FORMULATING PROBLEMS FOR REAL ALGEBRAIC GEOMETRY

MATTHEW ENGLAND

Abstract. We discuss issues of problem formulation for algorithms in real algebraic geometry, focussing on quantifier elimination by cylindrical algebraic decomposition. We recall how the variable ordering used can have a profound effect on both performance and output and summarise what may be done to assist with this choice. We then survey other questions of problem formulation and algorithm optimisation that have become pertinent following advances in CAD theory, including both work that is already published and work that is currently underway. With implementations now in reach of real world applications and new theory meaning algorithms are far more sensitive to the input, our thesis is that intelligently formulating problems for algorithms, and indeed choosing the correct algorithm variant for a problem, is key to improving the practical use of both quantifier elimination and symbolic real algebraic geometry in general.

1. Introduction

We will discuss the effect problem formulation can have on the use of symbolic algorithms for real algebraic geometry. This follows our recent work on cylindrical algebraic decomposition, one of the most important algorithms in this field. We discuss the issue of variable ordering, well known to play a key role, but also survey a number of other issues that are now pertinent.

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

$$\Phi := Q_{k+1}x_{k+1} \dots Q_n x_n \, \phi(x_1, \dots, x_n),$$

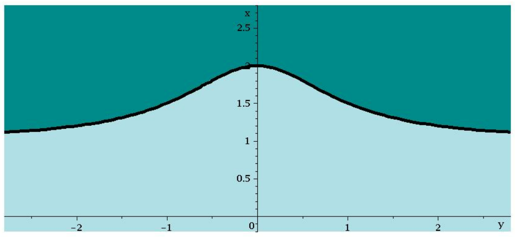
quantifier elimination (QE) is the problem of producing $\psi(x_1, \dots, x_k)$, a quantifier-free formulae equivalent to Φ. In the case $k = 0$ we have a *decision problem*: is Φ true? Tarski proved that QE is always possible for semi-algebraic formulae (polynomials and inequalities) over $\mathbb{R}$ [13]. The complexity of Tarski's method is indescribable as a finite tower of exponentials and so when Collins gave an alternative with cylindrical algebraic decomposition (CAD) [7] it was a major breakthrough despite complexity doubly exponential in the number of variables. CAD implementations remain the best option for many classes of problems.

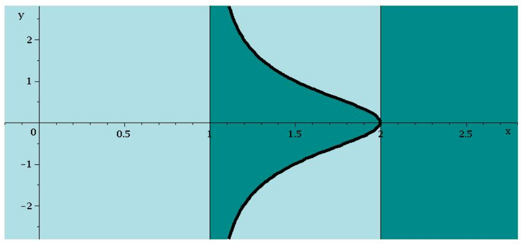
Collins' CAD algorithm works in two stages. First *projection* calculates sets of projection polynomials S_i in variables $(x_1, \dots, x_i)$ by applying an operator recursively starting with the polynomials from ϕ. Then in the *lifting* stage decompositions of real space in increasing dimensions are formed from the roots of those polynomials. First, the real line is decomposed according to the roots of the univariate polynomials. Then over each cell c in that decomposition the bivariate polynomials are taken at a sample point and a decomposition of $c \times \mathbb{R}$ is produced according to their roots. Taking the union gives the decomposition of $\mathbb{R}^2$ and we proceed this way to a decomposition of $\mathbb{R}^n$. The decompositions are cylindrical (projections of any two cells onto the first k coordinates are either identical or disjoint) and each cell is a semi-algebraic set (described by polynomial relations).

Collins' original algorithm uses a projection operator which guarantees CADs of $\mathbb{R}^n$ on which the polynomials in ϕ have constant sign, and thus Φ constant truth value, on each cell. Hence only a sample point from each cell needs to be tested and the equivalent quantifier free formula ψ can be generated from the semi-algebraic sets defining the cells in the CAD of $\mathbb{R}^k$ for which Φ is true. There have been numerous improvements, optimisations and extensions of CAD since Collins' work (with a summary of the first 20 years given in [8]).

