The complexity of cylindrical algebraic decomposition with respect to polynomial degree

Matthew England[1] and James H. Davenport[2]

[1] School of Computing, Electronics & Maths,
Faculty of Engineering, Environment & Computing,
Coventry University, Coventry, CV1 5FB, UK
`Matthew.England@coventry.ac.uk`,
WWW home page: `http://computing.coventry.ac.uk/~mengland/`

[2] Department of Computer Science,
University of Bath, Bath, BA2 7AY, UK
`J.H.Davenport@bath.ac.uk`,
WWW home page: `http://people.bath.ac.uk/masjhd/`

Abstract. Cylindrical algebraic decomposition (CAD) is an important tool for working with polynomial systems, particularly quantifier elimination. However, it has complexity doubly exponential in the number of variables. The base algorithm can be improved by adapting to take advantage of any equational constraints (ECs): equations logically implied by the input. Intuitively, we expect the double exponent in the complexity to decrease by one for each EC. In ISSAC 2015 the present authors proved this for the factor in the complexity bound dependent on the number of polynomials in the input. However, the other term, that dependent on the degree of the input polynomials, remained unchanged. In the present paper the authors investigate how CAD in the presence of ECs could be further refined using the technology of Gröbner Bases to move towards the intuitive bound for polynomial degree.

Keywords: computer algebra, cylindrical algebraic decomposition, equational constraint, Gröbner bases, quantifier elimination

1 Introduction

A *cylindrical algebraic decomposition* (CAD) is a *decomposition* of $\mathbb{R}^n$ (under a given variable ordering, so that the projections considered are $(x_1, \ldots, x_\ell) \to (x_1, \ldots, x_k)$ for $k < \ell$) into cells. The cells are arranged *cylindrically*, meaning the projections of any pair with respect to the given ordering are either equal or disjoint. In this definition *algebraic* is short for semi-algebraic meaning each CAD cell can be described with a finite sequence of polynomial constraints. 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), and more recently *truth-invariant* for input Boolean-valued formulae built from the polynomials (so on each cell each formula is either true or false).

Introduced by Collins for quantifier elimination (QE) in real closed fields [1], applications of CAD included parametric optimisation [24], epidemic modelling [10] and even motion planning [38]. Some recent applications include theorem proving [37], the derivation of optimal numerical schemes [22] and reasoning with multi-valued functions [18].

CAD has worst case complexity doubly exponential [9,19], due to the nature of the information to be recorded rather than the algorithm used [9]. Let n be the number of variables, m the number of input polynomials, and d the maximum degree (in any one variable) of the input. Then a complexity analysis in Section 5 of [21] shows that the best known variant of Collins' algorithm to produce a sign-invariant CAD for the polynomials [34] has an upper bound on the size of the CAD (i.e. number of cells) with dominant term

$$(2d)^{2^n-1}m^{2^n-1}2^{2^{n-1}-1}, \tag{1}$$

i.e. the CAD grows doubly exponentially with the number of variables n.

In fact, at the end of the projection stage, when we are considering $\mathbb{R}^1$, this analysis shows that we have M polynomials, each of degree D, where $D = d^{2^{O(n)}}$ and $M = m^{2^{O(n)}}$. Of course, by replacing $\{f, g\}$ by $\{fg\}$ we can reduce M at the cost of increasing D, but since it is much easier to find the roots of $\{f, g\}$ than $\{fg\}$, we do not want to do so. The lower bound in [19] shows that $D = d^{2^{\Omega(n)}}$, and that of [9] shows that, without artificial combination, $M = m^{2^{\Omega(n)}}$. Both rely on the technique from [26], and the formulae demonstrating this growth are not straightforward: in particular needing $O(n)$ quantifier alternations. But the underlying polynomials *are* simple: all linear for [9] and all bar two linear for [19]. Furthermore, each polynomial only involves a bounded number of variables (generally two), independent of n, showing that the doubly-exponential difficulty of CAD resides in the complicated number of ways simple polynomials can interact.

An approach to improve the performance of CAD and thus this bound is to build CADs which are not sign-invariant for polynomials but truth-invariant for formulae. This can be achieved by identifying *equational constraints* (ECs): polynomial equations logically implied by formulae. The presence of an EC restricts the dimension of the solution space and if exploited properly by the algorithm we may expect a reduction in complexity accordingly. Intuitively, we may expect the double exponent to decrease by 1 for each independent (in a sense to be made precise later) EC available.

In [21] the present authors described how to adapt CAD to make use of multiple (primitive) ECs. Suppose that our input formula consists of polynomials (as described above) and that ℓ suitable ECs can be identified. The algorithm in [21] was shown to have corresponding upper bound dominant term

$$(2d)^{2^n-1}m^{2^{n-\ell}-2}2^{\ell 2^{n-\ell}-3\ell}. \tag{2}$$

So while the bound is still doubly exponential with respect to n, some of the double exponents have been reduced by ℓ. To be precise, the double exponent of

m (and its corresponding constant factor) is reduced while the double exponent with respect to d (actually $2d$) has not. This is due to the focus of [21] being on reducing the number of polynomials created during the intermediate calculations with no attempt made to control degree growth.

The present paper is concerned with how to gain the corresponding improvement to the factor dependent on d to achieve the intuitive complexity bound. The hypothesis is that this should be possible by making use of the theory of Gröbner bases in place of iterated resultants. We start in Sections 2.1−2.2 by reviewing background material on CAD, and then focus on CAD in the presence of ECs in Sections 2.3−2.4. In Section 3 we consider how the growth of degree in iterated resultants grows compared to that of the true multivariate resultant (which encodes what is needed for CAD). In Section 3.3 we propose controlling this using Gröbner Bases and in Section 4 we give a worked example of how these can precondition CAD. In Section 5 we sketch how this improves upon the bound (2) and then we finish in Section 6 by discussing some outstanding issues.

