i) + 1) + 1). \quad (4)$$

As before, we define a heuristic to pick the ordering that minimizes (4). Given the similarity to `mods` and the use of the logarithm we refer to this as `logmods`.

For example, consider the set of polynomials S_3 as before and the variable ordering $x_3 \succ x_1 \succ x_2$ to produce S_2 and S_1 as before. We find $D_{x_3}(S_3) = 3$, $D_{x_1}(S_2) = 5$, and $D_{x_2}(S_1) = 14$. Therefore, (4) evaluates to 15.43, and it turns out that is the lowest value for all possible orderings, hence, `logmods` would choose this variable ordering.

4 Experiments and benchmarking

4.1 Benchmarking

The three-variable problems in the `QF_NRA` category of the SMT-LIB [2] are used to build a dataset for comparing the different heuristics.

For each of those problems and all possible orderings, we timed (see Section 4.1) how long it takes to build a sign-invariant CAD for the polynomials involved, discarding the problems in which the creation of the CAD timed out for all orderings. After building all the possible CADs, a dataset of “unique” problems is created (see Section 4.1).

Of the 5942 original problems, in 343 of them, all the orderings timed out. And out of the remaining 5599 problems, only 1019 unique problems were found. These 1019 problems will be used as benchmarks to compare the heuristics presented in Sections 2 and 3.

CAD Implementation For our experiments we used the function `CylindricalAlgebraicDecompose` in the MAPLE 2022 Library `RegularChains`, whose implementation is described in [10]. This actually implements a somewhat different CAD algorithm to the classical approach described above. Instead of projecting and lifting it first decomposes complex space and then refines this to a CAD [11], with the current implementation doing the complex decomposition incrementally by polynomial [9]. As reported in these papers, this approach can avoid some superfluous cell divisions. However, there is still the same choice of variable ordering to be made which can be crucial [12] with the Brown heuristic observed previously to work similarly well for the regular chains based algorithms [23].

Timings Timings are performed following the methodology of [19]. For each of the possible variable orderings, the polynomials defining the problems were given as input to the CAD in MAPLE with a time limit of 30 seconds. If none of the orderings finishes, all the orderings are attempted again with a time limit of 60 seconds.

Projection times are timed individually using our implementation in MAPLE of McCallum CAD projection (that returns the polynomials factorized) with a time limit of 10 seconds: these times are used to give a more meaningful comparison of heuristics that requires us to compute all the projections, with heuristics that do not need to do so.

Every CAD call was made in a separate MAPLE session launched from and timed in Python, to avoid MAPLE’s caching of intermediate results from one benchmark or ordering that may help another. From each timing, 0.075 seconds were removed: the average time that MAPLE takes to open on the computer when called from Python. It was removed as this is not a cost that would normally be paid but as a consequence of the benchmarking.

Uniqueness When studying the dataset it was observed that many examples were very similar to each other. Similar in the sense that they were described by very similar polynomials, resulting in CADs with equivalent tree structures for every variable ordering, making it likely that all aspects of the CAD generation were similar. It is well observed that there exist these families of very similar benchmarks in the SMT-LIB. Treating each of them as an independent benchmark could result in skewed experimental results. E.g. a heuristic that happens to perform well on a large family of almost identical benchmarks would receive a huge but unwarranted boost in the analysis if we do not take care.

To avoid this, the samples with the same number of cells in the CADs for all possible variable orderings are clustered and only one of them is included in the dataset. This ensures that there are no two problems with an equivalent CAD tree structure for each variable ordering.

4.2 Evaluation metrics

Existing evaluation metrics The most obvious metric to evaluate the choices of our heuristics is the total time taken to build CADs for all the problems with the orderings chosen by that heuristic: this metric will be referred to as total-time. Also, another metric that will be used to compare the different heuristics is the number of problems completed before timeout using the orderings chosen by the heuristic.