2. Variable ordering

When using CAD for QE we must project quantified variables first, but we are free to project the other variables in any order (and to change the order within quantifier blocks). The variable ordering used can make a big difference. For example, let $f := (x-1)(y^2+1)-1$ and consider the two minimal CADs visualised below. In each case we project down with the left figure projecting x first and the right y. In this case we see that wrong choice more than doubles the number of cells. Of course, this is just a toy example, but [5] defined a class of examples where changing variable ordering would change the number of cells required from constant to doubly exponential in the number of variables.





Various heuristics exist to help choose a good variable ordering:

Brown: Eliminate lowest degree variable first (with tie-breaking rules) [4, Section 5.2]. Quite effective but considers only the initial input rather than the full projection set.

sotd: For all admissible orderings, calculate the projection set and choose the one with smallest *sum of total degree* [9]. Performs well but costly with many orderings.

Greedy sotd: Allocate one variable of the ordering at a time by projecting each unallocated variable and choosing the one which increases the sotd least [9].

ndrr: The sotd based heuristics can be misled, or give ties, especially when the differences lie in the real geometry. In this case we can compare the *number of distinct real roots* of the univariate projection polynomials [3] (the first step of lifting).

Machine Learning: Essentially a meta-heuristic on the above [11].

3. Other questions of input formulation

A key improvement to CAD is the development of projection operators that guarantee only truth invariance of ϕ, rather than sign-invariance of the polynomials within. This is achieved by considering the logical structure of ϕ, but brings in sensitivity to such structure.

Designating ECs: An *equational constraint* (EC) is an equation logically implied by a formula. The algorithm in [12] builds a CAD relative to a designated EC which is sign-invariant for the polynomial defining the EC, and for the other polynomials only when the EC is satisfied. If a formula has more than one EC, which to designate?

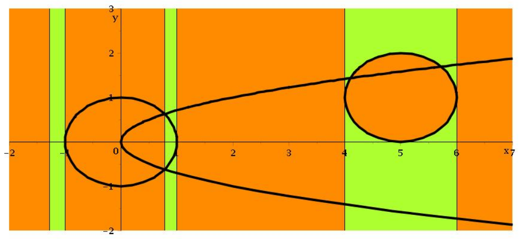
Sub-formulae for TTICAD: In [2] a truth-table invariant CAD (TTICAD) was defined as a CAD on whose cells the truth-table for a set of formulae is invariant. A new operator was presented which takes advantage of ECs in the separate formulae. If any formula has more than one EC then we have the issue above again. Further, TTICAD can be used to find a truth-invariant CAD for a single formula by breaking it up into sub-formulae, but how best to do this?

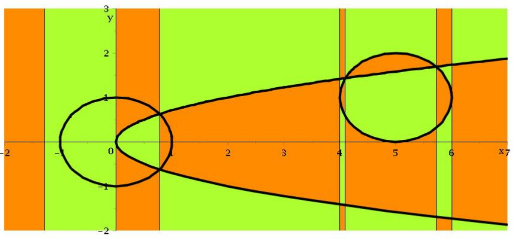
Experimental results in [2] suggested the heuristics above can also help with these questions, but when the issues are combined the number of possibilities can become overwhelming. Further, there are additional issues where the existing heuristics are of no help.

Order to process constraints: In [1] a new TTICAD algorithm is presented which is sensitive to the order in which constraints are considered. The images below represent two TTICADs relative to a formula defined by the polynomials graphed. The difference is caused solely by this ordering with the one on the right having three times more cells. In [10] new heuristics are developed to help with this choice.

Implicit ECs: Consider $\phi := (f_1 = 0 \wedge \phi_1) \vee (f_2 = 0 \wedge \phi_2)$. There is no explicit EC but the formula is logically equal to $(f_1 f_2 = 0) \wedge \phi$. Using this gives the benefit of the reduced projection set and thus less cells, but the increase in polynomial degrees may have an impact on timings.

Well-orientedness: Some CAD algorithms only work on input that is *well-oriented*. The precise details of this condition varies between algorithms and it is possible for input to be well-oriented for one but not another. This raises the question of whether a good choice can be made at the start, or if partial calculations can be reused?