2 CAD with respect to equational constraints

2.1 CAD computation scheme and terminology

We describe the computation scheme and terminology that CAD algorithms derived from Collins share. We assume a set of input polynomials (possibly derived from input formulae) in ordered variables $\boldsymbol{x} = x_1 \prec \ldots \prec x_n$. The *main variable* of a polynomial (mvar) is the highest ordered variable present.

The first phase of CAD, *projection*, applies projection operators recursively on the input polynomials, each time producing another set of polynomials with one less variable. Together these define the *projection polynomials* used in the second phase, *lifting*, to build CADs incrementally by dimension. First a CAD of the real line is built with cells (points and intervals) determined by 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 $\mathbb{R}^1$. This produces *sections* (where a polynomial vanishes) and *sectors* (the regions between) which together form the *stack* over the cell. Taking the union of these stacks gives the CAD of $\mathbb{R}^2$. The process is repeated until a CAD of $\mathbb{R}^n$ is produced.

All cells are represented by (at least) a sample point and an *index*. The latter is a list of integers, with the kth integer fixing variable x_k according to the ordered real roots of the projection polynomials in $(x_1, \ldots, x_k)$. If the integer is $2i$ the cell is over the ith root (counting low to high) and if $2i + 1$ over the interval between the ith and $(i+1)$th (or the unbounded intervals at either end).

In each lift we extrapolate the conclusions drawn from working at a sample point to the whole cell. The validity of this approach follows from the correct choice of projection operator. For sign-invariance to be maintained the operator must produce polynomials: *delineable* in a cell, meaning the portion of their zero set in the cell consists of disjoint sections; and, *delineable* as a set, meaning the sections of different polynomials are identical or disjoint. One of the projection operators used in this paper is

$$P(B) := \mathrm{coeff}(B) \cup \mathrm{disc}(B) \cup \mathrm{res}(B). \tag{3}$$

Here disc and coeff denote respectively the set of discriminants and coefficients of a set of polynomials; and res denotes either the resultant of a pair of polynomials or, when applied to a set, the set of polynomials

$$\mathrm{res}(A) = \{\mathrm{res}(f_i, f_j) \mid f_i \in A, f_j \in A, f_j \neq f_i\}.$$

We assume B is an irreducible basis for a set of polynomials in which every element has mvar x_n. For a general set of polynomials A we would proceed by letting B be an irreducible basis of the primitive part of A; apply the operators above; and take the union of the output with the content of A. The operator P was introduced in [34] along with proofs of related delineability results.

2.2 Brief summary of improvements to CAD

As discussed in the introduction, CAD has worst case complexity doubly exponential in the number of variables. For some problems there exist algorithms with better complexity [2], however, CAD implementations remain the best general purpose approach for many. This is due in large part to the numerous techniques developed to improve the efficiency of CAD since Collins' original work including: refinements to the projection operator [27], [34] [7], [25]; the early termination of lifting, such as when sufficient for QE [17] or for building a sub-CAD [41]; and symbolic-numeric lifting schemes [39], [30]. Some recent advances include further refinements to the projection operator when dealing with multiple formulae as input [4], [5]; local projection approaches [8], [40]; decompositions via complex space [14], [3]; and the development of heuristics for CAD problem formulation [6], [20], [42] including machine learned approaches [29].

2.3 Equational constraints

As discussed in the Introduction, identifying equational constraints can improve the performance of CAD.

Definition. A *QFF* is a quantifier free Tarski formula: a Boolean combination $(\wedge, \vee, \neg)$ of statements about the signs $(= 0, > 0, < 0)$ of integral polynomials.

An *equational constraint* (EC) is a polynomial equation logically implied by a QFF. An EC is said to be *explicit* if it is an atom of the QFF, and *implicit* otherwise.

Collins first suggested that the projection phase of CAD could be simplified in the presence of an EC [16]. The insight is that a CAD sign-invariant for the defining polynomial of an EC, and sign-invariant for any others only on sections of that polynomial, would be sufficient. The intuitive restriction of (3) is to use only those coefficients, discriminants and resultants which are derived from the EC polynomial, as in (4) below where $F \subseteq B$ is a basis for the EC polynomial.

$$P_F(B) := P(F) \cup \{\mathrm{res}(f, g) \mid f \in F, g \in B \setminus F\} \tag{4}$$

The validity of using this operator for the first projection was verified in [35], with subsequent projections returning to (3). The operator could only be used for a single EC in the main variable of the system as the delineability result for (4) could not be applied recursively, excluding its use at a subsequent projection to take advantage of any EC with corresponding main variable. This led to the development of the operator (5) in [36] which suffered no such reduction at the cost of including the discriminants that had been removed from (3) by (4).

$$P_F^*(B) := P_F(B) \cup \mathrm{disc}(B \setminus F) \tag{5}$$

See Section 2 of [21] for examples demonstrating these operators. A system to derive implicit ECs was also introduced by [36], based on the observation that the resultant of the polynomials defining two ECs itself defines an EC. This is essential for maximising the savings from ECs since the reduced operators (4), (5) are for use with a single EC; meaning the savings gained are dependent not on the number of ECs, but the number identified with different main variables.

In [21] the present authors reviewed the theory of reduced projection operators and deduced how it could also yield savings in the lifting phase; reducing both the number of cells we must lift over with respect to polynomials; and the number of such polynomials we lift with. These approaches meant that the projection polynomials are no longer a fixed set (key to some CAD implementations) and that the invariance structure of the final CAD can no longer be expressed in terms of sign-invariance of polynomials. For the worked example in [21, Section 4] combining the advances in this subsection allowed a sign-invariant CAD with 1,118,205 cells to be replaced by a truth invariant CAD with 93 cells.