In previous studies such as [21] accuracy, i.e. the percentage of times that the fastest ordering is chosen, is used as one of the main metrics. However, as discussed in [22], for our context, accuracy is not the most meaningful metric. This is because it is well observed that the second-best ordering may only be very marginally worse than the best ordering and so picking that should also be

considered accurate. Further, the timings may include small amounts of computational noise which change the ranking of orderings in such subsets and thus the accuracy score.

In [22] the authors proposed to address this by considering a heuristic as successful if it identifies any ordering that takes no more than 20% additional time than the optimal. This fitted their work on a machine learning classification problem, but this definition is not suitable for regression, or use to evaluate a continuous range of possibilities. It considers equally inaccurate an ordering that is 30% slower and an ordering that is three or four times slower, and even an ordering that timed out. We thus propose a new metric for use in the evaluation in place of accuracy.

Markup We suggest measuring the amount of time that the chosen ordering takes above the time of the optimal ordering, as a percentage of the optimal ordering:

$$\frac{\text{heuristic_time} - \text{optimal_time}}{\text{optimal_time}}.$$

This allows for problems of different sizes to be evaluated relative to their possible solutions. For example, suppose Problem A’s optimal ordering took 10 seconds and Problem B’s took 20s. If the chosen ordering for Problem A took 2s longer than optimal then the score would be 0.2; while if that happened for Problem B the score would be 0.1, recognizing that the excess 2s is a less substantial markup for the larger problem.

However, this can lead to distortions for problems where the optimal ordering is really fast. For example, if the optimal ordering takes 0.02s and the chosen ordering takes 4s then the metric above would give that problem a very huge influence over the final score. To avoid that situation, and taking into consideration that anything below a second would likely be acceptable to use for constructing a CAD, we propose instead to add one to all the timings, i.e.

$$\text{Markup} = \frac{(\text{heuristic_time} + 1) - (\text{optimal_time} + 1)}{\text{optimal_time} + 1}.$$

This measure still allows the evaluation of relative potential but reduces distortions from fast examples and computational noise. In the example above, the metric would evaluate to 3.9 instead of 199. We refer to this as *Markup*, i.e. a measure of how far from the optimal this choice was.

Markup combines the benefits of both accuracy and total-time. Like total-time does it can measure not only if a choice was worse than the virtual-best but also how worse it was. But it adapts better to the different sizes of examples, unlike total-time where performing slightly worse in a difficult problem can have more impact on the metric than performing really bad in an easy example. Like accuracy it gives the same relevance to all the instances, but unlike accuracy it does not define a choice as simply either right or wrong.

Timeouts For computing markup and total-time we must decide how to deal with cases where the chosen variable ordering leads to a timeout in CAD computation. In this case, when an ordering does not finish within the time limit given it will be assumed that it would have taken twice the time limit given.

4.3 Metrics and expensive heuristics

Note that some of our heuristics are cheap, manipulating over data easily extracted from the polynomial, while others are expensive, requiring the use of CAD projection and thus algebraic computations. When analyzing an expensive heuristic we have the choice of ignoring the cost of the heuristic or taking it into account. It is clear that the latter is more realistic because without paying this cost it would not be possible to make the choice. However, the former way of analyzing the heuristic also brings some interesting insight. Therefore, when presenting the metrics for these heuristics (Table 1), the metric without including the cost of the heuristic will be shown between brackets.

For example, the number of examples marked as complete stands for the number of problems in which the CAD was constructed with the heuristic’s choice of variable ordering before the timeout. To adjust this in expensive heuristics, we count as timeouts the problems in which the time taken to choose the ordering plus the time taken to build the CAD did not exceed the time limit. As the more realistic value, the latter is outside brackets and the former within.

