Experience with Heuristics, Benchmarks & Standards for Cylindrical Algebraic Decomposition

Matthew England*, and James H. Davenport†,
*Coventry University Faculty of Engineering, Environment and Computing, Coventry, CV1 2JH, U.K.
Email: Matthew.England@coventry.ac.uk
†University of Bath Department of Computer Science, Bath, BA2 7AY, U.K.
Email: J.H.Davenport@bath.ac.uk

***Abstract*—In the paper which inspired the SC^2 project, [E. Ábráham, *Building Bridges between Symbolic Computation and Satisfiability Checking*, Proc. ISSAC '15, pp. 1–6, ACM, 2015] the author identified the use of sophisticated heuristics as a technique that the Satisfiability Checking community excels in and from which it is likely the Symbolic Computation community could learn and prosper. To start this learning process we summarise our experience with heuristic development for the computer algebra algorithm Cylindrical Algebraic Decomposition. We also propose and discuss standards and benchmarks as another area where Symbolic Computation could prosper from Satisfiability Checking expertise, noting that these have been identified as initial actions for the new SC^2 community in the CSA project, as described in [E. Ábráham et al., *SC^2: Satisfiability Checking meets Symbolic Computation (Project Paper)*, Intelligent Computer Mathematics (LNCS 9761), pp. 28–43, Springer, 2015].**

I. Introduction

This article is inspired by the SC^2 project[1], a new initiative to forge a joint community from the existing fields of Symbolic Computation and Satisfiability Checking. For further details on the project we refer the reader to:

- [2] which introduced the two fields, describes some of the challenges and opportunities from working together, and outlines planned project actions; and
- [1] the accompanying paper to an invited talk at ISSAC 2015 which inspired the creation of the new project and community.

Within [1] the author outlines the strengths and weaknesses of the two communities, writing in the introduction:

> "Symbolic Computation is strong in providing powerful procedures for sets (conjunctions) of arithmetic constraints, but it does not exploit the achievements in SMT solving for efficiently handling logical fragments, using heuristics and learning to speed-up the search for satisfying solutions."

By *heuristic* the we mean a practical method to make a choice which is not guaranteed to be optimal. Although Computer Algebra Systems prize correctness and exact solutions there is still much scope for the use of heuristics and statistical methods in symbolic computation algorithms: both for tuning how individual algorithm are run and for selecting a particular algorithm to use in the first place. In regards to the latter point, we note that the `solve` procedures in Computer Algebra Systems are really meta-algorithms: algorithms to select specific procedures to use based on problem parameters. Although the individual procedures are usually well documented within the scientific literature we are not aware of any publications describing these meta-algorithms.

Another topic where Symbolic Computation might benefit from experience in Satisfiability Checking is standards and benchmarks. Competitions based on these form an integral part of the Satisfiability Checking community, and may have contributed to the remarkable progress made in practical algorithms. The lack of a comparable focus for the Symbolic Computation community was acknowledged in [2]. However, recent experiments have suggested the benchmarks for non-linear real arithmetic are insufficient and the development of new standards and benchmarks for the joint community has been identified as a key SC^2 project action in [2, Section 3.3].

In the present paper we outline our experience of these issues for a single Symbolic Computation algorithm, Cylindrical Algebraic Decomposition (CAD). The aim of the paper is to instigate the learning process from the Satisfiability Checking community by illustrating the current use of heuristics, benchmarks and standards in (at least one area of) Symbolic Computation and posing some questions. We start with a summary of the necessary background on CAD in Section II, then survey work with heuristics in CAD in Section III and our experience with standards and benchmarks in Section IV. We finish with conclusions and questions in Section V.

II. Cylindrical algebraic decomposition

A. Definition

A *Cylindrical Algebraic Decomposition* (CAD) is a *decomposition* of $\mathbb{R}^n$ into *cells* (connected subsets). By *algebraic* we mean semi-algebraic: i.e. each cell can be described with a finite sequence of polynomial constraints. Finally, the cells are arranged *cylindrically*, meaning the projections of any pair, with respect to the variable ordering in which the CAD was created, are either equal or disjoint. We assume variables labelled according to their ordering (so the projections considered are $(x_1, \ldots, x_\ell) \to (x_1, \ldots, x_k)$ for $k < \ell$) with the highest ordered variable present said to be the *main variable*. Hence CADs can be represented in a tree like format branching on the semi-algebraic conditions involving increasing main variable, as in the example below (with the branching from right to left; all $\surd$ indicating the positive root; and the tuples

[1] http://www.sc-square.org/

Copyright © by the paper's authors. Copying permitted for private and academic purposes.

In: E. Ábrahám, J. Davenport, P. Fontaine, (eds.): Proceedings of the 1st Workshop on Satisfiability Checking and Symbolic Computation (SC^2), Timisoara, Romania, 02-07-2016, published at http://ceur-ws.org

on the left sample points of the cells).

$$\begin{cases} (-2,0) & x<-1 \\ \begin{cases} (-1,-1) & y<0 \\ (-1,0) & y=0 \\ (-1,1) & 0<y \end{cases} & x=-1 \\ \begin{cases} (0,-2) & y<-\sqrt{-x^2+1} \\ (0,-1) & y=-\sqrt{-x^2+1} \\ (0,0) & -\sqrt{-x^2+1}<y<\sqrt{-x^2+1} \\ (0,1) & y=+\sqrt{-x^2+1} \\ (0,2) & \sqrt{-x^2+1}<y \end{cases} & -1<x<1 \\ \begin{cases} (1,-1) & y<0 \\ (1,0) & y=0 \\ (1,1) & 0<y \end{cases} & x=1 \\ (2,0) & 1<x \end{cases}$$

A CAD is produced to be invariant for input; originally *sign-invariant* for a set of input polynomials (so on each cell each polynomial is positive, zero or negative). The example above is a sign invariant CAD for the polynomial x^2+y^2-1 defining the unit circle. More recently CADs have been produced *truth-invariant* for input Boolean-valued polynomial formulae. A sign-invariant CAD for the polynomials in a formula is also truth-invariant for the formula; but we can often achieve truth-invariance with far less cells. For example suppose we need a CAD truth-invariant for the formula

$$x^2+y^2-1=0 \wedge (x-1)^2+y^2-1=0.$$

A sign-invariant CAD would require 55 cells (with the full dimensional ones shown on the left of Figure 1). However, a truth-invariant CAD would need only 7 cells: 2 of which are full dimension (as on the right of Figure 1) and 5 more to decompose the line $x=\frac{1}{2}$ at the points of intersection.