2.4 CAD with ECs

Algorithm 1 describes the CAD projection phase in the presence of multiple ECs described in the previous subsection. Note that (as with the previous theory of multiple ECs this is based on) we assume the ECs are primitive. Algorithm 1 applies the best possible (smallest validated) projection operator at each stage. The word *suitable* in the output declaration means a CAD lifting phase that makes well-orientedness checks in line with the theory of McCallum's projection operators (see [34], [35], [36] for details).

Algorithm 2 is one such suitable lifting algorithm. It uses the F_i (knowledge of which projection steps made use of an EC) to tailor its lifts: lifting only with respect to EC polynomials (steps 7−10) and only over cells where an EC was satisfied (steps 11−15) (lifting trivially to the cylinder otherwise). The correctness of these algorithms was proven in [21].

Algorithm 1: CAD Projection using multiple stated ECs

```
Input : A formula $\phi$ in variables $x_1, \ldots, x_n$, and a sequence of sets $\{E_k\}_{k=1}^{n}$;
        each either empty or containing a single primitive polynomial with
        mvar $x_k$ which defines an EC for $\phi$.
Output: A sequence of sets of polynomials ready for a suitable CAD lifting
        algorithm.
1  Extract from $\phi$ the set of defining polynomials $A_n$;
2  for $k = n, \ldots, 2$ do
3      Set $B_k$ to the finest squarefree basis for prim$(A_k)$;
4      Set $C$ to cont$(A_k)$;
5      Set $F_k$ to the finest squarefree basis for $E_k$;
6      if $F_k$ is empty then
7          Set $A_{k-1} := C \cup P(B_k)$;
8      else
9          Set $A_{k-1} := C \cup P^*_{F_i}(B_i)$;
10 return $A_1, \ldots, A_n; F_1, \ldots, F_n$.
```

Table 1 is recreated from [21] and shows the growth in the number and degree of the projection polynomials when following Algorithm 1 under the assumption that we have declared ECs for the first ℓ projections (so $0 < \ell \leq \min(m, n)$). Rather than the actual polynomials created the table keeps track of sets of polynomials known to have the *(M,D)-property*: the ability to be partitioned into M subsets, each with maximum combined degree D.

The (M,D)-property was introduced in McCallum's thesis and was used (along with tables like Table 1) to give a detailed comparison of the complexity of several different projection operators in [5, Section 2.3]. The key observation is that the number of real roots in a set with the (M,D)-property is at most MD (although in practice many will be in $\mathbb{C} \setminus \mathbb{R}$). Hence the number of cells in the CAD of $\mathbb{R}^1$ is bounded by twice the product of the final entries, plus 1.

Define d_i and m_i as the entries in the Number and Degree columns of Table 1 from the row with i Variables. Then the number of cells in the final CAD of $\mathbb{R}^n$ is bounded by

$$\prod_{i=1}^{n} [2m_i d_i + 1] \,. \tag{6}$$

Omitting the +1s from each term will usually allow for a closed form expression of the dominant term in the bound.

The derivation of bound (2) from Table 1 was given in [21, Section 5]. It involved considering the two improvements to the lifting phase. The first was lifting only with respect to EC polynomials; meaning that for the purposes of the bound we could set m_i to 1 for $i = n, \ldots, n - \ell$. The second was to lift trivially (to a cylinder) over those cells where an EC was false.

Denote by (†) the bound on the CAD of $\mathbb{R}^{n-(\ell+1)}$ given by (6) but with the product terminating at $n - (\ell + 1)$, as there can be no reduced lifting until this

Algorithm 2: CAD Lifting using multiple stated ECs

```
Input : The output of Algorithm 1: two sequences of polynomials sets
        $A_1, \dots, A_n; F_1, \dots, F_n$, the latter subsets of the former.
Output: Either: $\mathcal{D}$, a truth-invariant CAD of $\mathbb{R}^n$ for $\phi$ (described by lists $I$ and
        $S$ of cell indices and sample points); or FAIL, if not well-oriented.

1  If $F_1$ is not empty then set $p$ to be its element; otherwise set $p$ to the product of
   polynomials in $A_1$;
2  Build $\mathcal{D}_1 := (I_1, S_1)$ according to the real roots of $p$;
3  if $n = 1$ then
4      return $\mathcal{D}_1$;
5  for $k = 2, \dots, n$ do
6      Initialise $\mathcal{D}_k = (I_k, S_k)$ with $I_k$ and $S_k$ empty sets;
7      if $F_k$ is empty then
8          Set $L := B_k$;
9      else
10         Set $L := F_k$;
11     if $F_{k-1}$ is empty then
12         Set $\mathcal{C}_a := \mathcal{D}_{k-1}$ and $\mathcal{C}_b$ empty;
13     else
14         Set $\mathcal{C}_a$ to be cells in $\mathcal{D}_{k-1}$ with $I_{k-1}[-1]$ even;
15         Set $\mathcal{C}_b := \mathcal{D}_{k-1} \setminus \mathcal{C}_a$;
16     for each cell $c \in \mathcal{C}_a$ do
17         if An element of $L$ is nullified over $c$ then
18             return FAIL;
19         Generate a stack over $c$ with respect to the polynomials in $L$, adding
           cell indices and sample points to $I_k$ and $S_k$;
20     for each cell $c \in \mathcal{C}_b$ do
21         Extend to a single cell in $\mathbb{R}^k$ (cylinder over $c$), adding index and sample
           point to $I_k$ and $S_k$;
22 return $\mathcal{D}_n = (I_n, S_n)$.
```

point. The lift to $\mathbb{R}^{n-\ell}$ will involve stack generation over all cells, but only with respect to the EC which has at most $d_{n-\ell}$ real roots and thus the CAD of $\mathbb{R}^{n-\ell}$ at most $[2d_{n-\ell}+1](\dagger)$ cells. The next lift, to $\mathbb{R}^{n-\ell-1}$, will lift the sections with respect to the EC, and the sectors only trivially. Hence the cell count bound is $[2d_{n-(\ell-1)}+1]d_{n-\ell}(\dagger) + (d_{n-\ell}+1)(\dagger)$ with dominant term $2d_{n-(\ell-1)}d_{n-\ell}(\dagger)$.