5 Results and analysis

The results given by the analysis of the different heuristics to choose the variable ordering for the 1019 benchmarks are summarized in Table 1, and a survival plot comparing the heuristics is presented in Figure 2. To produce the survival plot, for each heuristic the times taken to solve the problems with the variable ordering chosen by the heuristic are sorted into increasing order to form a sequence (t_i) , discarding the timed-out problems; and the points $(k, \sum_{i=1}^k t_i)$ are then plotted. This plot encapsulates visually a lot of information about the success of the heuristics on a given dataset (it does not say anything about heuristics relative performance on particular problem instances).

Name	Accuracy	Total time	Markup	# Completed
sotd	0.43	11007(9656)	1.56(1.16)	931(946)
mods	0.64	8137(6637)	0.57(0.13)	979(990)
logmods	0.49	9085(7535)	0.91(0.47)	968(983)
greedy-sotd	0.4	15669(15533)	2.55(2.51)	840(841)
brown	0.56	7590	0.25	974
gmods	0.58	6945	0.17	987

Table 1: Evaluation metrics for the different heuristics to choose the variable orderings for CAD. For the expensive heuristics, the metrics without taking into account the cost of the heuristic can be seen between brackets. In bold, the best measure of the metric out of all the heuristics.

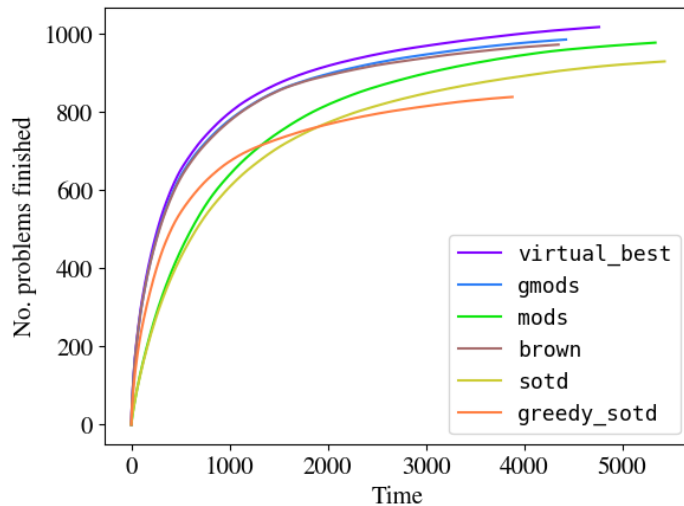


Fig. 2: Survival plot comparing heuristics on the benchmarks they can tackle before timeout.

The first thing to note is that *any* heuristic is significantly better than making a random choice, giving further evidence on the critical need for attention to this decision. We give further analysis on the heuristics grouped by their relative costs.

5.1 Expensive heuristics: `sotd` vs `mods`

As discussed earlier, one of the strategies that can be taken to choose an ordering is to compute all the projection phases for each variable ordering beforehand and base the decision on this information. Our new heuristic `mods` and the existing heuristic `sotd` follow this strategy. This approach requires a huge cost: it is possible to observe in Table 1 that one-fifth and one-seventh of the time taken to do a CAD by the ordering suggested by `mods` and `sotd` respectively is invested on deciding the ordering.

They both use the same information to take the decision, however, as can be seen in Table 1, `mods` outperforms `sotd` in all the presented metrics. The proposed heuristic picks the best ordering for almost two-thirds of the problems, while the existing one does so in less than half of them. The choices of `mods` reduce the total time by almost fifty minutes and solve almost 50 problems more with respect to the choices done by `sotd`.

Moreover, the ordering chosen by `mods` took on average 58% more time than the best ordering, while the choice of `sotd` took on average 160% more than it. Meaning that for a problem where the virtual-best ordering takes 10 seconds it is expected that the ordering proposed by `mods` takes 15.8 seconds while 26 seconds are expected from the ordering proposed by `sotd`.

When we compare `mods` to `logmods` we see that `logmods` is outperformed in all measured metrics. We thus conclude that the expected number of real roots for the polynomials in our benchmark set does not match well that of the random polynomials studied in [20].