B. Computation

CAD construction usually involves two phases. The first *projection*, applies operators recursively on polynomials, starting with the input. Each time the operator produces a set with one less variable which together define the *projection polynomials*. These are used in the second phase, *lifting*, to build CADs of increasing dimension. First a CAD of $\mathbb{R}^1$ is built by splitting on the real roots of the univariate polynomials (those in x_1 only). Next, a CAD of $\mathbb{R}^2$ is built by repeating the process over each cell in $\mathbb{R}^1$ with the bivariate polynomials in (x_1,x_2) evaluated at a sample point of the cell in $\mathbb{R}^1$; and the process is repeated until a CAD of $\mathbb{R}^n$ is produced. We call the cells where a polynomial vanishes *sections* and those regions in-between *sectors*, which together form the *stack* over the cell. In each lift we extrapolate the conclusions drawn from working at a sample point to the whole cell requiring validity theorems for the projection operator used.

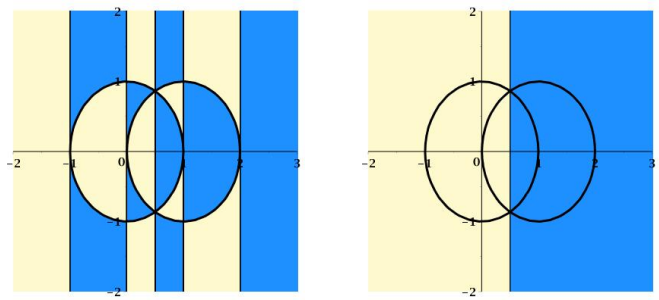

Fig. 1. Example visualising sign and truth-invariant CADs

CAD cells are represented by at least a *sample point* (as in the left of the example above), and an *index*: a list of positive integers with each integer indicating the section or stack each variable is within (in reference to the ordered roots of the projection polynomials). Some implementations will also encode the full algebraic description within each cell.

CAD was originally introduced by Collins for quantifier elimination (QE) in real closed fields [4]. Although CAD construction has complexity doubly exponential in the number of variables [26], applications range from parametric optimisation [34] and epidemic modelling [16], to reasoning with multi-valued functions [24] and the derivation of optimal numerical schemes [33]. There have been many improvements to Collins' original approach most notably refinements to the projection operators [50] [10], [53]; early termination of lifting [22] [62]; and symbolic-numeric schemes [58], [42]. Some recent advances include dealing with multiple formulae [7], [8]; local projection [14], [60]; and decompositions via complex space [21], [6]. For a more detailed introduction to CAD see e.g. [8].

III. Heuristic use for CAD

A. Choosing the variable ordering

The most well known choice required for CAD is that of the variable ordering, which the cylindricity is defined with respect to. This determines the order in which variables are eliminated during projection and the subspaces through which CADs are built incrementally during lifting. When using CAD for QE we must project variables in the order they are quantified, 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 have a great effect on the output produced. For example, let $f := (x-1)(y^2+1)-1$ and consider the minimal sign-invariant CAD in each variable ordering, as visualised in Figure 2. In each case we project down with the left figure projecting x first and the right y. In this toy example the *"wrong"* choice more than doubles the number of cells, while numerous experiments have shown that for larger examples the choice can determine whether a problem is tractable (see for example the experimental results in [8]). At the extreme end of this observation, [15] 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.

Several heuristics have been developed to choose the variable ordering, including:

`Brown`: Use the following criteria, starting with the first and breaking ties with successive ones:

(1) Eliminate variable if lowest overall degree.
(2) Eliminate variable if lowest (maximum) total degree in terms in which it occurs.
(3) Eliminate variable if smallest number of terms contains it.

Suggested by Brown in [13, Section 5.2].

`sotd`: For all admissible orderings, calculate the projection set and choose the one with smallest *sum of total degrees* for each of the monomials in each of the polynomials [27]. Performs well but more costly than `Brown`. A greedy alternative is to allocate one variable of the ordering at a time by projecting each unallocated variable and choosing the one which increases the `sotd` least.

`ndrr`: As with `sotd` construct the full projection set and choose the ordering whose set has the least *number of distinct real roots* of the univariate polynomials within [9]. Even more costly than `sotd`, but sensitive to the real geometry and shown to assist with examples where the `sotd` heuristic failed.

The papers cited each contain experimental results demonstrating their worth. In [39] the results of an experiment comparing the 3 on a data set of over 7000 examples were reported. The experiments showed `Brown` selecting the best ordering (as measured by lowest cell count) more often than the others. However a key finding was that there were substantial subsets of examples for which each heuristic did best. Further, when calculating the saving made by the heuristics (compared to the average cell count of the different orderings) the authors of [39] found that `sotd` actually made a greater saving for quantified problems on average (i.e. while `Brown` was superior on more examples, `sotd` was superior on examples with greater savings on offer). Together, this meant that recommending one heuristic at the expense of all others was not possible.