Subsequent lifts follow the same pattern and so the dominant term (omitting the +1s) in the cell count bound for $\mathbb{R}^n$ is

$$2d_n d_{n-1} \dots d_{n-(\ell-1)} d_{n-\ell} \prod_{i=1}^{n-(\ell+1)} \big[2m_i d_i + 1\big]. \tag{7}$$

As shown in [21] using Table 1 (7) evaluates to (2).

Table 1. Projection in CAD with projection operator (5) ℓ times and then (3).

Variables	**Number**	**Degree**
n	m	d
$n-1$	$2m$	$2d^2$
$n-2$	$4m$	$8d^4$
$\vdots$	$\vdots$	$\vdots$
$n-\ell$	$2^{\ell}m$	$2^{2^{\ell}-1}d^{2^{\ell}}$
$n-(\ell+1)$	$2^{2\ell}m^2$	$2^{2^{\ell+1}-1}d^{2^{\ell+1}}$
$n-(\ell+2)$	$2^{4\ell}m^4$	$2^{2^{\ell+2}-1}d^{2^{\ell+2}}$
$\vdots$	$\vdots$	$\vdots$
$n-(\ell+r)$	$2^{2^{r}\ell}m^{2^{r}}$	$2^{2^{\ell+r}-1}d^{2^{\ell+r}}$
$\vdots$	$\vdots$	$\vdots$
1	$2^{2^{(n-1-\ell)}\ell}m^{2^{n-1-\ell}}$	$2^{2^{n-1}-1}d^{2^{n-1}}$

3 Controlling degree growth

3.1 Iterated resultant calculations

As discussed in the Introduction, [21] showed that building truth-invariant CADs by taking advantage of ECs reduced the CAD complexity bound from (1) to (2). Most notably, the double exponent of the term with base m (number of input polynomials) decreased by ℓ; the number of projections made with respect to an EC. However, the term with base d (degree of input polynomials) was unchanged.

This term is doubly exponential due to the iterated resultant calculations during projection: the resultant of two degree d polynomials is the determinant of a $2d \times 2d$ matrix whose entries all have degree at most d, and thus a polynomial of degree at most $2d^2$. This increase in degree compounded by $(n-1)$ projections gives the first term of the bound (1). When building CAD in the presence of ECs many of these iterated resultants are avoided (thus reducing the *number* of polynomials, but not their degree). Indeed, the derivation of ECs via propagation is itself an iterated resultant calculation.

The purpose of the resultant in CAD construction is to ensure that the points in lower dimensional space where polynomials vanish together are identified, and thus that the behaviour over a sample point in a lower dimensional cell is indicative of the behaviour over the cell as a whole. Busé and Mourrain discuss in [13] the results of applying the iterative univariate resultant to multivariate polynomials, demonstrating decompositions into irreducible factors involving the multivariate resultants (following the formalisation of Jouanolou [31]). They show that the approach will identify polynomials of higher degree than the true multivariate resultant and thus more than required for the purpose of identifying implicit equational constraints. For example, given 3 polynomials in 3 variables of degree d the true multivariate resultant has degree $\mathcal{O}(d^3)$ rather than $\mathcal{O}(d^4)$.

The key result of [13] for our purposes follows. Note that this, using the formalisation of resultants in [31] [13, §2], considers polynomials of a given *total degree*. However, the CAD complexity analysis discussed above and later is (following previous work on the topic) with regards to polynomials of *degree at most d* in a given variable. For clarity we use the Fraktur font when discussing total degree and Roman fonts when the maximum degree.

Corollary ([13, Corollary 3.4]). Given three polynomials $f_k(\mathbf{x}, y, z)$ of the form

$$f_k(\mathbf{x}, y, z) = \sum_{|\alpha|+i+j \leq \mathfrak{d}_k} a^{(k)}_{\alpha,i,j} \mathbf{x}^\alpha y^i z^j \in S[\mathbf{x}][y, z],$$

where S is any commutative ring, then the iterated univariate resultant

$$\mathrm{Res}_y\left(\mathrm{Res}_z(f_1, f_2), \mathrm{Res}_z(f_1, f_3)\right) \in S[\mathbf{x}]$$

is of total degree at most $\mathfrak{d}_1^2\mathfrak{d}_2\mathfrak{d}_3$ in $\mathbf{x}$, and we may express it in multivariate resultants (following the formalism of Jouanolou [31]) as

$$\begin{aligned}\mathrm{Res}_y\left(\mathrm{Res}_z(f_1, f_2), \mathrm{Res}_z(f_1, f_3)\right) = (-1)^{\mathfrak{d}_1\mathfrak{d}_2\mathfrak{d}_3}\, \mathrm{Res}_{y,z}(f_1, f_2, f_3) \\ \times \mathrm{Res}_{y,z,z'}\left(f_1(\mathbf{x}, y, z), f_2(\mathbf{x}, y, z), f_3(\mathbf{x}, y, z')\delta_{z,z'}(f_1)\right).\end{aligned} \tag{8}$$

Moreover, if the polynomials f_1, f_2, f_3 are sufficiently generic and $n > 1$, then this iterated resultant has exactly total degree $\mathfrak{d}_1^2\mathfrak{d}_2\mathfrak{d}_3$ in $\mathbf{x}$ and both resultants on the right hand side of the above equality are distinct and irreducible.
[Although not stated as part of the result in in [13], under these genericity assumptions, $\mathrm{Res}_{y,z}(f_1, f_2, f_3)$ has total degree $\mathfrak{d}_1\mathfrak{d}_2\mathfrak{d}_3$ and the second resultant on the right hand side of (8) has total degree $\mathfrak{d}_1(\mathfrak{d}_1 - 1)\mathfrak{d}_2\mathfrak{d}_3$.]

