Choosing a variable ordering for truth-table invariant cylindrical algebraic decomposition by incremental triangular decomposition

Matthew England, Russell Bradford, James H. Davenport, and David Wilson

University of Bath, UK
{M.England, R.J.Bradford, J.H.Davenport, D.J.Wilson}@bath.ac.uk

Abstract. Cylindrical algebraic decomposition (CAD) is a key tool for solving problems in real algebraic geometry and beyond. In recent years a new approach has been developed, where regular chains technology is used to first build a decomposition in complex space. We consider the latest variant of this which builds the complex decomposition incrementally by polynomial and produces CADs on whose cells a sequence of formulae are truth-invariant. Like all CAD algorithms the user must provide a variable ordering which can have a profound impact on the tractability of a problem. We evaluate existing heuristics to help with the choice for this algorithm, suggest improvements and then derive a new heuristic more closely aligned with the mechanics of the new algorithm.

1 Introduction

A *cylindrical algebraic decomposition* (CAD) is: a *decomposition* of $\mathbb{R}^n$, meaning a collection of cells which do not intersect and whose union is $\mathbb{R}^n$; *cylindrical*, meaning the projections of any pair of cells with respect to a given variable ordering are either equal or disjoint; and, *(semi)-algebraic*, meaning each cell can be described using a finite sequence of polynomial relations. The original CAD by Collins [1] was introduced as a tool for quantifier elimination over the reals. Since then CAD has also been applied to problems including epidemic modelling [9], parametric optimisation [18], theorem proving [22], motion planning [23] and reasoning with multi-valued functions and their branch cuts [14].

Traditionally, a CAD is built *sign-invariant* with respect to a set of polynomials such that each one has constant sign in each cell, meaning only one sample point per cell need be tested to determine behaviour. Collins' algorithm works in two phases. In the *projection* phase an operator is repeatedly applied to polynomials each time producing a set in one fewer variables. Then in the *lifting* phase CADs of real space are built incrementally by dimension according to the real roots of these polynomials. A full description is in [1] and [13] summarises improvements from the first 20 years ([4] references more recent developments).

In 2009 an approach to CAD was introduced which broke with the projection and lifting framework [12]. Instead, a *complex cylindrical decomposition* (CCD) of $\mathbb{C}^n$ is built using triangular decomposition by regular chains, and then real root isolation is applied to move to a CAD of $\mathbb{R}^n$. We can view the CCD as an enhanced projection since gcds are calculated as well as resultants. It means the second phase is less expensive than lifting since case distinction can avoid identifying unnecessary roots. We use `PL-CAD` for CADs built by projection and lifting and `RC-CAD` for CADs built with the new approach. The initial work was improved in [11] by introducing purpose-built algorithms to refine a CCD incrementally by constraint whilst maintaining cylindricity and recycling subresultant calculations. A modification of the incremental algorithm to work with relations instead of polynomials then allowed for simplification in the presence of *equational constraints* (ECs): equations whose satisfaction is logically implied by the input. The output was no longer sign-invariant for polynomials but *truth-invariant* for a formula (the conjunction of relations). Similar ideas had been developed for `PL-CAD` [21] but were difficult to generalise to multiple ECs.

In [2], a new variant of `RC-CAD` was presented. Here, instead of building a CAD for a set of polynomials or relations we build one for a sequence of quantifier free formulae (QFFs) such that each formula has constant truth value on each cell: a *truth-table invariant* CAD or TTICAD. It followed the development of TTICAD theory for `PL-CAD` (see [4], [5]) and combined it with the benefits of `RC-CAD`. The CCD is built using a tree structure incrementally refined by constraint. ECs are dealt with first, with branches refined for other constraints in a formula only if the EC is satisfied. Further, when there are multiple ECs in a formula branches can be removed when the constraints are not all satisfied. See [2] and [11] for full details. Building a TTICAD is often the best way to obtain a truth-invariant CAD for a single formula (if the formula has disjunctions then treating each conjunctive clause as a subformula allows simplification in the presence of any ECs) but is also the object required for applications like simplification of complex functions via branch cut analysis (see [3] [17]). The implementation of [2] in the `RegularChains` Library [24] (denoted `RC-TTICAD`) is our topic here.