If the minimal cell count is a priority then one further approach, suggested in [64] is to compute the full dimensional cells for each possible ordering and pick the ordering with the minimum to derive a full CAD. Computing full dimensional cells avoids any work with algebraic numbers and so is not as costly as may be thought, although it does require more computation than `ndrr`. It was noted in [64] that the full-dimensional cell calculations could be done in parallel with the first to finish extended to a full CAD and the rest discarded.

B. Choices with equational constraints

1) Equational constraints: There are several ways in which we can modify CAD constriction to achieve truth-invariance (rather than sign-invariance) including refining sign-invariant CAD and truncating lifting once the truth-value is determined. A particularly fruitful approach is to take advantage of the Booloean structure of a formula through the identification of *Equational Constraints* (ECs): polynomial equations logically implied by a formula. Reduced projection operators (using a subset of the usual polynomials) have been proven valid for use when an EC is present with corresponding main variable [51], [52]. Results in [29], [31] suggest that the double exponent in the complexity bound decreases by 1 for each EC used, although this is restricted to primitive ECs meaning the classic lower-bound examples of [26], [15] are not violated [25].

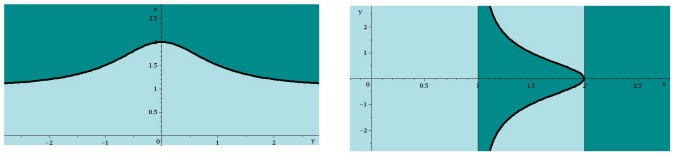

Fig. 2. CADs under different variable orderings

These reduced operators can only use one EC for each projection, so when there are multiple we must make a *designation*. Note that ECs need not appear explicitly as atoms (formula with no logical connectives) in the input formula, but could instead be implicit. For example, the resultant of any two ECs in the same main variable is itself an EC (not containing that variable) [52]. Propagating ECs in this way allows for the maximal use of the reduced operators. However, it can require multiple choices of which EC to designate at each projection. Section 4 of [29] described such an example where the *wrong* designation could make add tens of thousands to the cell count of the final output, making it more than 15 times bigger.

2) Making the designation: In [9] the authors experimented in using the `sotd` and `ndrr` measures on this question (the `Brown` heuristic was not applicable since it acted only on the input polynomials). In general they were useful in identifying the optimal designation, although both could be misled. As described above these heuristics essentially complete the projection stage of the algorithm for each ordering, which although minimal in comparison to the lifting stage, is likely far more computation than would normally be undertaken by a heuristic. This becomes an issue when the number of choices grows. Further, for these experiments a fixed variable ordering was used, and the question of addressing the two choices together (when the number of possibilities multiplies) has not been addressed.

3) Designation in TTICAD: In [7], [8] a truth-table invariant CAD (TTICAD) is 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 present in the separate formulae (with [7] developing the theory in the case where all had an EC and [8] extending to the general case). The operator essentially recognises when to consider the interaction of polynomials from different formulae. If an individual formula has multiple ECs then, as above, we must choose just one to designate for each projection.

C. Choices for CAD by Regular Chains

Recently an alternative CAD computational scheme has been proposed where, instead of projection and lifting, we: first cylindrically decompose complex space according to whether polynomials are zero or not using the theory of triangular decomposition by regular chains; and then refine to a CAD of real space. This was first proposed in [21] with an incremental version described in [20] and an extension to TTICAD in [6]. All versions are implemented within the `RegularChains` Library[2] with a summary in [19].

Most of the heuristics outlined earlier in this section are not directly applicable to the CAD by Regular Chains computation

[2] www.regularchains.org

scheme (as there is no *"cheap"* projection phase to derive information from). We outline some of the new heuristics developed for the choices this scheme involves.

1) Variable order in TTICAD by Regular Chains: This problem was considered in [30]. Two existing heuristics were compared: that of Brown introduced in Section III-A and another, denoted Triangular, already in use for other algorithms in the `RegularChains` Library. Triangular chooses first the variable with lowest degree occurring in the input; then breaks ties by choosing variables for which leading coefficients have lowest total degree; and finally sum of degrees in input. In addition the heuristics `sotd` and `ndrr` discussed above were used (even though the sets of projection polynomials built were not explicitly used later). The experiments found `sotd` to make the best choices, but due to its higher costs the Triangular heuristic was the most efficient choice overall. However, as with the experiments discussed in Section III-A, the example set could be subdivided into groups where different heuristics were dominant. Further experimentation and illustrative examples in [30] led to the development of a new heuristic (composed from parts of the others) tailored to the variable ordering choice for TTICAD by Regular Chains [6].

2) Constraint order in TTICAD by Regular Chains: The latest CAD algorithm within the `RegularChains` Library [20] processes constraints incrementally when building the complex cylindrical decomposition and thus are sensitive to the order in which constraints are considered. Further, in the case of TTICAD we have the extra question of what order to consider the formulae in. These issues were studied in [28] which considered the following example.

Assume the ordering $x \prec y$ and consider

$$\begin{aligned} f_1 &:= x^2 + y^2 - 1, \\ f_2 &:= 2y^2 - x, \\ f_3 &:= (x-5)^2 + (y-1)^2 - 1, \\ \phi_1 &:= f_1 = 0 \wedge f_2 = 0, \\ \phi_2 &:= f_3 = 0. \end{aligned}$$

The polynomials are graphed within the plots of Figure 3. If we want to study the truth of ϕ_1 and ϕ_2 (or say a parent formula $\phi_1 \vee \phi_2$) we need a TTICAD to take advantage of the ECs. There are two possible orders for the formulae and two possible to consider the constraints within ϕ_1. Hence 4 possible ways we could calculate a TTICAD by Regular Chains. Below we show how many cells are produced by proceeding in the orders indicated, with the two dimensional cells shown in Figure 3.

