# Using the distribution of cells by dimension in a cylindrical algebraic decomposition

David Wilson, Matthew England, Russell Bradford & James H. Davenport
Department of Computer Science, University of Bath, Bath, BA2 7AY, UK
E-mail: {D.J.Wilson, M.England, R.J.Bradford, J.H.Davenport}@bath.ac.uk

***Abstract*—We investigate the distribution of cells by dimension in cylindrical algebraic decompositions (CADs). We find that they follow a standard distribution which seems largely independent of the underlying problem or CAD algorithm used. Rather, the distribution is inherent to the cylindrical structure and determined mostly by the number of variables.**

**This insight is then combined with an algorithm that produces only full-dimensional cells to give an accurate method of predicting the number of cells in a complete CAD. Since constructing only full-dimensional cells is relatively inexpensive (involving no costly algebraic number calculations) this leads to heuristics for helping with various questions of problem formulation for CAD, such as choosing an optimal variable ordering. Our experiments demonstrate this can be highly effective.**

## I. Introduction

### A. Background on CAD

A **cylindrical algebraic decomposition** (CAD) is:

- a **decomposition** of $\mathbb{R}^n$, meaning a collection of cells which do not intersect and whose union is $\mathbb{R}^n$;
- **cylindrical**, meaning the projections of any pair of cells with respect to a given variable ordering are either equal or disjoint;
- **(semi)-algebraic**, meaning each cell can be described using a finite sequence of polynomial relations.

The first algorithm to produce CADs was introduced by Collins [1]. Here we start with a projection phase to derive a set of polynomials relative to the input, then CADs are built incrementally by dimension according to the zeros of those polynomials (a process known as lifting).

Each cell is represented by a **cell-index**: an $n$-tuple of integers defining its position in the CAD. An even integer is referring to a variable taking the value of one of the (ordered) real roots of the projection polynomials and an odd integer means that a variable is within an interval between two of these. In addition each cell would usually be accompanied by a sample point used in the construction.

We would normally compute a CAD to solve an underlying problem. Most notably, it can be a tool for quantifier elimination (QE) over the reals. Here we must build a CAD relative to a quantified formula such that the Boolean value of that formula is invariant (true or false) in each cell. Then an equivalent quantifier-free formula may be computed from the semi-algebraic descriptions of the true cells. CAD has been applied elsewhere, including problems in parametric optimisation [17], epidemic modelling [7], theorem proving [23], motion planning [27] and reasoning with multi-valued functions and their branch cuts [13].

The original, and most common CAD algorithm gives output that is **sign-invariant** with respect to a set of polynomials. This means that each polynomial in the input has constant sign on each cell of the CAD created. However, a CAD can be produced more efficiently if we work closer to the underlying problem, for example, by building CADs invariant with respect to the truth of formulae [4], or making use of the structure of the quantifiers [11]. Other important advances in CAD theory include the use of certified numerics [25], [19] and an alternative approach to Collins' algorithm using the theory of regular chains and triangular decomposition [9], [2].

In this paper we are more concerned with a CAD as the mathematical object satisfying the definition above (rather than the particular algorithm which produces one).

### B. Contribution

Each cell in a CAD has a **dimension** which can range from 0 (when the cell is a point) to $n$ (when the cell is of full-dimension in $\mathbb{R}^n$). We describe a subset of cells from a CAD as a **sub-CAD**. The sub-CAD consisting of only those cells with full-dimension has been a much studied topic [20], [24]. It can be identified far more efficiently than the complete CAD and is sufficient to solve certain classes of problems. More generally, we define those cells in a CAD with the same dimension as a **layer** and an **$\ell$-layered sub-CAD** as a sub-CAD of $\mathbb{R}^n$ consisting of those cells with dimensions $d \in [n-\ell+1, n]$ (i.e. the top $\ell$ layers of the CAD). We call the CAD consisting of all $n+1$ layers the **complete CAD**. See Section 2.2 of [26] for details including algorithms to produce such sub-CADs both directly and recursively (where additional layers are created one at a time).

In Section II we investigate the spread of cell dimensions in a CAD. We discover that they conform to a common distribution regardless of the problem studied or algorithm used. Rather the distribution is a feature of the cylindrical structure and determined mostly by the number of variables. This means that the size (number of cells) of a CAD may be predicted accurately by the number of full-dimensional cells (which can be computed far quicker). In Section III we investigate using this as a heuristic for deciding questions of problem formulation for CAD, showing promising experimental results. We now continue the introduction with a simple example to illustrate the ideas so far.

### C. Motivating Example

*Example 1:* Consider

$$f = x - y^2, \qquad g = x^2 - y^2 - 1$$

which are graphed respectively by the circle and parabola in the first image of Figure 1.

A CAD is defined implicitly with respect to a variable ordering (defining the projections used for cylindricity). Let us assume first an ordering $y \succ x$ meaning projections are from $(x, y) \to x$. The second image of Figure 1 visualises a sign-invariant CAD for $\{f, g\}$ in this ordering by marking each cell with a black box. If a box lies at the intersection of the curves with each other or the dotted lines (which include the $y$-axis) then they indicate a cell of dimension 0: just that point. Otherwise, if the box lies on one of the curves or dotted lines then it indicates a cell of dimension 1: that line segment. The remaining boxes indicates cells of full dimension: portions of $\mathbb{R}^2$ bounded by the curves or dotted lines.