5.2 Cheaper heuristics: `gmods` vs `brown`

The heuristics presented in Section 5.1 exerted a large amount of effort to solve the problem of choosing the ordering, at odds with the behaviour of most algorithm optimization heuristics.

We now look at the cheaper heuristics: `greedy-sotd` which greatly reduces the amount of projection information used to make a decision (compared to `sotd` and `mods`) and `brown` and `gmods` which do not use any such information beyond that required to build the single CAD.

We first note that even without taking into account the higher cost of `greedy-sotd`, it is greatly outperformed by the two heuristics that do not make use of projection information at all. When comparing `brown` and `gmods`: both heuristics have similar accuracies, however, the choices of `gmods` reduce the total time by ten minutes, and solves 13 problems more with respect to the choices of `brown`. Moreover, the ordering chosen by `gmods` took on average 17% more time than the best ordering, while the choice of `brown` took on average 25% more than it. Meaning that for a problem where the optimal ordering takes 10 seconds it is expected that the ordering proposed by `gmods` takes 11.7 seconds while 12.5 seconds are expected from the ordering proposed by `brown`.

To further understand how these two heuristics compare, and if there are subsets of problems in which one of them performs better than the other, an

adversarial plot is presented in Figure 3. This plots for each benchmark the time taken by the two heuristics against each other. In that figure, it can be observed that most of the points are close to the diagonal line, implying that both heuristics perform similarly for most of the instances.

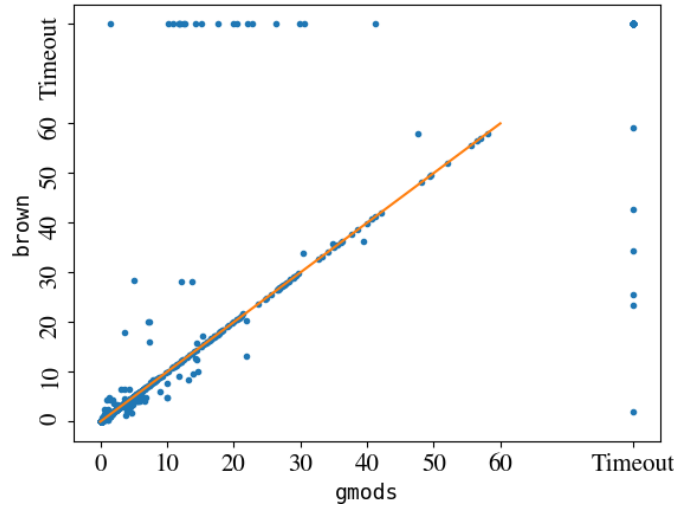


Fig. 3: Adversarial plot comparing `gmods` and `brown`.

However, it can be also observed that for some problems the ordering suggested by `gmods` timed out while `brown` proposed an ordering that completes and vice versa. This phenomenon is more common in favour of `gmods` but leaves open the possibility of a combination or meta-heuristic outperforming either.

5.3 Expensive vs cheap approach: `mods` vs `gmods`

It can be observed looking at the accuracies in Table 1 that `mods` picks better orderings than `gmods` and is superior in all other metrics if the cost of the heuristic is not taken into account (looking at the values between brackets in the same table). I.e. it has the strongest predictive power. However, as discussed in Section 4.2, accuracy is not the most interesting metric and `gmods` outperforms `mods` in all fair comparisons that take the cost of the heuristic into account.

In Figure 2 we see that `greedy-sotd` outperforms `sotd` at first, solving many problems in a shorter time, but in the long run `sotd` ends up solving more in total. In fact, it is possible to observe that the greedy heuristics start ahead of the expensive heuristics. This implies that it is especially disadvantageous

to compute all projections when working on easy problems, and motivates a separate analysis excluding the easiest problems.