- $\boldsymbol{\phi_1 \to \phi_2}$ **and** $\boldsymbol{f_1 \to f_2}$**:** 37 cells.
- $\boldsymbol{\phi_1 \to \phi_2}$ **and** $\boldsymbol{f_2 \to f_1}$**:** 81 cells.
- $\boldsymbol{\phi_2 \to \phi_1}$ **and** $\boldsymbol{f_1 \to f_2}$**:** 25 cells.
- $\boldsymbol{\phi_2 \to \phi_1}$ **and** $\boldsymbol{f_2 \to f_1}$**:** 43 cells.

No previously discussed heuristic was applicable to this problem. For choosing which EC to process first in a given formula an argument could be made for measuring a set of polynomials shown to be rendered sign-invariant by the algorithm (leading to Heuristic 1 in [28]). The only heuristic derived for the other choices was to measure the sum of degrees of the polynomials in the complex cylindrical decomposition created. As with the `ndrr` and full dimensional cells heuristics above; this requires more computation than is ideal (although it is the real root refinement that makes up the bulk of the CAD by Regular Chains computation time).

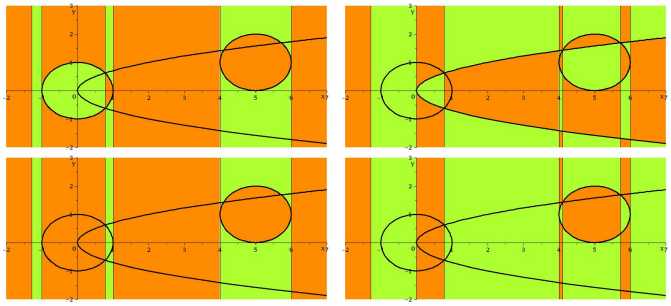

Fig. 3. Visualisations of the four TTICADs which can be built using the Regular Chains Library for the example in this section. The figures on the top have $\phi_1 \to \phi_2$ and those on the bottom $\phi_2 \to \phi_1$. The figures on the left have $f_1 \to f_2$ and those on the right $f_2 \to f_1$.

D. Gröbner Basis preconditioning

A *Gröbner Basis* G is a particular generating set of an ideal I defined with respect to a monomial ordering [17]. One definition is that the ideal generated by the leading terms of I is generated by the leading terms of G. Gröbner Bases (GB) are used extensively to study ideals and the polynomials that define them as they allow properties such as dimension and number of zeros to be easily deduced. Although like CAD the calculation of GB is doubly exponential in the worst case [48], GB computation is now mostly trivial for any problem on which CAD construction is tractable.

It was first observed in [18] that replacing a conjunction of polynomial equalities in a CAD problem by their GB (logically equivalent) could be useful for the CAD computation. Of the ten test problems studied: 6 were improved by the GB preconditioning (speed-up varying from 2- to 1700-fold); 1 problem resulted in a 10-fold slow-down; 1 timed out when GB preconditioning was applied, but would complete without; and 2 were intractable both for CAD construction alone and the GB preconditioning step. The problem was revisited in [66]. As expected, there had been a big decrease in the computation timings, especially for the GB. However, it was still the case that 2 of the problems were hindered by GB preconditioning.

The key conclusion is that GB preconditioning will on average benefit CAD (sometimes very significantly) but could on occasion hinder it (to the point of making a tractable CAD problem intractable). We are yet to understand why this occurs, but the authors of [66] did develop a metric to predict when it will. They defined the *Total Number of Indeterminates* (`TNoI`) of a set of polynomials A as

$$\texttt{TNoI}(A) = \sum_{a \in A} \texttt{NoI}(a)$$

where $\texttt{NoI}(a)$ is the number of indeterminates in a polynomial a. The heuristic is to build a CAD for the preconditioned polynomials only if the `TNoI` decreased. For most of their test problems the heuristic made the correct choice, but there were examples to the contrary.

E. Use of machine learning

Finally, we note recent experiments using machine learning, specifically support vector machines (see for example [56]), to make choices for CAD construction:

- In [40] the authors used an SVM to choose between the three heuristics for CAD variable ordering outlined in Section III-A. Simple problem features were selected (e.g. degrees, proportion of monomials containing each variable) and parameter optimisation was applied to maximise Matthews' Correlation Coefficient [47]. Over 7000 examples were studied, and over the 1721 reserved for the test set the machine learned choice was found to outperform each heuristic individually on average.
- In [38] the authors used an SVM to predict when it will be useful to precondition a CAD problem with GB (see Section III-D). The features used were from both the original input and the GB: so the study was answering the question *should we use this GB* rather than *should we compute it* (relevant since the GB computation was trivial for the problem set involved). The machine learned choice outperformed both using GB universally, and the human defined `TNoI` heuristic.

We also note that a recent paper [45] applied a support vector machine (seeded with the problem features from [40]) to suggest the order in which QE should be performed on subformulae of a non-prenex formula. Experimental results on more than 2,000 non-trivial examples showed that machine learning was doing better than the human derived heuristics, following appropriate parameter optimisation.

IV. Standards and Benchmarks

A. History of benchmarking in computer algebra

The Computer Algebra community has occasionally recognised the importance of benchmarks. The PoSSo and FRISCO projects aimed to do this for polynomials systems and symbolic-numeric problems respectively in the 1990s. PoSSO, with which the second author was involved, collected a series of benchmark examples for GB, and a descendant of these can still be found online[3]. However, this does not appear to be maintained; and the polynomials are not stored in a machine-readable form. Polynomials from this list still crop up in various papers, but there is no systematic reference, and it is not clear whether people are really referring to the same example. Several of the examples are families, which is good but means that a benchmark has to contain specific instances. The PoSSo project did its best to do "level playing field" comparisons, but at the time different implementations ran on different hardware / operating systems meaning this was not really achievable. The environment is much simpler these days, and it would be feasible to organise true contests.