In fact this is the minimal sign-invariant CAD for $\{f, g\}$ in the ordering, that is, the one with the fewest number of cells which satisfies the definitions. For example, consider $x \in (-1, 0)$. Then the sign-invariant condition means we must distinguish the five cells indicated (the two portions of the circle and the spaces between, above and below). We could not extend these cells beyond $(-1, 0)$ without violating the cylindricity or sign-invariance conditions. Similar arguments show that the 51 cells indicated are indeed the minimum.

The third image in Figure 1 shows only the full dimensional cells (as portions of $\mathbb{R}^2$ coloured differently to neighbouring cells). There are 17 of these. The question answered affirmatively in this paper is whether the complexity of the second image (number of black squares) can be predicted accurately by the complexity of the third image (number of different coloured portions). The experiments in Section II suggest that for a CAD in two variables approximately 0.334 of the cells are full dimensional. Hence knowing the number of full dimensional cells only, we would correctly predict there to be $17/0.334 = 50.898 \simeq 51$ cells in the complete CAD.

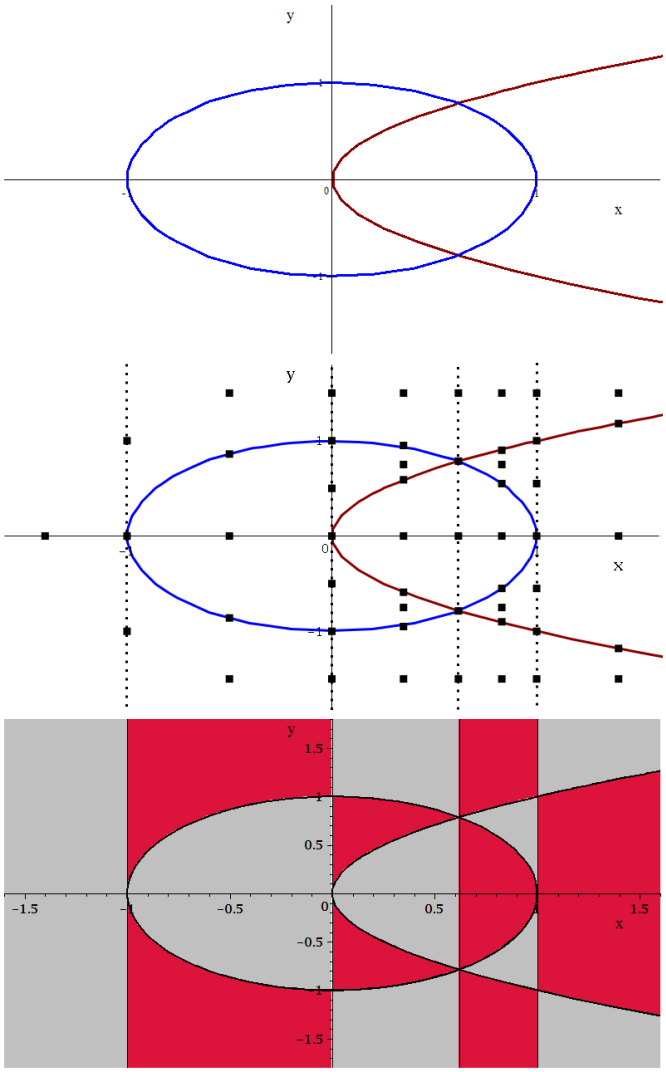
**Remarks:** We note the following about this example:

1) Although the full dimensional cells may be sufficient to solve certain problems others will require knowledge of the complete CAD. For example, the formula $f = 0 \wedge g < 0$ is not true on any cell of full dimension.
2) The images above relate to a hypothetical minimal CAD, not necessarily one produced by a known algorithm. CAD algorithms identify points of intersection by taking resultants. In this case we have

$$\text{res}_y(f, g) = (x^2 + x - 1)^2$$

which has a pair of roots at $x = \frac{1}{2}(-1 \pm \sqrt{5})$. The root at $0.618$ identifies the real intersections of $f$ and $g$ while the other at $-1.618$ identifies intersections in $\mathbb{C}^2$ (at points with complex $y$ coordinate).
Hence, while not required for a sign-invariant CAD, known CAD algorithms would split the leftmost cell into three (the line $x = -1.618$ and two full dimensional cells either side).

Fig. 1: From top to bottom: graphs of $f = x - y^2$ and $g = x^2 + y^2 - 1$; a sign-invariant CAD for $\{f, g\}$, the full dimensional cells in the CAD. The latter two use variable order $y \succ x$, meaning projections are from $(x, y) \to x$.

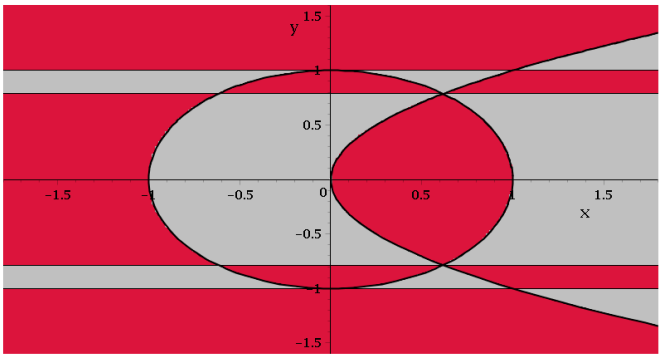



In this case, the number of full dimensional cells increases to 18 and the predicted number of total cells becomes $18/0.334 = 53.892 \simeq 54$, one more than the total.

Constructing CADs for the example above is simple with modern technology. However, for larger problems (particularly those with more variables) CAD can be challenging. Although the cost of computing the full-dimensional cells also increases with the size of the problem, it is much easier as it avoids any computation with algebraic numbers. Hence, in many situations it is feasible to use the full dimensional cells as a metric to predict the size of the complete CAD, or the feasibility of computing it.