The results when we restrict to only the hardest 134 problems (those whose optimal time need more than 10 seconds) are plotted in Figure 4. Now `mods` performs almost as well as `brown`. This further highlights the superiority of `gmods` over the rest in this particularly relevant slice of the problems where easy problems are excluded. Thus these expensive heuristics may still have a role as we expand our analysis to still harder problems.

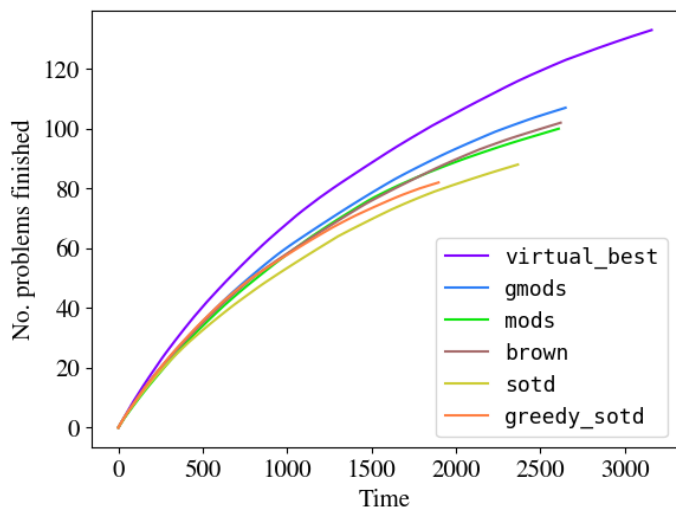


Fig. 4: Survival plot comparing heuristics on harder benchmarks only.

6 Final thoughts

6.1 Conclusions

The new heuristics motivated in this paper by the complexity analysis of CAD have clearly become the new state-of-the-art to choose the variable ordering for CAD. This leads to the most important conclusion of the paper: theoretical complexity analyses of an algorithm are a very powerful tool not just to compare algorithms but also to optimize them. These results also show that the benefits of a greedy or lazy approach in algebraic computation.

We would also highlight how out of the 5599 problems of three variables in the `QF_NRA` category of the `SMTLIB` library, only 1019 were found to be unique (in the sense of 4.1). I.e. three-quarters, at least of the three-variable problems in the `QF_NRA` category, are from the point of view of CAD copies of other

problems in that category. Their differences may be important when using SAT solvers to study the logic, or other incomplete methods, but for an analysis of a complete solver they should be merged as in our methodology.

Finally, we note that `logmods` failed in our experiments, implying that for the polynomials found in the `QF_NRA` category of the `SMTLIB` library the expected number of roots is not proportional to the logarithm of their degree, and therefore they do not follow the same distribution as studied in [20]. This is not that surprising given it is well documented that polynomials from applications are often different to those generated from a simple random generator.

6.2 Future work

An obvious future work is to experiment with the new heuristics on problems with more variables, higher degree, or data from other sources. We see a number of avenues beyond this for more research.

First we note that the results here should feed into work on machine learning methods to make such optimization decision. In fact, it would be interesting to study the extend to which the metrics presented here were used in the machine learning classifiers of [21] who created similar metrics among hundreds via an automated method. We are also interested in building simpler ML models using a restricted set of features. These could be more interpretable and thus offer further insights on how the features connect.

Next, we note that the actual CAD complexity analyses in e.g. [17], [3], [18] [25] were performed not on the projection polynomials as a single set but an optimal arrangement of them. This is known as the (m,d)-property and stems from the PhD thesis of McCallum. It would be interesting to see if there is any heuristic that can be deduced from the analysis involving this property.

We are also interested to look if the additional information encoded in `mods` could be obtained more cheaply than using CAD projections. Especially since the heuristic does not use all the information in these projections, only degree information. This would allow us to make choices only 13% slower on average than the virtual best (based on the results between brackets – without heuristic cost – in Table 1). Alternatively, another greedy version of `mods` could be developed, in which, similarly to `greedy-sotd`, instead of computing the whole projection phase for each possible variable ordering, a projection step is done for each available variable.