In [13] the authors interpret this result as follows. "The resultant $R_{12} := \mathrm{Res}_z(f_1, f_2)$ defines the projection of the intersection curve between the two surfaces $\{f_1 = 0\}$ and $\{f_2 = 0\}$. Similarly, $R_{13} := \mathrm{Res}_z(f_1, f_3)$ defines the projection of the intersection curve between the two surfaces $\{f_1 = 0\}$ and $\{f_3 = 0\}$. Then the roots of $\mathrm{Res}_y(R_{12}, R_{13})$ can be decomposed into two distinct sets: the set of roots x_0 such that there exists y_0 and z_0 such that

$$f_1(x_0, y_0, z_0) = f_2(x_0, y_0, z_0) = f_3(x_0, y_0, z_0),$$

and the set of roots x_1 such that there exist two distinct points (x_1, y_1, z_1) and (x_1, y_1', z_1') such that

$$f_1(x_1, y_1, z_1) = f_2(x_1, y_1, z_1) \qquad \text{and} \qquad f_1(x_1, y_1', z_1') = f_3(x_1, y_1', z_1').$$

The first set gives rise to the term $\mathrm{Res}_{x,y,z}(f_1, f_2, f_3)$ in the factorization of the iterated resultant $\mathrm{Res}_y(\mathrm{Res}_{12}, \mathrm{Res}_{13})$, and the second set of roots corresponds to the second factor." Only the first set are of interest to us *if* the f_i are all ECs. However, for a general CAD construction, the second set of roots may also be necessary as they indicate points where the geometry of the sectors changes.

3.2 How large are these resultants

Suppose we are considering three ECs defined by f_1, f_2 and f_3; that we wish to eliminate two variables $z = x_n$ and $y = x_{n-1}$; and that the f_i have degree at most d in each variable *separately*. Then we may naïvely set each $\mathfrak{d}_i = nd$ to bound the total degree.

The following approach does better. Let $K = S[x_1, \ldots, x_{n-2}, y, z]$ and $L = S[\xi_1, \ldots, \xi_N, y, z]$. Only a finite number of monomials in $x_1, \ldots, x_{n-2}$ occur as coefficients of the powers of y, z in f_1, f_2 and f_3. Map each such monomial $x^\alpha = \prod_{i=1}^{n-2} x_i^{\alpha_i}$ to $\widetilde{m_j} := \xi_j^{\max \alpha_i}$ (using a different ξ_j for each monomial[3]) and let $\widetilde{f_i} \in L$ be the result of applying this map to the monomials in f_i. Note that the operation $\widetilde{\ }$ commutes with taking resultants in y and z (though not in the x_i of course).

The total degree in the ξ_j of $\widetilde{f_i}$ is the same as the maximum degree in all the $x_1, \ldots, x_{n-2}$ of f_i, i.e. bounded by d, and hence the total degree of the $\widetilde{f_i}$ in all variables is bounded by $3d$ (d for the ξ_i, d for y and d for z). If we apply (8) to the $\widetilde{f_i}$, we see that

$$\mathrm{Res}_y\left(\mathrm{Res}_z(\widetilde{f_1}, \widetilde{f_2}), \mathrm{Res}_z(\widetilde{f_1}, \widetilde{f_3})\right)$$

has a factor $\mathrm{Res}_{y,z}(\widetilde{f_1}, \widetilde{f_2}, \widetilde{f_3})$ of total degree (in the ξ_j) $(3d)^3$. Hence, by inverting $\widetilde{\ }$, we may conclude $\mathrm{Res}_{y,z}(f_1, f_2, f_3)$ has maximum degree, in each x_i, of $(3d)^3$.

The results of [31] [13] apply to any number of eliminations. In particular, if we have eliminated not 2 but $\ell - 1$ variables we will have a polynomial $\mathrm{Res}_{x_{n-\ell+1} \cdots x_n}(f_{n-\ell}, \ldots, f_n)$ of maximum degree $\ell^\ell d^\ell$ in the remaining variables $x_1, \ldots, x_{n-\ell}$ as the last implicit EC.

These resultants $\mathrm{Res}_{x_{n-\ell+1} \cdots x_n}$ therefore only have singly-exponential growth, rather than the doubly-exponential growth of the iterated resultants: can we compute them?

3.3 Gröbner bases in place of iterated resultants

A *Gröbner Basis* G is a particular generating set of an ideal I defined with respect to a monomial ordering. One definition is that the ideal generated by the leading terms of I is generated by the leading terms of G. Gröbner Bases (GB) allow properties of the ideal to be deduced such as dimension and number of zeros and so are one of the main practical tools for working with polynomial systems. Their properties and an algorithm to derive a GB for any ideal were introduced by Buchberger in his PhD thesis of 1965 [11]. There has been much research to improve and optimise GB calculation, with the F_5 algorithm [23] perhaps the most used approach currently.

Like CAD the calculation of GB is necessarily doubly exponential in the worst case [32] (when using a lexicographic order), although recent work in [33]

[3] It would be possible to economise: if $x_1 x_2^2 \mapsto \xi_1^2$, then we could map $x_1^2 x_2^4$ to ξ_1^4 rather than a new ξ_2^4. Since this trick is used purely for the analysis and not in implementation, we ignore such possibilities.

showed that rather than being doubly exponential with respect to the number of variables present the dependency is in fact on the dimension of the ideal. Despite this worst case bound GB computation can often be done very quickly usually to the point of instantaneous for any problem tractable by CAD.