This can be useful for deciding questions of problem formulation for CAD. For example, there may be a free or constrained choice over the variable ordering. When using CAD for QE variables must be projected in the order they are

Fig. 2: The full dimensional cells in a CAD for $f = x - y^2$ and $g = x^2 + y^2 - 1$, with variable ordering $x \succ y$.




quantified but we can change the ordering of free variables, or variables in blocks of the same quantifier. The minimal sign-invariant CAD for the example above with ordering $x \succ y$ has 47 cells. The 16 full dimensional cells are shown in Figure 2. If we calculated just these we would predict $16/0.334 = 47.905 \simeq 48$ cells in the complete CAD. Hence using the number of full-dimensional cells as a heuristic would lead us to use the variable ordering producing the smaller complete CAD.

There do exist other heuristics for making such choices, which can be much cheaper, although they may not be as closely correlated. See [15] for a recent summary. An important example is Brown's heuristic [6] which uses only simple measures on the input to decide an ordering. Recent studies show that while it usually gives a good choice, there are classes of problems where it is not successful [18]. For this example the measures used by this heuristic do not discriminate between $y \succ x$ and $x \succ y$.

Another useful heuristic is `sotd` which is tailored to a specific CAD algorithm, measuring the total degrees of every monomial in every projection polynomial produced. For this example, `sotd` is actually misled to pick $y \succ x$ (essentially because the resultant of the polynomials in $y$ factors so that as a projection polynomials it has lower `sotd`, despite the same number of real roots as the resultant in $x$). A heuristic developed in [5] to count the size of the induced decomposition of the real line would identify the optimal ordering, but only due to the extra root at $-1.618$ being identified despite not being required for the minimal sign-invariant CAD in $y \succ x$.

## II. Distribution of Cells by Dimension

### A. Distribution for existing problems

We studied a set problems from the CAD Example Bank [28], sourced in turn from the papers [8] and [9]. For each problem a sign-invariant CAD was calculated using an implementation of [21] in Maple as detailed in [16]. Then for each problem the distribution of cells by dimensions was plotted, as displayed in Figure 3. We see that all examples with the same number of variables seem to share a similar distribution of cell dimensions: a roughly normal distribution biased towards cells of large dimension. The closest standard distribution would be a binomial with a $p$ value greater than $0.5$.

Fig. 3: CAD cell dimension distribution for examples from [28]. The lines are coloured according to the number of variables in the problem (from 2 to 6 going from left to right).

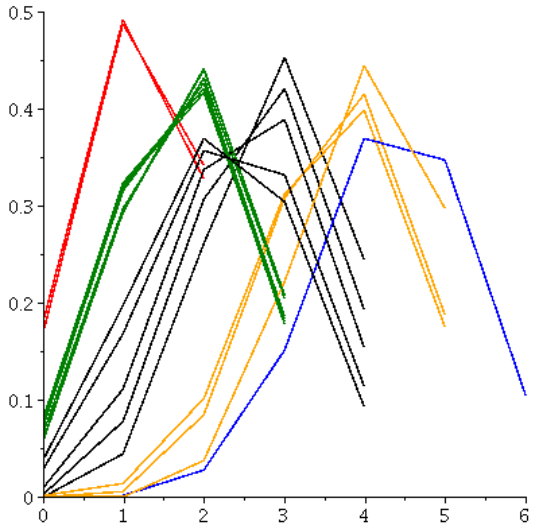
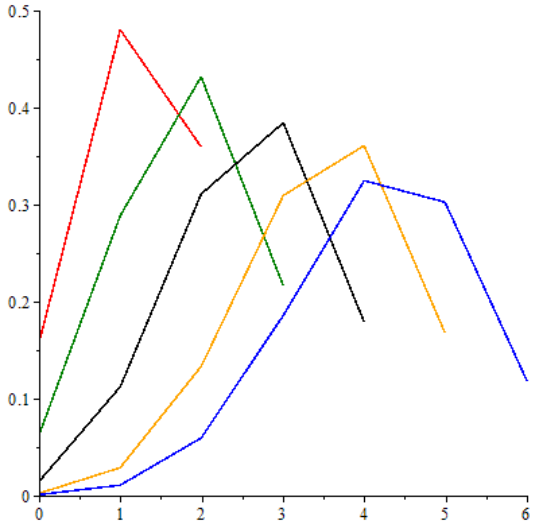



Fig. 4: Binomial distributions which match the distribution of CAD cells in Figure 3. Calculated from left to right with $(n, p) = [(2, 0.6), (3, 0.6), (4, 0.65), (5, 0.7), (6, 0.7)]$.




Recall that the **binomial distribution** for $n$ trials with probability $p$ of success is given by:

$$\mathbb{P}(X = x) := \begin{cases} \binom{n}{x} p^x (1-p)^{n-x} & 0 \le x \le n, \\ 0 & \text{otherwise.} \end{cases}$$

We determined by eye the $p$-values that best match the examples from [28] and plot these in Figure 4. We find that as $n$ increases, the most suitable value of $p$ increases.

These experiments demonstrate that the distribution of CAD cell dimensions is almost independent of the polynomials the CAD is computed with respect to, instead being determined mostly by the number of variables present.