All CAD algorithms require the user to specify an ordering on the variables. For `PL-CAD` this determines the order of projection and thus the sequence of Euclidean spaces considered en-route to $\mathbb{R}^n$. For `RC-CAD` if determines both the triangular decompositions performed and the refinement to $\mathbb{R}^n$. Depending on the application there may be a free or constrained choice. For example, in quantifier elimination we must order the variables as they are quantified but may change the ordering within quantifier blocks. Problems easy in one variable ordering can be infeasible in another, with [8] giving problems where one ordering leads to a cell count constant in the number of variables and another to one doubly exponential (irrespective of the algorithm used). Hence any choice must be made intelligently. We write $y \succ x$ if y is greater than x in an ordering (noting that `PL-CAD` eliminates variables from greatest to lowest in the ordering).

We start in Section 2 by evaluating (with respect to `RC-TTICAD`) existing heuristics for choosing the variable ordering. Then in Section 3 we suggest some extensions to improve their use before developing our own heuristic more closely aligned to `RC-TTICAD`. We give our conclusions in Section 4.

2 Evaluating existing heuristics

In what follows we assume f is a polynomial, v a variable and P the set of polynomials defining the input to RC-TTICAD. Let $\mathtt{deg}(f, v)$ be the degree of f in v, $\mathtt{tdeg}(f)$ the total degree of f and $\mathtt{lcoeff}(f, v)$ the leading coefficient of f when considered as a univariate polynomial in v. For a set let `max` be the maximum value, `sum` the sum of values and `#` the number of values. We start by considering two heuristics already in use for choosing the variable ordering in algorithms from the `RegularChains` Library [24].

Triangular: Start with the first criteria, breaking ties with successive ones.
1. Let $v^{[1]} = \mathtt{max}(\{\mathtt{deg}(f, v), \mid f \in P\})$. Then set $y \succ x$ if $y^{[1]} < x^{[1]}$.
2. Let $v^{[2]} = \mathtt{max}(\{\mathtt{tdeg}(\mathtt{lcoeff}(f, v)), \mid f \in P \text{ (containing } v)\})$. Then set $y \succ x$ if $y^{[2]} < x^{[2]}$.
3. Let $v^{[3]} = \mathtt{sum}(\{\mathtt{deg}(f, v), \mid f \in P\})$. Then set $y \succ x$ if $y^{[3]} < x^{[3]}$.

Brown: Start with the first criteria, breaking ties with successive ones.
1. Set $y \succ x$ if $y^{[1]} < x^{[1]}$ (as defined in the heuristic above).
2. Let $v^{[4]} = \mathtt{max}(\{\mathtt{tdeg}(t), \mid t \text{ is a monomial (containing } v) \text{ from a polynomial in } P\})$. Then set $y \succ x$ if $y^{[4]} < x^{[4]}$.
3. Let $v^{[5]} = \mathtt{\#}(\{t, \mid t \text{ is a monomial (containing } v) \text{ from a polynomial in } P\})$. Then set $y \succ x$ if $y^{[5]} < x^{[5]}$.

These use only simple measures on the input. The first was implemented for [10] (although not detailed there) and is used for various algorithms in the RegularChains Library (being the default for `SuggestVariableOrder`). The second was first described in the CAD tutorial notes [7] and in [19] was shown to do well in choosing a variable ordering for Qepcad (an implementation of PL-CAD).

The next two heuristics were developed for PL-CAD and work by running the projection phase for each possible variable ordering and picking an optimal ordering using a measure of the projection set. Our implementations use the projection polynomials generated by McCallum's operator [20] on P.

Sotd: Select the variable ordering with the lowest *sum of total degrees* for each of the monomials in each of the polynomials in the projection set.