A reasonably common CAD technique is to precondition systems with multiple ECs by replacing the ECs by their GB. I.e. let $E = \{e_1, e_2, \dots\}$ be a set of polynomials; $G = \{g_1, g_2, \dots\}$ a GB for E; and B any Boolean combination of constraints, $f_i \, \sigma_i \, 0$, where $\sigma_i \in \{<, >, \leq, \geq, \neq, =\}$) and $F = \{f_1, f_2, \dots\}$ is another set of polynomials. Then

$$\begin{aligned} \Phi &= (e_1 = 0 \land e_2 = 0 \land \dots) \land B \\ \Psi &= (g_1 = 0 \land g_2 = 0 \land \dots) \land B \end{aligned}$$

are equivalent and a CAD truth-invariant for either could be used to solve problems involving Φ.

As discussed, the cost of computing the GB itself is minimal so the question is whether it is beneficial to CAD. The first attempt to answer this question was given by Buchberger and Hong in 1991 [12] (using GB and CAD implementations in the SAC-2 system [15]). These experiments were carried out before the development of reduced projection operators and so the CADs computed were sign-invariant (and thus also truth-invariant for the formulae involved). Of the 10 problems studied: 6 were improved by the GB preconditioning, (speed-up from 2-fold to 1700-fold); 1 problem resulted in a 10-fold slow-down; 1 timed out after GB but completed without; and the other 2 were intractable both for CAD and GB. The problem was recently revisited by Wilson et al. [43] who studied the same problem set using Qepcad-B for the CAD and Maple 16 for the GB. There had been a huge improvement to the GB computation but it was still the case that two of the problems were hindered by GB preconditioning. A recent machine learning experiment to decide when GB precondition should be applied [28] found that 75% of a data set of 1200 randomly generated CAD problems benefited from GB preconditioning.

If we consider GB preconditioning of CAD in the knowledge of the improved projection schemes for ECs (Subsection 2.4) then we see an additional benefit from the GB. It provides ECs which are not in the main variable of the system removing the need for iterated resultants to find implicit ECs to use in subsequent projections.

Since our aim is to produce one EC in each of the last ℓ variables, we need to choose an ordering on monomials which is lexicographic with respect to $x_n \succ x_{n-1} \succ \cdots \succ x_{n-\ell+1}$: it does not actually matter (from the point of view of the theory: general theory suggests that 'total degree reverse lexicographic in the rest' would be most efficient in practice) how we tie-break after that.

Let us suppose (in line with [21]) that we have ℓ ECs $f_1, \dots, f_\ell$ (at least one of then, say f_1 must include x_n, and similarly we can assume f_2 includes x_{n-1} and so on), such that these imply (even over $\mathbb{C}$) that the last ℓ variables are determined (not necessarily uniquely) by the values of $x_1, \dots, x_{n-\ell}$. Then the polynomials f_1, $\mathrm{Res}_{x_n}(f_1, f_2)$, $\mathrm{Res}_{x_n, x_{n-1}}(f_1, f_2, f_3)$ etc. are all implied by the f_i. Hence either they are in the GB, or they are reduced to 0 by the GB, which implies that smaller polynomials are in the GB. Hence our GB will contain polynomials (which are ECs) of degree (in each variable separately) at most

$$d,\ 4d^2,\ 27d^3,\ \ldots,\ ((\ell+1)d)^{\ell+1}.$$

Note that we are not making, and in the light of [33] cannot make, any similar claim about the polynomials in fewer variables. Note also that it is vital that the equations be in the last variables for this use of [31,13] to work.

4 Worked Example

We will work with the polynomials

$$f_1 := xy - z^2 - w^2, \qquad f_2 := x + y^2 + z + w,$$
$$f_3 := x - y^2 + z - w, \qquad h := x + y + z + w,$$

and the semi-algebraic system

$$\phi := f_1 = 0 \wedge f_2 = 0 \wedge f_3 = 0 \wedge h > 0.$$

We assume a variable ordering $z \succ y \succ x \succ w$ (meaning we will first project with respect to z) and seek a CAD truth-invariant for ϕ.

We have 3 ECs to take advantage of. However, they all have main variable z and so only one of them may be a designated EC for projection purposes. The existing theory [36], [21] would suggest propagating the ECs by calculating:

$$\begin{aligned}
r_1 &:= \operatorname{res}(f_1, f_2, z) = -y^4 - 2wy^2 - 2xy^2 - 2w^2 - 2wx - x^2 + xy,\\
r_2 &:= \operatorname{res}(f_1, f_3, z) = -y^4 - 2wy^2 + 2xy^2 - 2w^2 + 2wx - x^2 + xy,\\
r_3 &:= \operatorname{res}(f_2, f_3, z) = -2y^2 - 2w;
\end{aligned}$$

and then

$$\begin{aligned}
R_1 &:= \operatorname{res}(r_1, r_2, y) = 256x^4(w^4 + 2w^2x^2 + x^4 + wx^2),\\
R_2 &:= \operatorname{res}(r_1, r_3, y) = \operatorname{res}(r_2, r_3, y)\\
&= 16w^4 + 32w^2x^2 + 16x^4 + 16wx^2.
\end{aligned}$$

Of course, since R_1 contains R_2 as a factor we have $\operatorname{res}(R_1, R_2, x) = 0$.

We have multiple choices for running Algorithm 1 since we can only declare one polynomial as an EC with a set main variable. There are hence $3 \times 3 \times 2 = 18$ possible configurations. We have built the CAD for each choice (lifting using the improved procedure developed in [21]). Depending on the choice made the number of cells in the final CAD varies between 73 and 335 (with average value 185). The minimum is achieved using designations (f_2, r_3, R_2).