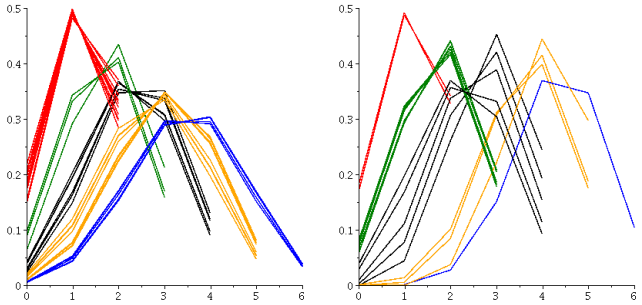
### B. Distribution for random cylindrical decompositions

Our experiments so far suggest the distribution of cell dimensions is largely independent of the individual problems. We would like to go further and demonstrate that they are also independent of the CAD algorithm and implementation used.

We define **combinatorially random CADs**. These are not CADs constructed for randomly generated problems: instead they are hypothetical decompositions of real space made randomly, but with a cylindrical structure. First a number of variables is chosen at random and then the real line is decomposed into a random number of points and intervals between (a CAD of $\mathbb{R}^1$). Then each cylinder over a cell is split into a random number of cells: these are assumed sections (zeros of some polynomial) and sectors (the regions in between) but actually here we are constructing just a combinatorial object: a collection of cell indices with no associated projection polynomials. The process is continued until we reach $\mathbb{R}^n$.

We constructed 45 of these objects in MAPLE using the `rand` command to iteratively build the cell indices (from which cell dimension are easily determined). Variables were chosen randomly from $\{2,\dots,6\}$ and cylinders were split using a random number of sections from $\{1,\dots,7\}$. Figure 5a shows the distribution of cell dimensions, and is given alongside the examples from [28] showing a striking similarity between distributions with the same number of variables.

Fig. 5: Comparing the distribution of cell dimensions of combinatorially random CADs with those created for real examples.



(a) Combinatorially random. (b) Examples from [28].

### C. Investigating the combinatorial structure of a CAD

We now provide some formal justification of the binomial distribution observed in examples. In this subsection assume that $\mathcal{D}$ is a CAD or sub-CAD of $\mathbb{R}^n$ and $\mathfrak{D}_i$ the number of cells in $\mathcal{D}$ of dimension $i$ (for $i = 0,\dots,n$). We consider the **induced CADs** of $\mathcal{D}$: the CADs of $\mathbb{R}^j$ ($j = 1,\dots n-1$) formed by projecting $\mathcal{D}$ with respect to the ordering in which it is defined. The induced CAD of $\mathbb{R}^1$ is a decomposition of the real line into $k_1$ points and $k_1+1$ intervals.

We make the simplifying assumption that when considering the cylinders over cells from the same induced CAD of $\mathcal{D}$ they are split into the same number of cells. That is, we assume the cylinder over each cell of $\mathbb{R}^m$ consists of $2k_m+1$ cells in $\mathbb{R}^{m+1}$ ($k_m$ of which are the same dimension as the base cell).

*Lemma 1:* Let $\mathcal{D}$ be as described above. Then

$$\mathfrak{D}_i = \sum_{\substack{P\subseteq[n]\\|P|=i}} \left( \prod_{a\in P}(k_a+1) \prod_{b\in[n]\setminus P} k_b \right),$$

where $[n]$ is the combinatorial shorthand for $\{1,\dots,n\}$.

In particular we have:

$$\mathfrak{D}_0 = \prod_{i=1}^{n} k_i, \qquad \mathfrak{D}_n = \prod_{i=1}^{n}(k_i+1).$$

*Proof:* The dimension of a cell in $\mathcal{D}$ is equal to the sum of the parity of its cell indices. For a cell to have dimension $i$, it must therefore have $i$ odd indices and $n-i$ even indices.

We can characterise an $i$-dimensional cell by the position of its odd indices. Call the set of these positions $P$. For a fixed $P$ there are many cells associated. There are a total $k_1+1$ choices of 1-cells if $1\in P$ and $k_1$ choices of 0-cells if $1\notin P$. Continuing to build the CAD, at level $j$ there are $k_j+1$ cell choices if $j\in P$ and $k_j$ choices if $j\notin P$.

Therefore, for a fixed $P$, the number of cells that have an appropriate cell index is given by the product inside the parenthesis. All that remains is to sum over all possible subsets $P$ of $[n]$ which have cardinality $i$. ■

We can see above that in the case where $k_i = k$ for all $i$, then $\{\mathfrak{D}_i\}$ is simply the sequence of binomial coefficients of $(k+(k+1)x)^n$. In fact, we can see the relationship with the binomial distribution without this extra assumption.

*Lemma 2:* Let $\mathcal{D}$ be as given in Lemma 1. Then the generating function for $\mathfrak{D}_i$ is given by:

$$\prod_{i=1}^{n}(k_i+(k_i+1)x).$$

That is, $\mathfrak{D}_i$ is the coefficient of $x^i$ in the expansion of the above product.