Acknowledgements

The authors would like to thank AmirHosein Sadeghimanesh for his interesting conversations and his constructive criticism, and for sharing his Maple code to perform CAD projections. We also thank the anonymous reviewers whose comments helped us improve the paper.

The research of the first author is supported financially from a scholarship of Coventry University. The research of the second author is supported by EPSRC

Grant EP/T015748/1, *Pushing Back the Doubly-Exponential Wall of Cylindrical Algebraic Decomposition* (the DEWCAD Project).

Research Data and Code Statement

Data and code necessary to generate the figures and results presented in this paper are available at: <https://doi.org/10.5281/zenodo.6750528>

References

1. Abraham, E., Davenport, J.H., England, M., Kremer, G.: Deciding the consistency of non-linear real arithmetic constraints with a conflict driven search using cylindrical algebraic coverings. *Journal of Logical and Algebraic Methods in Programming* **119**, 100633 (2021). <https://doi.org/10.1016/j.jlamp.2020.100633>
2. Barrett, C., Fontaine, P., Tinelli, C.: *The Satisfiability Modulo Theories Library (SMT-LIB)* (2016), www.SMT-LIB.org
3. Bradford, R., Davenport, J.H., England, M., McCallum, S., Wilson, D.: Truth table invariant cylindrical algebraic decomposition. *Journal of Symbolic Computation* **76**, 1–35 (2016). <https://doi.org/10.1016/J.JSC.2015.11.002>
4. Bradford, R., Davenport, J.H., England, M., Wilson, D.: Optimising problem formulation for cylindrical algebraic decomposition. In: *Intelligent Computer Mathematics. CICM 2013. Lecture Notes in Computer Science*, vol. 7961 LNAI, pp. 19–34. Springer, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39320-4_2
5. Brown, C.W.: Companion to the Tutorial Cylindrical Algebraic Decomposition. In: *International Symposium on Symbolic and Algebraic Computation - ISSAC (2004)*, <https://www.usna.edu/Users/cs/wcbrown/research/ISSAC04/handout.pdf>
6. Brown, C.W.: Open Non-uniform Cylindrical Algebraic Decompositions. In: *Proceedings of the International Symposium on Symbolic and Algebraic Computation, ISSAC*. pp. 85–92. Association for Computing Machinery (2015). <https://doi.org/10.1145/2755996.2756654>
7. Brown, C.W., Davenport, J.H.: The complexity of quantifier elimination and cylindrical algebraic decomposition. *Proceedings of the International Symposium on Symbolic and Algebraic Computation, ISSAC* pp. 54–60 (2007). <https://doi.org/10.1145/1277548.1277557>
8. Brown, C.W., Daves, G.C.: Applying Machine Learning to Heuristics for Real Polynomial Constraint Solving. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. vol. 12097 LNCS, pp. 292–301. Springer (2020). https://doi.org/10.1007/978-3-030-52200-1_29
9. Chen, C., Moreno Maza, M.: An Incremental Algorithm for Computing Cylindrical Algebraic Decompositions. In: *Computer Mathematics*, pp. 199–221. Springer Berlin Heidelberg, Berlin, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43799-5_17
10. Chen, C., Moreno Maza, M.: Cylindrical algebraic decomposition in the RegularChains library. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* **8592**, 425–433 (2014). https://doi.org/10.1007/978-3-662-44199-2_65