The topic of benchmarking in computer algebra has most recently been taken up by the SymbolicData Project[4] [35] which is beginning to build a database of examples in XML format (although currently not with any suitable for CAD). The software described in [36] was built to translate problems in that database into executable code for various computer algebra systems. The authors of [36] discuss the peculiarities of computer algebra that make benchmarking particularly difficult including the fact that results of computations need not be unique and that the evaluation of the correctness of an output may not be trivial (or may be the subject of research itself).

A final point to note is that while SAT / SMT-solvers have only a few clear possible answers (e.g. sat, unsat, unknown) in computer algebra there is also the quality of the result to consider (e.g., size of quantifier-free formula produced). With CAD the output size is usually correlated to computation time, but this is not always the case with other algorithms.

B. The present authors' recent experience with CAD

In our work we have taken various approaches to experimenting with CAD:

1 (a) Test new results on examples previously used to evaluate CAD algorithms.

For example, the experiments in [66] started with the 10 examples in [18] while [24] and [63] focussed on classic examples from [44] and [23]. The latter were derived from applications of CAD while the former seem to be a collection of geometric problems invented by the authors. Other papers that have contributed such test problems include [5], [49] [11], [27]. We wonder whether historic repetition within the literature is alone a strong enough reason to be benchmark?

In [65], [61] an attempt was made to gather together all those test examples in the literature for CAD, along with references of their first appearance in the literature and encodings for some computer algebra systems.

1 (b) Supplement existing examples with modified versions suitable for demonstrating the feature in question.

In [7], [8] the new TTICAD algorithm that was the subject of the paper offered an improvement on the state of the art for examples consisting of multiple formulae; or a single formulae in disjunctive normal form. Such examples had not been the topic of any CAD papers before and no existing examples were capable of demonstrating the savings on offer. The experiments produced in these papers were made of up two sets:

- Formulae produced to describe the branch cuts of multivalued functions in a proposed simplification formula [30], with CAD to be applied so that the complex domain could be decomposed into regions where the functions were univariate, and thus the formula applicable or not.
- Formulae produced by adapting the logical connectors in previous examples from the literature in [65] so that conjunctions became disjunctions.

Clearly the former set is of great interest as they represent a real example for the algorithms; but they all conform to a single structure and so are arguably too uniform to alone draw broad conclusions from. The second set were produced to be somehow close to the *accepted* test examples of the literature,

[3] http://www-sop.inria.fr/saga/POL/

[4] www.symbolicdata.org

but whether this is any better than inventing a new examples from scratch is debatable.

2 Derive new sets of random examples

A recent experiment using machine learning [40], [38] (see Section III-E) exposed a shortcoming in the above techniques. To train the SVMs hundreds of examples are required (with hundreds more then needed for validation and testing). The dataset from the literature in [65] contained not nearly enough examples and while the datasets discussed in the next section were sufficient for the first experiment in [40] they proved too uniform for [38]. We were left with no choice but to generate large quantities of new examples, which we did using the random polynomial generator in MAPLE. We had applied this technique also in [28], [30] receiving positive feedback from reviewers for the technique; but the initial reviews of [38] were all negative on the use of random data. It seems the appropriateness of this technique varies with the community (conference) in question. We opine that had we used data from the example bank of MetiTarski examples discussed in the next section then reviewers may have praised the focus on examples from a real application; even though MetiTarski themselves derive examples for benchmarks using random polynomials.

C. Sources of large benchmarks sets

We note some other sources of large sets of benchmark problems that represent real applications of CAD:

- MetiTarski[5] [3], [54] is an automatic theorem prover designed to prove theorems involving real-valued special functions (such as log, exp, sin, cos and sqrt). In general this theory is undeciadable but MetiTarski is able to solve many problems by applying real polynomial bounds and then using real quantifier elimination tools like CAD. Applications of MetiTarski in turn derive examples for CAD.
- The NRA (non-linear real arithmetic) category of the SMT-LIB library[6] which according to [43] consists mostly of problems originating from attempts to prove termination of term-rewrite systems.

These two data sets where included in the nlsat Benchmark Set[7] produced to evaluate the work in [43]. This also included verification conditions from the Keymaera [55] and parametrized generalizations of the problem from [37]. Together this dataset had many thousands of problems. However, we note that the problems come from a small number of classes and may have some hidden uniformity.

As mentioned above, the nlsat dataset was unsuitable to use for our machine learning experiment in [38]. Every single problem within that had more than one equality was aided by GB preconditioning, in fact a great many simply had a GB containing only 1 indicating the problem had no solution. Previous experiments on small example sets suggested GB preconditioning sometimes harms CAD computation and this was verified by analysis of a large randomly generated dataset in [38]. Thus while the nlsat dataset is an excellent starting point it needs to be expanded to be less uniform.

We finish this section by noting one possible source of examples for the future.

- The Todai Robot Project[8] [46] is a Japanese AI project that aims to have an artificial intelligence pass the entrance examination for the University of Tokyo by 2021. A majority of questions on the Mathematics exam can be resolved by real quantifier elimination with a variety of techniques employed [41]. A key difficulty is that the natural language processing of the question derives a formula of far greater complexity than the human derived equivalent. This process derives a large bank of examples of CAD problems, as discussed in [45] for example. The authors of [45] told us there are plans to make this data set public.

V. CONCLUSIONS AND QUESTIONS

After surveying the work in Section III we see that several approaches to the creation of heuristics have been taken: ranging from human identified algebraic features, justified by mathematical arguments and observations to different extents; to machine learned choices using a support vector machine. We are interested to hear how these compare with the heuristics used in SAT-solvers and what lessons can be learned.

We can identify at least two areas where CAD is in need of further heuristic development.

- **How best to take decisions in tandem:** The work surveyed all considered choices to be made for CAD *in isolation* (assuming other choices had already been fixed). Of course, in reality this may not be the case. We must decide which decisions to prioritise; how heuristics can be combined; and how the combinatorial blow-up of decisions can be contained. Does the SAT-solving community have experience in similar issues?