*Proof:* Expanding out the product you obtain $x^i$ precisely by choosing $x$ from $i$ different linear factors. This amounts to selecting $i$ integers from the set $[n]$. For a given $P$, each $j\in P$ contributes $k_j+1$ to the coefficient of the generated monomial, and each $j\notin P$ contributes $k_j$. This means the coefficient of the generated $x^i$ is:

$$\prod_{a\in P}(k_a+1) \prod_{b\in[n]\setminus P} k_b.$$

The coefficient of $x^i$ in the expansion of the product is precisely the summation of all such coefficients:

$$\sum_{\substack{P\subseteq[n]\\|P|=i}} \left( \prod_{a\in P}(k_a+1) \prod_{b\in[n]\setminus P} k_b \right),$$

which from Lemma 1 is precisely $\mathfrak{D}_i$. ■

## III. Heuristics for CAD problem formulation

As suggested in the introduction, the consistent distribution of cell dimensions can be used for predicting the size of the complete CAD. Since constructing these cells is far simpler (avoiding all computation with algebraic numbers) we can use it as the basis for heuristics for questions of problem formulation. To experiment with this we collated the distributions for our example set, split by number of variables. Table I shows the proportion of cells that were full-dimensional for the examples in [28]. We can then make a prediction by comparing the 1-layered CAD for the problem in hand to the average distribution for the number of variables present

TABLE I: Average fraction of cells with full-dimensional in CADs for the examples in [28].

| Variables | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| Fraction | 0.334 | 0.192 | 0.161 | 0.181 |

### A. A new heuristic for choosing a variable ordering

Example 1 in the introduction already demonstrated the idea of a heuristic based on full-dimensional cells for picking a variable ordering. In that example a modest saving was made but more generally the choice of ordering can determine the tractability of a problem. In [3] a class of examples were presented where the ordering changed the complexity from constant to doubly exponential in the number of variables. The following examples show that a heuristic based on full-dimensional cells works for problems in three dimensions also, and the potential costs and benefits of using it.

*Example 2:* We consider the two sets of polynomials,

$$F_1 := \left\{x^2+y^2+z^2-1, xy-yz+3, x+y-yz^4\right\};$$
$$F_2 := \left\{x^2+y-z^3-1, x^2-5y+1, z^3-xy, z^3-5\right\}.$$

These are defined in the three variables $x, y, z$ and so there are six possible variable orderings for a CAD. We assume all are admissible and seek the one which gives the smallest CAD.

For each variable ordering, we create a 1-layered sign-invariant sub-CAD using the algorithm in [26] and a complete sign-invariant CAD using the algorithm in [21] (both implemented in the Maple package `ProjectionCAD` [16]). The number of cells and computation times for these are recorded, along with the prediction of the number of cells in the full-CAD that would be made by considering the number of full-dimensional cells (multiplying the number of cells in the 1-layered sub-CAD cell count by $\frac{1}{0.192}$). Tables II and III show these results for CADs with respect to $F_1$ and $F_2$ respectively. In each column, the minimal values have been emboldened.

Consider first Table II concerning $F_1$. We see that $x \succ y \succ z$ offers the smallest and quickest 1-layered sub-CAD, and correspondingly, the smallest and quickest complete CAD. The total time for computing all six 1-layered sub-CADs is $3.829$ seconds which, along with computing the CAD for $x \succ y \succ z$, means that it would take $6.748$ seconds to obtain a CAD using this heuristic. This means the heuristic offers a maximum potential saving of $43.402$ seconds over computing directly the CAD for $z \succ y \succ x$ and 1672 cells over the CAD for $y \succ z \succ x$. If we had just picked one variable ordering at random then on average our CAD would have taken 18.773 seconds to compute and have 1431 cells. Hence on average the heuristic saves 12.025 seconds and 892 cells, and is faster than 2/3 of the individual computations.

TABLE II: Using 1-layered sub-CADs as a heuristic to pick the variable ordering for a CAD with respect to $F_1$.

| | Cells | | | Time | |
|---|---|---|---|---|---|
| Order | 1-LCAD | Prediction | CAD | 1-LCAD | CAD |
| $x \succ y \succ z$ | **118** | **615** | **539** | **0.341** | **2.919** |
| $x \succ z \succ y$ | 160 | 833 | 789 | 0.474 | 5.720 |
| $y \succ x \succ z$ | 340 | 1771 | 1799 | 0.736 | 13.055 |
| $y \succ z \succ x$ | 432 | 2250 | 2211 | 0.950 | 30.638 |
| $z \succ x \succ y$ | 224 | 1167 | 1133 | 0.549 | 10.156 |
| $z \succ y \succ x$ | 392 | 2042 | 2117 | 0.779 | 50.150 |

TABLE III: Using 1-layered sub-CADs as a heuristic to pick the variable ordering for a CAD with respect to $F_2$.

| | Cells | | | Time | |
|---|---|---|---|---|---|
| Order | 1-LCAD | Prediction | CAD | 1-LCAD | CAD |
| $[x, y, z]$ | 544 | 2833 | 2949 | 1.112 | 11.734 |
| $[x, z, y]$ | 710 | 3698 | 3995 | 1.025 | 13.870 |
| $[y, x, z]$ | **264** | **1375** | **1299** | 0.758 | **5.033** |
| $[y, z, x]$ | 312 | 1625 | 1545 | **0.527** | 5.784 |
| $[z, x, y]$ | 592 | 3083 | 3207 | 0.971 | 12.955 |
| $[z, y, x]$ | 448 | 2333 | 2347 | 0.711 | 9.006 |

Now consider the results for $F_2$ in Table III. This time the 1-layered sub-CAD cell counts correctly chooses ordering $y \succ x \succ z$ but if we used the timings we would have identified $y \succ z \succ x$, whose complete CAD is larger and takes longer to compute. This indicates that the cell count is a more useful and consistent measure for CAD complexity than computation time. The total time to compute all 1-layered sub-CADs for $F_2$ is $5.104$ seconds which means that using the heuristic and producing a CAD takes $10.137$ seconds. If we consider picking an ordering at random then on average the CAD would have 2557 cells and take 9.730 seconds. Hence for this example the heuristic would save a significant number of cells but the time savings would be outweighed by the computation cost.