11. Chen, C., Moreno Maza, M., Xia, B., Yang, L.: Computing cylindrical algebraic decomposition via triangular decomposition. *Proceedings of the International Symposium on Symbolic and Algebraic Computation, ISSAC* pp. 95–102 (2009). <https://doi.org/10.1145/1576702.1576718>
12. Chen, C., Zhu, Z., Chi, H.: Variable Ordering Selection for Cylindrical Algebraic Decomposition with Artificial Neural Networks. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 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**(Automata Theory and Formal Languages), 134–183 (1975). https://doi.org/10.1007/3-540-07407-4_17
14. Collins, G.E.: Quantifier Elimination by Cylindrical Algebraic Decomposition — Twenty Years of Progress. In: *Texts and Monographs in Symbolic Computation*, pp. 8–23. Springer, Vienna (1998). https://doi.org/10.1007/978-3-7091-9459-1_2
15. 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. ACM Press, New York, New York, USA (2004). <https://doi.org/10.1145/1005285.1005303>
16. England, M., Bradford, R., Chen, C., Davenport, J.H., Maza, M.M., Wilson, D.: Problem Formulation for Truth-Table Invariant Cylindrical Algebraic Decomposition by Incremental Triangular Decomposition. In: *Lecture Notes in Computer Science (Intelligent Computer Mathematics)*, pp. 45–60. Springer Verlag (2014). https://doi.org/10.1007/978-3-319-08434-3_5
17. England, M., Bradford, R., Davenport, J.H.: Improving the use of equational constraints in cylindrical algebraic decomposition. *Proceedings of the International Symposium on Symbolic and Algebraic Computation, ISSAC* pp. 165–172 (2015). <https://doi.org/10.1145/2755996.2756678>
18. England, M., Bradford, R., Davenport, J.H.: Cylindrical algebraic decomposition with equational constraints. *Journal of Symbolic Computation* **100**, 38–71 (2020). <https://doi.org/10.1016/j.jsc.2019.07.019>
19. England, M., Florescu, D.: Comparing Machine Learning Models to Choose the Variable Ordering for Cylindrical Algebraic Decomposition. In: *Lecture Notes in Computer Science*. vol. *Intelligen*, pp. 93–108. Springer Verlag (2019). https://doi.org/10.1007/978-3-030-23250-4_7
20. Fairley, W.B.: The Number of Real Roots of Random Polynomials of Small Degree. *The Indian Journal of Statistics, Series B* **38**(2), 144–152 (1976), <http://www.jstor.org/stable/25052004>
21. Florescu, D., England, M.: Algorithmically generating new algebraic features of polynomial systems for machine learning. *CEUR Workshop Proceedings* **2460** (2019). <https://doi.org/10.48550/1906.01455>
22. Florescu, D., England, M.: Improved Cross-Validation for Classifiers that Make Algorithmic Choices to Minimise Runtime Without Compromising Output Correctness. In: *Lecture Notes in Computer Science*, vol. *Mathematic*, pp. 341–356. Springer (2020). https://doi.org/10.1007/978-3-030-43120-4_27
23. Huang, Z., England, M., Wilson, D.J., Bridge, J., Davenport, J.H., Paulson, L.C.: Using Machine Learning to Improve Cylindrical Algebraic Decomposition. *Mathematics in Computer Science* **13**(4), 461–488 (dec 2019). <https://doi.org/10.1007/s11786-019-00394-8>

24. Kremer, G., Abraham, E.: Fully incremental cylindrical algebraic decomposition. *Journal of Symbolic Computation* **100**, 11–37 (2020). <https://doi.org/10.1016/j.jsc.2019.07.018>
25. Li, H., Xia, B., Zhang, H., Zheng, T.: Choosing the Variable Ordering for Cylindrical Algebraic Decomposition via Exploiting Chordal Structure. *Proceedings of the International Symposium on Symbolic and Algebraic Computation, ISSAC* pp. 281–288 (2021). <https://doi.org/10.1145/3452143.3465520>
26. McCallum, S., Parusiński, A., Paunescu, L.: Validity proof of Lazard’s method for CAD construction. *Journal of Symbolic Computation* **92**, 52–69 (2019). <https://doi.org/10.1016/j.jsc.2017.12.002>
27. Wilson, D., England, M., Bradford, R., Davenport, J.H.: Using the distribution of cells by dimension in a cylindrical algebraic decomposition. *Proceedings - 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>