Ndrr: Select the variable ordering with the lowest *number of distinct real roots* of the univariate projection polynomials

Sotd was suggested in [15] where it was found to be a good heuristic for CAD in Redlog (another implementation of PL-CAD). Ndrr was suggested in [6] as a means to identify differences occurring only in real space and thus missed by measures on degree. These heuristics are clearly more expensive but note that the lifting phase does the bulk of the work for PL-CAD, with the projection phase often trivial (and if not then the lifting phase is likely infeasible).

To evaluate the heuristics we generated 600 random examples, each with two QFFs themselves a conjunction of two constraints. There were 100 for each of six system types: **00**, **10**, **20**, **11**, **12**, **22**. Each digit in these labels refers to the number of those constraints which are equalities (with the others strict inequalities). The polynomials defining the constraints were sparse and in three variables, generated using MAPLE's `randpoly` function. `RC-TTICAD` was applied to build CADs for the problems using each of the six possible variable orderings. A time out of 12 minutes a problem was used affecting only six examples (one with system type **20**, two with **10** and three with **00**). For the others, the cell count and computation time (in seconds) for each CAD was recorded.

Table 1 summarises this data, showing the average and median values for each system. As expected `RC-TTICAD` does better in the presence of ECs. We note the anomaly between system types **10** and **20**: it seems the savings from truncating branches where ECs are not simultaneously satisfied are wiped out by the costs of doing so. The savings would probably be restored in the QFFs contained further non-ECs requiring more processing per branch.

Next we note that the median cell counts and timings are considerably less than the mean average for every system type, indicating the presence of outliers. We provided a third piece of data: the median of the values for each problem when averaged over the six possible orderings. This will still avoid outlier problems but not outlier orderings. In every case this value is much closer to the mean average, indicating that most outlying data comes from bad orderings rather than bad problems, and thus highlighting the practical importance of the ordering.

We performed the following calculations for each problem and each heuristic:

1. Calculate the average cell count and timing for the problem from the six possible variable orderings.
2. Run and time each heuristic for choosing a variable ordering for the problem.
3. Record the cell count and timing of the heuristic's choice. If a heuristic chooses multiple orderings we take the first lexicographically.
4. Calculate the saving from using the heuristic's choice compared to the problem average, i.e. $(1) - (3)$ for cell counts and $(1) - (2) - (3)$ for timings.
5. Evaluate the savings as percentages of the problem average, i.e. $100(4)/(1)$.

Table 2 (the first four rows) shows averages of the values in (5) over problems of the same system type and the whole problem set. All four existing heuristics offer significant cell savings and so are making good selections of variable ordering. Although Sotd offers the highest cell savings overall, its higher costs means the

Table 1. The performance on `RC-TTICAD` over all variable orderings. Displayed are the mean and median values and the median of the values after averaging over orderings.

System	**Cell Count**			**Computation Time**		
	Mean	Median	Median of av.	Mean	Median	Median of av.
22	750.13	478	612.67	1.84	1.37	1.58
12	934.42	682	861.50	2.73	2.12	2.47
11	1355.45	839	1212.33	3.41	2.10	2.99
20	3271.51	2193	2918	8.90	6.02	7.92
10	2949.02	1528	2275	8.44	4.71	6.62
00	9838.76	4874	8566.67	34.46	17.05	29.88

Triangular heuristic is the most time efficient. The heuristics' costs decrease as a percentage of the CAD computation time for systems with fewer ECs and so Sotd can achieve a much higher saving for problems of type **00** than **22**. But there are other differences between systems not explained by running times, such as Brown generally saving more cells than Triangular but not for systems **20**.

3 Extensions, improvements and a new heuristic

Combining measures In [6] Ndrr was developed to help with problems where Sotd could not due to differences occurring in real space only. Hence a logical extension is to use their measures in tandem. We have used the same evaluation for heuristics SN (where Ndrr is used as a tie-break for Sotd) and NS (where Sotd is the tie-break) with results given in Table 2. In both cases the tie-breaker gives marginally higher cell savings than using the single heuristic, with NS giving the highest cell saving so far, but Brown remaining the most efficient for computation time. The costs of running these heuristics will be higher than using the single measure (at least for problems where the first measure tied) but these extra costs are usually less than the extra time savings obtained.