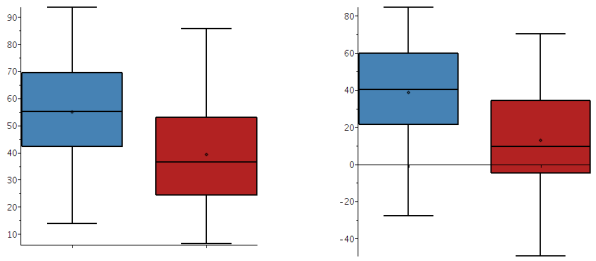
We repeated this experiment on 75 examples each with $\{f_1, f_2, f_3\}$ where $f_i$ are random polynomials in $\{x, y, z\}$ (two quadratic, one linear) generated with Maple's `randpoly` command. This time we build the 1-layered sub-CADs using the recursive algorithm (Algorithm 4 in [26]). This not only constructs a layered sub-CAD, but also a set of unevaluated function calls. When evaluated these produce both more CAD cells and another set of unevaluated calls. Combining the new cells with the existing layered sub-CAD will give a sub-CAD with an extra layer (that is, including those cells of one lower dimension). If we proceed until there are no evaluated calls left then the cells obtained give the complete CAD. The advantage here is that once an ordering has been selected we do not need to repeat the construction of the full-dimensional cells (giving a further modest saving). In the event that more than one ordering had the minimal number of full-dimensional cells we selected the first ordering lexicographically (equivalent to a random choice for these random examples).

We found that on average the heuristic saved 38.7% of the cells and 12.9% of the computation time for a problem

| | Cells | | Time | |
|---|---|---|---|---|
| | Problem Max | Problem Av | Problem Max | Problem Av |
| Average Example | $4,719$ $55.0\%$ | $2,220$ $38.7\%$ | $64.3$ $38.7\%$ | $10.0$ $12.9\%$ |
| Best Example | $12,816$ $93.6\%$ | $5,204$ $84.6\%$ | $631.9$ $84.6\%$ | $143.3$ $70.1\%$ |
| Worst Example | $762$ $13.9\%$ | $297$ $6.49\%$ | $-44.1$ $-27.9\%$ | $-66.1$ $-49.3\%$ |

TABLE IV: Using the number of full dimensional cells to pick the variable ordering for 75 random examples.

Fig. 6: Box plots showing the savings of using the 1-layered sub-CAD heuristic on the 75 random examples.



(a) Percentage savings in cells. (b) Percentage savings in time.

when compared to picking an ordering at random. However, this masks a lot of variance in the data. Table IV gives more details, showing also the examples with the best and worst savings. We see that even for the worst example the heuristic makes a cell saving, but that for some examples a time saving does not occur (or more likely the saving is outweighed by the computation cost of the heuristic). Table IV also compares not only to the average of the different orderings but also to the maximum values, showing the worst cases that can be avoided. Figure 6 shows box plots summarising the 75 examples reinforcing the findings:

- The cell savings are always positive, meaning the heuristic is an excellent tool for achieving a near minimal CAD. If the CAD in question is to be computed with extensively in a further application then this may be of great importance.
- The time saving can be negative (a cost). Hence when the aim is to use the heuristic to speed up computations care must be taken. Further, as the number of variables $n$ increases so too will the time required to compute $n!$ 1-layered sub-CADs and these potential costs. It is likely that a more appropriate use of the full-dimensional cells from this perspective may be to break ties that result from other (cheaper) measures, or for use when the underlying application (such as quantifier structure) limits the permissible orderings.
- When compared to the worst possibility for a problem instead of the problem average of course the savings are greater. Further, for all but a few outliers the time savings are positive. Hence the heuristic can be used as a risk-reduction measure.

### B. Algorithms to implement the heuristic

Algorithm 1 demonstrates how the heuristic described above could be implemented efficiently. We assume a generic CAD input $F$: this was a set of polynomials for the examples above but could more generally be a sequence of formulae say (see [4]). First in step 1 we must identify all admissible variable orderings (if there are no restrictions then this is simply all the permutations of the variables defining the polynomials). Throughout the variable $mc$ stores the minimum number of cells in a computed 1-layered sub-CAD. The admissible orderings are considered in turn and the full dimensional cells computed. Here (step 4) an algorithm should be used that is compatible with the required CAD (i.e. same invariance condition). Ideally it would also be recursive to avoid unnecessary calculation (as described above and in [26]). Hence for an ordering $v$ we produce both the 1-layered sub-CAD $L_v$ and a set of unevaluated function calls $U_v$. If the number of cells computed is a new minimum this is stored. At the end the variable ordering which contributed the minimal number of full dimensional cells has its layered sub-CAD extended to a complete CAD for the problem (step 8).

**Algorithm 1:** `LayeredHeuristic`

**Input** : A CAD input $F$.
**Output**: A CAD for $F$ and a variable ordering $opt$.

1 Set $V$ to be the admissible variable orderings;
2 $mc \leftarrow \infty$;
3 **for** $v \in V$ **do**
4 $\quad L_v, U_v \leftarrow$ `OneLayeredSubCAD`$(F, v)$;
5 $\quad$ **if** $|L_v| < mc$ **then**
6 $\quad\quad mc \leftarrow |L_v|$;
7 $\quad\quad opt \leftarrow v$;
8 $\mathcal{D} \leftarrow$ `FullCAD`$(F, opt, [L_{opt}, U_{opt}])$;
9 **return** $[opt, \mathcal{D}]$;

Algorithm 2 describes a (as yet unimplemented) parallel algorithm. As before we would construct 1-layered sub-CADs for all admissible variable orderings (step 3). We then await the first to finish and abort the rest (step 6). We compute the complete CAD for this ordering by evaluating the inert function calls in step 9 (which can also be in parallel).