Now consider taking a GB of $\{f_1, f_2, f_3\}$. We use a plex monomial ordering on the same variable ordering as the CAD to achieve a basis defined by

$$g_1 = z + x, \qquad g_2 = y^2 + w, \qquad g_3 = -w^2 - x^2 + xy$$
$$g_4 = w^2x + w^2y + x^3 + wx, \qquad g_5 = x^4 + 2x^2w^2 + w^4 + x^2w$$

This is an alternative generating set for the ideal defined by the explicit ECs and thus all the $g_i = 0$ are also ECs for ϕ. We consider using these as the designated ECs when building the CAD. There is no longer any choice regarding the EC with mvar z or x but there are 3 possibilities for the designation with mvar y. Designating g_2 yields 73 cells (the best previous cell count) while either of g_3 or g_4 yields 45 cells. Note in particular that the degrees of the GB polynomials are on average lower (and never greater) than those of the iterated resultants.

5 Sketch of the effect on complexity

Following Section 3 we see that when building a lexicographical basis the degree of the polynomials in the GB is restricted and thus this will be a better method for the identification of implicit ECs to use in subsequent projections than iterated resultants. Let us sketch how this will effect the complexity of CAD following the techniques set out in [5], [21] and summarised in Section 2.4.

The designated ECs will have lower degrees $d, 4d^2, 27d^3$ and in general $(sd)^s$ for the EC with mvar x_{n-s}. From now on we will ignore the constant factors and focus on the exponents of d generated. This is both for simplicity in the analysis, and because we have not found a closed form solution for the product of the constant factors in the new analysis. But we note that when using GB the constant factors grow exponentially in ℓ while with iterated resultants they grow doubly exponentially in ℓ (as in Table 1). Further, the constant term can be shown to be strictly lower for all but the first few projections, with the issue there a laxness of the analysis not the algorithm (as in Section 3.1 we saw that the multivariate resultants was itself a factor of the iterated resultant).

We keep track of both the degree of the designated EC and the degree of the entire set of polynomials. The reduced projection operator $P_F(B)$ will still take discriminants and coefficients of these; and resultants of them with the other projection polynomials. Thus the highest degree polynomial produced grows with the exponent of d being the sum of the exponent from the designated EC and that from the other polynomials. This generates the top half of Table 2 and we see that the exponents form the so called *Lazy Caterer's sequence*[4] otherwise known as the Central Polygonal Numbers. The remaining projections use the sign-invariant projection operator and so the degree is squared each time, leading to the bottom half of Table 2.

[4] The On-Line Encyclopedia of Integer Sequences, 2010, Sequence A000124, https://oeis.org/A000124

Table 2. Maximum degree of projection polynomials produced for CAD when using projection operator (5) ℓ times and then (3).

Variables	**Maximum Degree**	
	EC	**Others**
n	d	d
$n-1$	d^2	d^2
$n-2$	d^3	d^4
$n-3$	d^4	d^7
$\vdots$	$\vdots$	$\vdots$
$n-\ell$	$d^{\ell+1}$	$d^{\ell(\ell+1)(1/2)+1}$
$n-(\ell+1)$	$d^{\ell(\ell+1)+2}$	
$n-(\ell+2)$	$d^{2\ell(\ell+1)+2^2}$	
$n-(\ell+3)$	$d^{2^2\ell(\ell+1)+2^3}$	
$\vdots$	$\vdots$	
$n-(\ell+r)$	$d^{2^{r-1}\ell(\ell+1)+2^r}$	
$\vdots$	$\vdots$	
1	$d^{2^{n-\ell-2}\ell(\ell+1)+2^{n-\ell-1}}$	

We can now consider the generic bound (7) using the degrees from Table 2 as the d_i. The term with base d may be computed by

$$\prod_{s=0}^{\ell}\left(d^{s+1}\right)\prod_{r=1}^{n-\ell-1}\left(d^{2^{r-1}\ell(\ell+1)+2^r}\right).$$

The exponent of d evaluates to

$$2^{(n-\ell)}\tfrac{1}{2}(\ell^2+\ell+2)-\tfrac{1}{2}(\ell^2+\ell)-2. \tag{9}$$

Let us compare this with the term with base m from (2). As with the improvements in [21], the improvements here have allowed the reduction of the double exponent from by ℓ, the number of ECs used. However, the reduction is not quite as clean as the exponential term in the single exponent is multiplied by a quadratic in ℓ. This is to be expected as the singly exponential dependency on ℓ in the Number column of Table 1 was only in the term with constant base while for Table 2 the term with base d is itself single exponential in ℓ.

6 Discussion

We have considered the issue of CAD in the presence of multiple ECs. We followed our recent work in [21] which reduced the complexity with respect to the number of polynomials to see if similar achievements can be obtained with the respect to polynomial degree. We suggest that using Gröbner Bases in place of iterated resultants allows for the same reduction in the double exponent by the number of ECs used during projection.

We have sketched the complexity results but defer the full analysis (using the (M,D)-property as detailed in [5]) until a number of issues can be cleared up. We discuss some of them here:

– Will using GB not risk increasing the base number of polynomials in m?

On one level this seems unlikely (since we are starting with a generating set all in the main variable and deriving another which would mostly not be) but we have yet to rule it out. Of course, the number of polynomials in the input can bear little relation to the number generated by projection.

We note that there is an alternative way to use GB for CAD than that outlined in Section 3.3 (replacing a set of ECs by another). We could instead use the GB purely as an implicit EC generation tool; and just add selected polynomials from it to our input without replacing anything. For example, the GB in the worked example of Section 4 had 3 polynomials with main variable y only one of which can be the designated EC. Rather than replacing all the f_i by all the g_i we could instead just add 2 of the g_i (one in main variable y and one in x) to the input set to act as designated ECs at lower levels. This approach would cap the increase in m to the number of designated ECs we can identify. Taking this approach with the example in Section 4 would not change the final cell counts.