There are many implementations of CAD including: the dedicated command line program Qepcad [12]; Mathematica [58], [59], where CAD is not available directly but is used as a subroutine for quantifier elimination; the `Redlog` package for REDUCE [57]; and 3 different MAPLE libraries - the `RegularChains` Library (see Section III-C; `SyNRAC` [67] (now part of the Todai Robot project) and our own `ProjectionCAD` module [32].

- **How to choose between different implementations?** Each implementation includes unique pieces of theory and features and excels on different examples. Ideally, we would have a single implementation which encompasses all recent advances. A more manageable step may be an overarching MAPLE command to choose between the 3 packages there. SMT-solvers are designed to use a variety of different theory solvers and how they choose between these may offer valuable lessons here.

Surveying Section IV raises a number of questions about how benchmark sets should be produced:

[5] https://www.cl.cam.ac.uk/~lp15/papers/Arith

[6] http://smtlib.cs.uiowa.edu/

[7] http://cs.nyu.edu/~dejan/nonlinear/

[8] http://21robot.org

- How best to generate large numbers of examples which are not internally uniform?
- How important is it that the benchmarks come from current applications?
- How important is it that the benchmarks have historically been used in the literature?
- Are randomly generated examples a fairer way to evaluate the software, or irrelevant as too far removed from applications?

The SAT / SMT community posses a unified, large and growing set of benchmarks in the SMT-LIB library and so we may be able to extrapolate lessons from this. However, as noted above, this library may be too uniform [38], and comments from the anonymous referees suggest that this and other critics are already a topic of discussion in the SMT community.

Acknowledgements

Thanks to our collaborators on the work surveyed here: Russell Bradford, James Bridge, Changbo Chen, Zongyan Huang, Scott McCallum, Marc Moreno Maza, Lawrence Paulson, and David Wilson. Thanks also to the anonymous referees for their comments which improved the paper.

Most of the work surveyed here was supported by EPSRC grant EP/J003247/1. The authors are now supported by EU H2020-FETOPEN-2016-2017-CSA project $\mathcal{SC}^2$ (712689).

REFERENCES

[1] E. Ábráham. Building bridges between symbolic computation and satisfiability checking. In *Proceedings of the 2015 International Symposium on Symbolic and Algebraic Computation*, ISSAC '15, pages 1–6. ACM, 2015.

[2] E. Ábrahám, J. Abbott, B. Becker, A.M. Bigatti, M. Brain, B. Buchberger, A. Cimatti, J.H. Davenport, M. England, P. Fontaine, S. Forrest, A. Griggio, D. Kroening, W.M. Seiler, and T. Sturm. SC^2: Satisfiability checking meets symbolic computation. In M. Kohlhase, M. Johansson, B. Miller, L. de Moura, and F. Tompa, editors, *Intelligent Computer Mathematics: Proceedings CICM 2016*, LNCS 9791, pages 28–43. Springer International Publishing, 2016.

[3] B. Akbarpour and L.C. Paulson. MetiTarski: An automatic theorem prover for real-valued special functions. *Journal of Automated Reasoning*, 44(3):175–205, 2010.

[4] D. Arnon, G.E. Collins, and S. McCallum. Cylindrical algebraic decomposition I: The basic algorithm. *SIAM Journal of Computing*, 13:865–877, 1984.

[5] D.S. Arnon. A cluster-based cylindrical algebraic decomposition algorithm. *Journal of Symbolic Computation*, 5(1-2):189–212, 1988.

[6] R. Bradford, C. Chen, J.H. Davenport, M. England, M. Moreno Maza, and D. Wilson. Truth table invariant cylindrical algebraic decomposition by regular chains. In V.P. Gerdt, W. Koepf, W.M. Seiler, and E.V. Vorozhtsov, editors, *Computer Algebra in Scientific Computing*, LNCS 8660, pages 44–58. Springer International Publishing, 2014.

[7] R. Bradford, J.H. Davenport, M. England, S. McCallum, and D. Wilson. Cylindrical algebraic decompositions for boolean combinations. In *Proceedings of the 38th International Symposium on Symbolic and Algebraic Computation*, ISSAC '13, pages 125–132. ACM, 2013.

[8] R. Bradford, J.H. Davenport, M. England, S. McCallum, and D. Wilson. Truth table invariant cylindrical algebraic decomposition. *Journal of Symbolic Computation*, 76:1–35, 2015.

[9] R. Bradford, J.H. Davenport, M. England, and D. Wilson. Optimising problem formulations for cylindrical algebraic decomposition. In J. Carette, D. Aspinall, C. Lange, P. Sojka, and W. Windsteiger, editors, *Intelligent Computer Mathematics*, LNCS 7961, pages 19–34. Springer Berlin Heidelberg, 2013.

[10] C.W. Brown. Improved projection for cylindrical algebraic decomposition. *Journal of Symbolic Computation*, 32(5):447–465, 2001.

[11] C.W. Brown. Simple CAD construction and its applications. *Journal of Symbolic Computation*, 31(5):521–547, 2001.

[12] C.W. Brown. An overview of QEPCAD B: a tool for real quantifier elimination and formula simplification. *Journal of Japan Society for Symbolic and Algebraic Computation*, 10(1):13–22, 2003.

[13] 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.

[14] C.W. Brown. Constructing a single open cell in a cylindrical algebraic decomposition. In *Proceedings of the 38th International Symposium on Symbolic and Algebraic Computation*, ISSAC '13, pages 133–140. ACM, 2013.

[15] C.W. Brown and J.H. Davenport. The complexity of quantifier elimination and cylindrical algebraic decomposition. In *Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation*, ISSAC '07, pages 54–60. ACM, 2007.