Greedy heuristics A *greedy* variant of Sotd was also suggested in [15] with the variable ordering decided alongside the projection phase. At each step the projection operator is evaluated with respect to all unallocated variables and the variable whose set has lowest sum of total degree of each monomial of each new polynomial is fixed in the ordering. We denote this GS in Table 2 where we see it offers less cell savings than full Sotd (though still competitive) but has lower costs and so gives more time savings. The cost of Sotd will increase alongside the number of admissible variable orderings and so for such problems the greedy variant may offer the only sensible approach. A greedy variant of Ndrr is not possible since that measure is on the univariate polynomials only.

Using information from `PL-TTICAD` The projection sets used so far are those for a sign-invariant `PL-CAD`, thus considering not the input constraints but the polynomials defining them. Since `RC-TTICAD` is building a TTICAD (smaller for all except systems **00**), a sensible extension is to use the projection phase from `PL-TTICAD` [5]. However, we cannot match the declared output structure exactly: `PL-TTICAD` uses (at most) one declared EC per QFF (with others treated the same as non-ECs). Hence, for QFFs with 2 ECs we will run the projection phase with the first of these declared (so for example, systems **20** are treated the same as **10**). We denote the heuristics applying Sotd and Ndrr with this projection set as S-TTI and N-TTI. From Table 2 we see they offer substantially more cell savings than their standard versions. They also achieve higher time savings: both due to the improved choices and lower running costs (since the TTICAD projection operator is a subset of the sign-invariant one). We can also run the greedy variant of Sotd with the `PL-TTICAD` projection phase (denoted GS-TTI in Table 2) which will lose some of the cell savings but increase the time savings.

Developing a new heuristic We now aim to develop a new heuristic, which considers more algebraic information than the input but is tailored to `RC-TTICAD` itself rather than a `PL-CAD` algorithm. The main saving offered by the regular chains approach is case distinction meaning that not all projection factors are considered universally. For example, the second coefficient in a polynomial is only considered when the first vanishes (and then only evaluated modulo that constraint). Consider a set of polynomials consisting of the following:

- the discriminants, leading coefficients and cross-resultants of the polynomials forming the first constraint in each QFF;
- if a QFF has no EC then also the (other) discriminants, leading coefficients and cross resultants of all polynomials defining constraints there;
- if a QFF has more than one EC then also the resultant of the polynomial defining the first with that of the second.

Here the resultants, discriminants and coefficients are taken with respect to the first variable in the ordering. These polynomials will all be sign-invariant in the output: see [2], [11] for the algorithm specifications and [16] for a fuller discussion and examples (from a study in the context of choosing the constraint ordering). This set does not contain all polynomials computed by `RC-TTICAD`, but those which are considered in their own right rather than modulo others.

We define a new heuristic to pick an orderings in two stages: First variables are ordered according to maximum degree of the polynomials forming the input (as with Triangular and Brown). Then ties are broke by calculating the set of polynomials above for each unallocated variable and ordering according to sum of degree (in that variable). This is denoted NewH in Table 2 and we see it achieves almost as many savings as S-TTI despite using fewer polynomials.

We could go further by including some more of the missing information. For example, we can use the degree of the omitted discriminants, resultants and leading coefficients as a third tie-break. This heuristic is denoted NewH-ext and the results of its evaluation are in the final row of Table 2. We see it achieves even higher cell savings (and the greatest time savings of any heuristic).

4 Conclusions

We have demonstrated that the variable ordering is important for `RC-TTICAD` and using any heuristic is advantageous. Simple measures on the input can be effective, but more cell savings can be obtained by using additional information. Existing heuristics obtained this from the projection phase of `PL-CAD` and we have suggested a new heuristic aligned to `RC-TTICAD` which identifies polynomials of most importance to the algorithm. It was sufficient for allocating two variables (and hence ordering three) as required by our problem set. Extending to problems with more variables is a topic of future work.