Algorithm 2 chooses the ordering by time taken to compute the full-dimensional cells rather than the number of full-dimensional cells, used by Algorithm 1. We saw in Example 2 that the number of cells can be more closely correlated. We could modify Algorithm 2 to use cells instead by: starting computation of a complete CAD for an ordering only if it had fewer full-dimensional cells than any other computed; if at any point more than two complete CAD computations had been launched, abort the one with fewer full-dimensional cells.

### C. Other questions of problem formulation

We have focused so far on using the new observations on the distribution of CAD cell dimensions as a heuristic for choosing the variable ordering. However, essentially what we have is a measure of CAD complexity and so we could apply it to other questions of problem formulation for CAD.

**Algorithm 2:** `ParallelLayeredHeuristic`

**Input** : A CAD input $F$.
**Output**: A CAD for $F$ in a variable ordering *opt*.

Set $V$ to be the admissible variable orderings;
**for each** $v \in V$ **do in parallel**
  **launch** $L_v, U_v \leftarrow$ `OneLayCAD`$(F, v)$;
Let $v_0$ be the first to finish;
**foreach** $v \in V \setminus \{v_0\}$ **do**
  **abort** $L_v, U_v$;
**repeat**
  **for each** $c \in U_{v_0}$ **do in parallel**
    evaluate $c$ and add new cells to $L_{v_0}$ and new unevaluated calls to $U_{v_0}$;
**until** $U_{v_0}$ *is empty*;
**return** $[v_0, L_{v_0}]$;

*Example 3:* Consider again the polynomials

$$f = x - y^2, \qquad g = x^2 - y^2 - 1$$

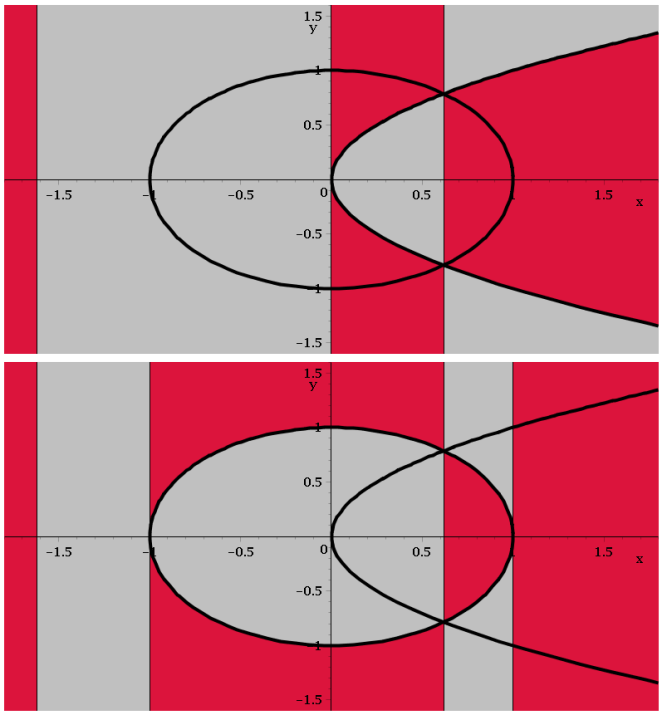
introduced in Example 1. When considered in Section I we built minimal sign-invariant CADs for these polynomials. However, depending on the underlying application these may provide more information than required. Suppose that $f$ and $g$ both formed **equational constraints** (ECs) for the problem: equations whose truth is logically implied by an input formula. In [10] a CAD invariant with respect to an EC was defined as a CAD sign-invariant for the polynomial defining an EC and sign-invariant for other polynomials only when that EC is zero. An algorithm to produce such CADs was later presented in [22]. An implementation of this in `ProjectionCAD` [16] produced a CAD invariant with respect to $f$ using 21 cells, and one with respect to $g$ using 25 cells.

The full dimensional cells of these CADs are shown in Figure 7. Using $f$ as the EC creates 8 full-dimensional cells and using $g$ creates 9. If both are ECs then we can choose to build either of the CADs and a judgement based on the number of full-dimensional cells would lead to the optimal choice.

**Remarks:** We note the following about this example:

1) Of course, when solving a system of equations there are far more efficient techniques to use than CAD. However, the concept of a CAD with respect to an EC described above is equally applicable to a system of equations and inequalities.
2) In this example both polynomials were ECs but savings were only made from one. Ideally, we could use an even simpler CAD where cells are invariant only with the truth of the conjunction of the ECs. Steps towards this minimal CAD are described in [2] where multiple ECs may be used (following the approach to CAD by regular chains computation originating in [9]). However, in this case there is an analogous questions of problem formulation: the order in which ECs are presented to the algorithm. This was investigated in [14] where heuristics were developed to help with the problem. It is likely that the number of full-dimensional cells could also be used to make this choice

Fig. 7: The image shows CADs built using $f = x - y^2$ and $g = x^2 + y^2 - 1$. The top image is sign-invariant with respect to $f$ and the second with respect to $g$. In each case the CAD is also sign-invariant with respect to the other polynomial when the first is zero.




Other questions of problem formulation for CAD are investigated in [5] and it is likely that the number of full-dimensional cells could be used as a heuristic for each (with similar caveats on how to use it as outlined above). We finish by considering a question of problem formulation itself (rather than how a problem is presented to a CAD algorithm).