[16] C.W. Brown, M. El Kahoui, D. Novotni, and A. Weber. Algorithmic methods for investigating equilibria in epidemic modelling. *Journal of Symbolic Computation*, 41:1157–1173, 2006.

[17] B. Buchberger. Bruno Buchberger's PhD thesis (1965): An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal. *Journal of Symbolic Computation*, 41(3-4):475–511, 2006.

[18] B. Buchberger and H. Hong. Speeding up quantifier elimination by Gröbner bases. Technical report, 91-06. RISC, Johannes Kepler University, 1991.

[19] C. Chen and M. Moreno Maza. Cylindrical algebraic decomposition in the RegularChains library. In H. Hong and C. Yap, editors, *Mathematical Software – ICMS 2014*, LNCS 8592, pages 425–433. Springer Heidelberg, 2014.

[20] C. Chen and M. Moreno Maza. An incremental algorithm for computing cylindrical algebraic decompositions. In R. Feng, W. Lee, and Y. Sato, editors, *Computer Mathematics*, pages 199—221. Springer Berlin Heidelberg, 2014.

[21] C. Chen, M. Moreno Maza, B. Xia, and L. Yang. Computing cylindrical algebraic decomposition via triangular decomposition. In *Proceedings of the 2009 International Symposium on Symbolic and Algebraic Computation*, ISSAC '09, pages 95–102. ACM, 2009.

[22] G.E. Collins and H. Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *Journal of Symbolic Computation*, 12:299–328, 1991.

[23] J.H. Davenport. A "Piano-Movers" Problem. *SIGSAM Bull.*, 20(1-2):15–17, 1986.

[24] J.H. Davenport, R. Bradford, M. England, and D. Wilson. Program verification in the presence of complex numbers, functions with branch cuts etc. In *14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, SYNASC '12, pages 83–88. IEEE, 2012.

[25] J.H. Davenport and M. England. Need polynomial systems be doubly exponential? In G.-M. Greuel, T. Koch, P. Paule, and A. Sommese, editors, *Mathematical Software – Proceedings of ICMS 2016*, LNCS 9725, pages 157–164. Springer International Publishing, 2016.

[26] J.H. Davenport and J. Heintz. Real quantifier elimination is doubly exponential. *Journal of Symbolic Computation*, 5(1-2):29–35, 1988.

[27] A. Dolzmann, A. Seidl, and T. Sturm. Efficient projection orders for CAD. In *Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation*, ISSAC '04, pages 111–118. ACM, 2004.

[28] 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. In S.M. Watt, J.H. Davenport, A.P. Sexton, P. Sojka, and J. Urban, editors, *Intelligent Computer Mathematics*, LNCS 8543, pages 45–60. Springer International, 2014.

[29] M. England, R. Bradford, and J.H. Davenport. Improving the use of equational constraints in cylindrical algebraic decomposition. In *Proceedings of the 2015 International Symposium on Symbolic and Algebraic Computation*, ISSAC '15, pages 165–172. ACM, 2015.

[30] M. England, R. Bradford, J.H. Davenport, and D. Wilson. Choosing a variable ordering for truth-table invariant cylindrical algebraic decomposition by incremental triangular decomposition. In H. Hong and C. Yap, editors, *Mathematical Software – ICMS 2014*, LNCS 8592, pages 450–457. Springer Heidelberg, 2014.

[31] M. England and J.H. Davenport. The complexity of cylindrical algebraic decomposition with respect to polynomial degree. *To appear In: Proceedings CASC 2016 (Springer LNCS)*, 2016.

[32] M. England, D. Wilson, R. Bradford, and J.H. Davenport. Using the Regular Chains Library to build cylindrical algebraic decompositions by projecting and lifting. In H. Hong and C. Yap, editors, *Mathematical Software – ICMS 2014*, LNCS 8592, pages 458–465. Springer Heidelberg, 2014.

[33] M. Erascu and H. Hong. Synthesis of optimal numerical algorithms using real quantifier elimination (Case Study: Square root computation). In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, ISSAC '14, pages 162–169. ACM, 2014.

[34] I.A. Fotiou, P.A. Parrilo, and M. Morari. Nonlinear parametric optimization using cylindrical algebraic decomposition. In *Decision and Control, 2005 European Control Conference. CDC-ECC '05.*, pages 3735–3740, 2005.

[35] H.G. Graebe, A. Nareike, and S. Johanning. The SymbolicData project: Towards a computer algebra social network. In M. England, J.H. Davenport, A. Kohlhase, M. Kohlhase, P. Libbrecht, W. Neuper, P. Quaresma, A.P. Sexton, P. Sojka, J. Urban, and S.M. Watt, editors, *Joint Proceedings of the MathUI, OpenMath and ThEdu Workshops and Work in Progress track at CICM*, number 1186 in CEUR Workshop Proceedings, 2014.

[36] A. Heinle and V. Levandovskyy. The SDEval benchmarking toolkit. *ACM Communications in Computer Algebra*, 49(1):1–9, 2015.

[37] H. Hong. Comparison of several decision algorithms for the existential theory of the reals. Technical report, RISC, Linz, 1991.

[38] Z. Huang, M. England, J.H. Davenport, and L. Paulson. Using machine learning to decide when to precondition cylindrical algebraic decomposition with Groebner bases. In *18th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, SYNASC '16. Preprint: Arxiv 1608.04219. IEEE, 2016.

[39] Z. Huang, M. England, D. Wilson, J.H. Davenport, and L. Paulson. A comparison of three heuristics to choose the variable ordering for cad. *ACM Communications in Computer Algebra*, 48(3):121–123, 2014.

[40] 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. In S.M. Watt, J.H. Davenport, A.P. Sexton, P. Sojka, and J. Urban, editors, *Intelligent Computer Mathematics*, LNAI 8543, pages 92–107. Springer International, 2014.