– Will the GB always produce as many ECs with different main variables as the iterated resultant method?
– How to proceed in the case where we have non-primitive ECs?

This paper (as with most previous work on ECs) only deals with the case of primitive designated ECs. We refer the reader to the final section of [21] where we sketch approaches that could be adapted to deal with this (including the theory of TTICAD from [4] [5]).

– How is the complexity affected when the projections using ECs are not in strict succession?
– Can we mix the orderings in the CAD and the GB?

Finally, we return to the fact acknowledged in Section 3.3 that previous work on using GB to precondition CAD [12], [43], [28] has found that it is not always beneficial and how that interacts with the claims of this paper. The simple answer is that the analysis offered here is of the worst case and makes no claim to the average complexity. However, we actually hypothesise that it was it was the fact that the CAD computations involved in those paper did not take advantage of the new multiple EC technology which will account for many of the cases were GB hindered CAD. We plan future experiments to test this hypothesis.

Acknowledgements This work was originally supported by EPSRC grant: EP/J003247/1 and is now supported by EU H2020-FETOPEN-2016-2017-CSA project $\mathcal{SC}^2$ (712689). We are also grateful to Professor Buchberger for reminding JHD that Gröbner bases were applicable here.

References

1. D. Arnon, G.E. Collins, and S. McCallum. Cylindrical algebraic decomposition I: The basic algorithm. *SIAM Journal of Computing*, 13:865–877, 1984.
2. S. Basu, R. Pollack, and M.F. Roy. *Algorithms in Real Algebraic Geometry.* Volume 10 of Algorithms and Computations in Mathematics. Springer-Verlag, 2006.
3. 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*, volume 8660 of *Lecture Notes in Computer Science*, pages 44–58. Springer International Publishing, 2014.
4. 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.
5. 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.
6. 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*, volume 7961 of *Lecture Notes in Computer Science*, pages 19–34. Springer Berlin Heidelberg, 2013.
7. C.W. Brown. Improved projection for cylindrical algebraic decomposition. *Journal of Symbolic Computation*, 32(5):447–465, 2001.
8. 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.
9. 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.
10. 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.
11. B. Buchberger. Bruno Buchbergers 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.

12. B. Buchberger and H. Hong. Speeding up quantifier elimination by Gröbner bases. Technical report, 91-06. RISC, Johannes Kepler University, 1991.
13. L. Busé and B. Mourrain. Explicit factors of some iterated resultants and discriminants. *Mathematics of Computation*, 78:345–386, 2009.
14. 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.
15. G.E. Collins. The SAC-2 computer algebra system. In B.F. Caviness, editor, *EUROCAL '85*, volume 204 of *Lecture Notes in Computer Science*, pages 34–35. Springer Berlin Heidelberg, 1985.
16. G.E. Collins. Quantifier elimination by cylindrical algebraic decomposition – 20 years of progress. In B. Caviness and J. Johnson, editors, *Quantifier Elimination and Cylindrical Algebraic Decomposition*, Texts & Monographs in Symbolic Computation, pages 8–23. Springer-Verlag, 1998.
17. G.E. Collins and H. Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *Journal of Symbolic Computation*, 12:299–328, 1991.
18. 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.
19. J.H. Davenport and J. Heintz. Real quantifier elimination is doubly exponential. *Journal of Symbolic Computation*, 5(1-2):29–35, 1988.
20. 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*, volume 8543 of *Lecture Notes in Artificial Intelligence*, pages 45–60. Springer International, 2014.
21. 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.
22. 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.
23. J.C. Faugère. A new efficient algorithm for computing groebner bases without reduction to zero (F5). In *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*, ISSAC '02, pages 75–83. ACM, 2002.
24. 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.
25. J. Han, L. Dai, and B. Xia. Constructing fewer open cells by gcd computation in CAD projection. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, ISSAC '14, pages 240–247. ACM, 2014.
26. J. Heintz. Definability and fast quantifier elimination in algebraically closed fields. *Theoretical Computer Science*, 24(3):239–277", 1983.
27. H. Hong. An improvement of the projection operator in cylindrical algebraic decomposition. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, ISSAC '90, pages 261–264. ACM, 1990.

28. Z. Huang, M. England, J.H. Davenport, and L. Paulson. Using machine learning to decide when to precondition cylindrical algebraic decomposition with Groebner bases. Under Preparation, 2016.
29. 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*, volume 8543 of *Lecture Notes in Artificial Intelligence*, pages 92–107. Springer International, 2014.
30. 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.
31. J.P Jouanolou. Le formalisme du résultant. *Advances in Mathematics*, 90(2):117–263, 1991.
32. 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.
33. E.W. Mayr and S. Ritscher. Dimension-dependent bounds for gröbner bases of polynomial ideals. *Journal of Symbolic Computation*, 49:78–94, 2013.
34. 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.
35. 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.
36. 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.
37. L.C. Paulson. Metitarski: Past and future. In L. Beringer and A. Felty, editors, *Interactive Theorem Proving*, volume 7406 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2012.
38. J.T. Schwartz and M. Sharir. On the "Piano-Movers" Problem: I. The case of a two-dimensional rigid polygonal body moving amidst polygonal barriers. *Communications on Pure and Applied Mathematics*, 36(3):345–398, 1983.
39. A. Strzeboński. Cylindrical algebraic decomposition using validated numerics. *Journal of Symbolic Computation*, 41(9):1021–1038, 2006.
40. 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.
41. D. Wilson, R. Bradford, J.H. Davenport, and M. England. Cylindrical algebraic sub-decompositions. *Mathematics in Computer Science*, 8:263–288, 2014.
42. 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.
43. 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*, volume 7362 of *Lecture Notes in Computer Science*, pages 280–294. Springer, 2012.