The heuristics performance varied with the system classes and so heuristics that changed along with this performed better. The precise relationships at work here are not always clear to see. Machine learning on the set of measures used

Table 2. Comparing the savings (as a percentage of the problem average) in cells (C) and net timings (NT) from various heuristics.

Heuristic	22		12		11		20		10		00		All	
	C	NT	C	NT	C	NT	C	NT	C	NT	C	NT	C	NT
Triangular	32.6	33.9	33.9	34.0	40.9	41.3	47.9	46.8	47.7	47.2	56.0	58.8	43.0	43.6
Brown	37.6	39.1	39.3	39.8	45.9	47.1	45.0	44.3	51.6	50.9	61.9	64.5	46.8	47.5
Sotd	36.7	23.9	37.9	27.7	49.4	40.4	42.8	39.5	56.3	53.9	59.9	61.8	47.1	41.0
Ndrr	40.1	21.2	44.1	33.0	40.2	30.7	35.7	34.4	54.8	51.3	54.0	54.3	44.9	37.4
SN	37.0	24.3	37.2	27.4	49.2	40.4	42.5	39.6	56.0	53.5	60.4	62.5	47.0	41.1
NS	41.3	22.6	41.2	30.7	47.8	37.1	38.7	36.0	57.1	51.7	58.4	60.2	47.3	39.6
GS	35.0	32.7	33.7	32.5	49.5	46.5	39.8	38.9	52.3	52.1	52.5	55.9	43.8	43.3
S-TTI	42.7	40.4	46.4	43.2	55.0	49.1	48.4	48.1	61.2	60.2	59.9	61.7	52.2	50.3
N-TTI	48.5	37.1	46.8	40.5	48.6	42.3	47.8	46.9	59.0	55.3	54.0	54.3	50.7	46.0
GS-TTI	46.4	47.2	44.9	44.5	56.7	54.7	49.3	50.2	56.7	57.5	52.8	55.9	51.1	51.6
NewH	45.9	45.5	41.8	43.5	51.4	50.8	48.2	47.6	56.4	52.4	67.0	68.5	51.7	51.3
NewH-ext	46.2	45.9	42.2	43.3	51.6	51.4	49.3	49.5	55.9	52.0	67.0	68.5	52.0	51.7

by the heuristics may offer a meta-heuristic greater than the sum of its parts (as was found to be the case recently when choosing a variable ordering for Qepcad [19]). Finally, we note that when using RC-TTICAD there are questions of problem formulation other that the variable ordering to use. As implied in Section 3, the order the constraints are presented affects the output. Advice on making this choice intelligently was recently derived in [16].

Acknowledgements

This work was supported by EPSRC grant EP/J003247/1. RC-TTICAD was developed by Changbo Chen, Marc Moreno Maza and the present authors.

References

1. D. Arnon, G.E. Collins, and S. McCallum. Cylindrical algebraic decomposition I: The basic algorithm. *SIAM J. Comput.*, 13:865–877, 1984.
2. R. Bradford, C. Chen, J.H. Davenport, M. England, M. Moreno Maza, and D. Wilson. Truth table invariant cylindrical algebraic decomposition by regular chains. *Submitted for publication.* Preprint: `http://opus.bath.ac.uk/38344/`, 2014.
3. R. Bradford and J.H. Davenport. Towards better simplification of elementary functions. *Proc. ISSAC '02*, pages 16–22. ACM, 2002.
4. R. Bradford, J.H. Davenport, M. England, S. McCallum, and D. Wilson. Cylindrical algebraic decompositions for boolean combinations. *Proc. ISSAC '13*, pages 125–132. ACM, 2013.
5. R. Bradford, J.H. Davenport, M. England, S. McCallum, and D. Wilson. Truth table invariant cylindrical algebraic decomposition. *Submitted for publication.* Preprint: `http://opus.bath.ac.uk/38146/`, 2014.