[41] H. Iwane, T. Matsuzaki, N.H. Arai, and H. Anai. Automated natural language geometry math problem solving by real quantifier elimination. In *Proceedings of the 10th International Workshop on Automated Deduction in Geometry*, ADG '14, pages 75–84, 2014.

[42] H. Iwane, H. Yanami, H. Anai, and K. Yokoyama. An effective implementation of a symbolic-numeric cylindrical algebraic decomposition for quantifier elimination. In *Proceedings of the 2009 conference on Symbolic Numeric Computation*, SNC '09, pages 55–64, 2009.

[43] D. Jovanovic and L. de Moura. Solving non-linear arithmetic. In B. Gramlich, D. Miller, and U. Sattler, editors, *Automated Reasoning: 6th International Joint Conference (IJCAR)*, LNCS 7364, pages 339–354. Springer, 2012.

[44] W. Kahan. Branch cuts for complex elementary functions. In A. Iserles and M.J.D. Powell, editors, *Proceedings The State of Art in Numerical Analysis*, pages 165–211. Clarendon Press, 1987.

[45] M. Kobayashi, H. Iwane, T. Matsuzaki, and H. Anai. Efficient subformula orders for real quantifier elimination of non-prenex formulas. In S.I. Kotsireas, M.S. Rump, and K.C. Yap, editors, *Mathematical Aspects of Computer and Information Sciences (MACIS '15)*, LNCS 9582, pages 236–251. Springer International Publishing, 2016.

[46] T. Matsuzaki, H. Iwane, H. Anai, and N. Arai. The most uncreative examinee: A first step toward wide coverage natural language math problem solving. In C.E. Brodley and P. Stone, editors, *Proceedings of 28th Conference on Artificial Intelligence*, AAAI '14, pages 1098–1104. AAAI Press, 2014.

[47] B.W. Matthews. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure*, 405(2):442–451, 1975.

[48] E.W. Mayr and A.R. Meyer. The complexity of the word problems for commutative semigroups and polynomial ideals. *Advances in Mathematics*, 46(3):305–329, 1982.

[49] S. McCallum. An improved projection operation for cylindrical algebraic decomposition of three-dimensional space. *Journal of Symbolic Computation*, 5(1-2):141–161, 1988.

[50] S. McCallum. An improved projection operation for cylindrical algebraic decomposition. In B. Caviness and J. Johnson, editors, *Quantifier Elimination and Cylindrical Algebraic Decomposition*, Texts & Monographs in Symbolic Computation, pages 242–268. Springer-Verlag, 1998.

[51] S. McCallum. On projection in CAD-based quantifier elimination with equational constraint. In *Proceedings of the 1999 International Symposium on Symbolic and Algebraic Computation*, ISSAC '99, pages 145–149. ACM, 1999.

[52] S. McCallum. On propagation of equational constraints in CAD-based quantifier elimination. In *Proceedings of the 2001 International Symposium on Symbolic and Algebraic Computation*, ISSAC '01, pages 223–231. ACM, 2001.

[53] S. McCallum, A. Parusiński, and L. Paunescu. Validity proof of Lazard's method for CAD construction. *Preprint: Arxiv 1607:00264*, 2016.

[54] L.C. Paulson. Metitarski: Past and future. In L. Beringer and A. Felty, editors, *Interactive Theorem Proving*, LNCS 7406, pages 1–10. Springer, 2012.

[55] A. Platzer, J.D. Quesel, and P. Rümmer. Real world verification. In R.A. Schmidt, editor, *Automated Deduction (CADE-22)*, LNCS 5663, pages 485–501. Springer Berlin Heidelberg, 2009.

[56] B. Schölkopf, K. Tsuda, and J.-P. Vert. *Kernel methods in computational biology*. MIT Press, 2004.

[57] A. Seidl and T. Sturm. A generic projection operator for partial cylindrical algebraic decomposition. In *Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation*, ISSAC '03, pages 240–247. ACM, 2003.

[58] A. Strzeboński. Cylindrical algebraic decomposition using validated numerics. *Journal of Symbolic Computation*, 41(9):1021–1038, 2006.

[59] A. Strzeboński. Computation with semialgebraic sets represented by cylindrical algebraic formulas. In *Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation*, ISSAC '10, pages 61–68. ACM, 2010.

[60] A. Strzeboński. Cylindrical algebraic decomposition using local projections. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, ISSAC '14, pages 389–396. ACM, 2014.

[61] D. Wilson. Real geometry and connectedness via triangular description: Cad example bank, 2013.

[62] D. Wilson, R. Bradford, J.H. Davenport, and M. England. Cylindrical algebraic sub-decompositions. *Mathematics in Computer Science*, 8:263–288, 2014.

[63] D. Wilson, J.H. Davenport, M. England, and R. Bradford. A "piano movers" problem reformulated. In *15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, SYNASC '13, pages 53–60. IEEE, 2013.

[64] D. Wilson, M. England, J.H. Davenport, and R. Bradford. Using the distribution of cells by dimension in a cylindrical algebraic decomposition. In *16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, SYNASC '14, pages 53–60. IEEE, 2014.

[65] D.J. Wilson, R.J. Bradford, and J.H. Davenport. A repository for CAD examples. *ACM Communications in Computer Algebra*, 46(3):67–69, 2012.

[66] D.J. Wilson, R.J. Bradford, and J.H. Davenport. Speeding up cylindrical algebraic decomposition by Gröbner bases. In J. Jeuring, J.A. Campbell, J. Carette, G. Reis, P. Sojka, M. Wenzel, and V. Sorge, editors, *Intelligent Computer Mathematics*, LNCS 7362, pages 280–294. Springer, 2012.

[67] H. Yanami and H. Anai. Development of SyNRAC. In *Proceedings of the 6th international conference on Computational Science: Part II. (LNCS vol 3992)*, ICCS '06, pages 462–469, 2006.