4. Formulating problems, preprocessing and algorithm choice

Finally, we remark on some related issues which have come to light recently.

Precondition input: In [15] the idea of preconditioning the input to CAD using Groebner bases was investigated, with [2] extending this to TTICAD. The former found that this could be extremely beneficial, but not universally so. A heuristic was developed to identify when, but [2] found this was not suitable for TTICAD.

Deriving the mathematics: In [14] a long standing motion planning problem was solved using CAD by changing the analysis used to formulate the input formula. Instead of a description of the feasible region a negation of one for the infeasible region was used. Such a reformulation was easy to do but made a great difference to the feasibility of CAD. How can we identify such benefits in general?

Algorithm choice: Recently an alternative to the projection and lifting approach to CAD has been investigated, in which a decomposition of complex space is first built using triangular decompositions and regular chains theory [6] (this was how the

TTICAD algorithm in [1] differed from [2]). Experiments in [6] and [1] show that the different approaches outperform each other for different examples. How can we classify examples for use with one approach or the other?

5. Conclusions

We have summarised issues of problem formulation which can dramatically affect the performance of CAD. In some cases heuristics have been developed to help, but there is still much work to be done in making these practical and in extending them to the currently unanswered questions. It is likely that much of what is learned here could be used throughout QE, or more generally for symbolic algebraic geometry.

References

[1] R. Bradford, C. Chen, J.H. Davenport, M. England, M. Moreno Maza, and D. Wilson. Truth table invariant cylindrical algebraic decomposition by regular chains. Preprint: `arxiv:1401.6310`

[2] R. Bradford, J.H. Davenport, M. England, S. McCallum, and D. Wilson. Cylindrical algebraic decompositions for boolean combinations. In *Proc. ISSAC '13*, pages 125–132. ACM, 2013.

[3] 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.

[4] C.W. Brown. Companion to the tutorial: Cylindrical algebraic decomposition, presented at ISSAC '04. `http://www.usna.edu/Users/cs/wcbrown/research/ISSAC04/handout.pdf`, 2004.

[5] C.W. Brown and J.H. Davenport. The complexity of quantifier elimination and cylindrical algebraic decomposition. In *Proc. ISSAC '07*, pages 54–60. ACM, 2007.

[6] C. Chen and M. Moreno Maza. An incremental algorithm for computing cylindrical algebraic decompositions. *Proc. ASCM '12*. To appear in LNAI, Springer. Preprint: `arxiv:1210.5543`, 2012.

[7] G.E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Proc. 2nd GI Conference on Automata Theory and Formal Languages*, pages 134–183. Springer-Verlag, 1975.

[8] G.E. Collins. Quantifier elimination by cylindrical algebraic decomposition – 20 years of progress. In *Quantifier Elimination and Cylindrical Algebraic Decomposition*, pages 8–23. Springer-Verlag, 1998.

[9] A. Dolzmann, A. Seidl, and T. Sturm. Efficient projection orders for CAD. In *Proc. ISSAC '04*, pages 111–118. ACM, 2004.

[10] 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*, 2014. Preprint: `arxiv:1404.6371`

[11] 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*, 2014. Preprint: `arxiv:1404.6369`

[12] S. McCallum. On projection in CAD-based quantifier elimination with equational constraint. In *Proc. ISSAC '99*, pages 145–149. ACM, 1999.

[13] A. Tarski. A decision method for elementary algebra and geometry. In *Quantifier Elimination and Cylindrical Algebraic Decomposition*, pages 24–84. Springer-Verlag, 1998.

[14] D. Wilson, J.H. Davenport, M. England, and R. Bradford. A "piano movers" problem reformulated. In *Proc. SYNASC '13*. IEEE, 2013.

[15] D.J. Wilson, R.J. Bradford, and J.H. Davenport. Speeding up cylindrical algebraic decomposition by Gröbner bases. In *Intelligent Computer Mathematics* (LNCS 7362), pages 280–294. Springer, 2012.

University of Bath, UK.
E-mail address: `M.England@bath.ac.uk`