6. R. Bradford, J.H. Davenport, M. England, and D. Wilson. Optimising problem formulations for cylindrical algebraic decomposition. In: *Intelligent Computer Mathematics* (LNCS 7961), pages 19–34. Springer Berlin Heidelberg, 2013.
7. C.W. Brown. Companion to the tutorial: Cylindrical algebraic decomposition, presented at ISSAC 2004. Available from: `http://www.usna.edu/Users/cs/wcbrown/research/ISSAC04/handout.pdf`.
8. C.W. Brown and J.H. Davenport. The complexity of quantifier elimination and cylindrical algebraic decomposition. *Proc. ISSAC '07*, pages 54–60. ACM, 2007.
9. C.W. Brown, M. El Kahoui, D. Novotni, and A. Weber. Algorithmic methods for investigating equilibria in epidemic modelling. *J. Symbolic Computation*, 41:1157–1173, 2006.
10. C. Chen, J.H. Davenport, F. Lemaire, M. Moreno Maza, B. Xia, R. Xiao, and Y. Xie. Computing the real solutions of polynomial systems with the RegularChains library in Maple. *ACM C.C.A.*, 45(3/4):166–168, 2011.
11. C. Chen and M. Moreno Maza. An incremental algorithm for computing cylindrical algebraic decompositions. *Proc. ASCM '12*. To appear in LNAI, Springer. Preprint: `http://arxiv.org/abs/1210.5543`, 2012.
12. C. Chen, M. Moreno Maza, B. Xia, and L. Yang. Computing cylindrical algebraic decomposition via triangular decomposition. *Proc. ISSAC '09*, pages 95–102. ACM, 2009.
13. G.E. Collins. Quantifier elimination by cylindrical algebraic decomposition – 20 years of progress. In: *Quantifier Elimination and Cylindrical Algebraic Decomposition*, Texts & Monographs in Symb. Com., pages 8–23. Springer-Verlag, 1998.
14. J.H. Davenport, R. Bradford, M. England, and D. Wilson. Program verification in the presence of complex numbers, functions with branch cuts etc. *Proc. 14th SYNASC '12*, pages 83–88. IEEE, 2012.
15. A. Dolzmann, A. Seidl, and T. Sturm. Efficient projection orders for CAD. *Proc. ISSAC '04*, pages 111–118. ACM, 2004.
16. M. England, R. Bradford, C. Chen, J.H. Davenport, M. Moreno Maza, and D. Wilson. Problem formulation for truth-table invariant cylindrical algebraic decomposition by incremental triangular decomposition. *To Appear* Proc. CICM '14 (LNAI 8543), pages 46–60, 2014. Preprint: `http://opus.bath.ac.uk/39231/`
17. M. England, R. Bradford, J.H. Davenport, and D. Wilson. Understanding branch cuts of expressions. In: *Intelligent Computer Mathematics* (LNCS vol. 7961), pages 136–151. Springer Berlin Heidelberg, 2013.
18. I.A. Fotiou, P.A. Parrilo, and M. Morari. Nonlinear parametric optimization using cylindrical algebraic decomposition. *Proc. CDC-ECC '05*, pages 3735–3740, 2005.
19. Z. Huang, M. England, D. Wilson, J.H. Davenport, L. Paulson, and J. Bridge. Applying machine learning to the problem of choosing a heuristic to select the variable ordering for cylindrical algebraic decomposition. *To Appear:* Proc. CICM '14 (LNAI 8543), pages 92–107, 2014. Preprint: `http://opus.bath.ac.uk/39232/`
20. S. McCallum. An improved projection operation for cylindrical algebraic decomposition. In: *Quantifier Elimination and Cylindrical Algebraic Decomposition*, Texts & Monographs in Symb. Comp., pages 242–268. Springer-Verlag, 1998.
21. S. McCallum. On projection in CAD-based quantifier elimination with equational constraint. *Proc. ISSAC '99*, pages 145–149. ACM, 1999.
22. L.C. Paulson. Metitarski: Past and future. In: *Interactive Theorem Proving* (LNCS vol. 7406), pages 1–10. Springer, 2012.
23. D. Wilson, J.H. Davenport, M. England, and R. Bradford. A "piano movers" problem reformulated. *Proc. SYNASC '13*, pp. 53–60. IEEE, 2014.
24. The RegularChains Library: `http://www.regularchains.org`