*Example 4:* We consider the problem of moving a ladder of length 3 through a right-angled corridor of width 1 (moving from position 1 to position 2 in Figure 8). This is an example of a **piano movers problem** and was first proposed in [12], where it was noted that a CAD could be analysed to give an exact solution. Of course, a simple analysis shows there is no solution (with it possible only is the ladder is less than $\sqrt{8}$) but we are interested in how this could be decided automatically.

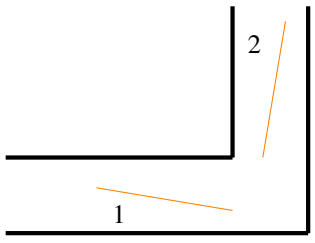



Fig. 8: The piano movers problem defined in [12]

In [12] the author described the feasible regions for the ladder, but found that computing a CAD was computationally infeasible, and 28 years later a CAD for this formulation is still

very costly. In [27] an alternative formulation was proposed: a description of the infeasible region was provided and negated instead. This simplified the CAD construction considerably.

We want to identify when such a reformulation of the problem would be beneficial. In [27], [26] the process of computing a **layered variety sub-CAD** is described: as well as restricting to cells of full-dimension we also restrict to cells on a given variety. The variety was an EC for both formulations (defining the length of the ladder). The process took around 200 seconds to produce 101,924 cells for the new formulation in [27] but timed out for the original one in [12], correctly identifying the tractable formulation for the complete CAD.

## IV. Conclusions

By considering empirical and combinatorial evidence we have shown that the distribution of cells by dimension in a CAD is consistent for a fixed number of variables. This means the size of a CAD can be accurately predicted from the number of full dimensional cells offering new heuristics for CAD problem formulation. Such heuristics can be made more efficient by using them with the recursive layered sub-CAD algorithm from [26] which allows for lazy evaluation of inert computations, avoiding recalculation of results. There is potential for further time savings through parallelism.

We demonstrated extensively that the number of full-dimensional cells is an effective heuristic for picking the optimal variable ordering for CAD, but that depending on the number of permissible ordering the time savings can be outweighed by the cost of running the heuristic. We also demonstrated how the ideas could be used for other questions of problem formulation, with the final example suggesting that the heuristic could be tailored further to the CAD required (using a layered variety sub-CAD when the complete CAD will be invariant with respect to an EC).

## Acknowledgements

This work was supported by EPSRC grant: EP/J003247/1.

## References

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

[2] R. Bradford, C. Chen, J.H. Davenport, M. England, M. Moreno Maza, and D. Wilson. Truth table invariant cylindrical algebraic decomposition by regular chains. To appear: *Proc. CASC '14*.
Preprint: http://opus.bath.ac.uk/38344/, 2014.

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

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

[5] R. Bradford, J.H. Davenport, M. England, and D. Wilson. Optimising problem formulations for cylindrical algebraic decomposition. In *Intelligent Computer Mathematics*, (LNCS 7961), pages 19–34. Springer Berlin Heidelberg, 2013.

[6] 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.

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

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

[9] C. Chen, M. Moreno Maza, B. Xia, and L. Yang. Computing cylindrical algebraic decomposition via triangular decomposition. *Proc. ISSAC '09*, pages 95–102. ACM, 2009.

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

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

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

[13] J.H. Davenport, R. Bradford, M. England, and D. Wilson. Program verification in the presence of complex numbers, functions with branch cuts etc. In *Proc. SYNASC '12*, pages 83–88. IEEE, 2012.

[14] M. England, R. Bradford, C. Chen, J.H. Davenport, M. Moreno Maza, and D. Wilson. Problem formulation for truth-table invariant cylindrical algebraic decomposition by incremental triangular decomposition. To appear: *Proc. CICM '14*, LNAI 8543, pages 46–60.
Preprint: http://opus.bath.ac.uk/39231/, 2014.

[15] 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. To appear: *Proc. ICMS '14*. Preprint: http://opus.bath.ac.uk/39492/, 2014.

[16] M. England, D. Wilson, R. Bradford, and J.H. Davenport. Using the regular chains library to build cylindrical algebraic decompositions by projecting and lifting. To appear: *Proc. ICMS '14*.
Preprint: http://opus.bath.ac.uk/39493/, 2014.

[17] 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.

[18] Z. Huang, M. England, D. Wilson, J.H. Davenport, L. Paulson, and J. Bridge. Applying machine learning to the problem of choosing a heuristic to select the variable ordering for cylindrical algebraic decomposition. To appear: *Proc. CICM '14*, LNAI 8543, pages 92–107. Preprint: http://opus.bath.ac.uk/39232/, 2014.

[19] H. Iwane, H. Yanami, H. Anai, and K. Yokoyama. An effective implementation of a symbolic-numeric cylindrical algebraic decomposition for quantifier elimination. In *Proc. SNC '09*, pages 55–64, 2009.

[20] S. McCallum. Solving polynomial strict inequalities using cylindrical algebraic decomposition. *The Computer Journal*, 36(5):432–438, 1993.

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

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

[23] L.C. Paulson. Metitarski: Past and future. In *Interactive Theorem Proving* (LNCS 7406), pages 1–10. Springer, 2012.

[24] A. Strzeboński. Solving systems of strict polynomial inequalities. *Journal of Symbolic Computation*, 29(3):471–480, 2000.

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

[26] D. Wilson, R. Bradford, J.H. Davenport, and M. England. Cylindrical algebraic sub-decompositions. To appear: *Mathematics in Computer Science*. Preprint: http://opus.bath.ac.uk/38148/, 2014.

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

[28] 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.