Department of
Computer Science

Technical Report

Layered Cylindrical Algebraic Decomposition

D. J. Wilson and M. England

Technical Report 2013-05 **August 2013**

ISSN 1740-9497

Contact Address:
Department of Computer Science
University of Bath
Bath, BA2 7AY
United Kingdom
URL: http://www.cs.bath.ac.uk

ISSN 1740-9497

LAYERED CYLINDRICAL ALGEBRAIC DECOMPOSITION

DAVID JOHN WILSON AND MATTHEW ENGLAND

Abstract. In this report the idea of a **Layered CAD** is introduced: a truncation of a CAD to cells of dimension higher than a prescribed value. Limiting to full-dimensional cells has already been investigated in the literature, but including more levels is shown to also be beneficial for applications. Alongside a direct algorithm, a recursive algorithm is provided. A related topological property is defined and related to robot motion planning. The distribution of cell dimensions in a CAD is investigated and layered CAD ideas are combined with other research. All research is fully implemented within a freely available Maple package, and all results are corroborated with experimental results.

Contents

Date: July 24, 2013.

1. Background

It is a well-established problem to study the behaviour of a system of polynomials over the real numbers. Initially devised by Collins ([Col75]) as a tool for real quantifier elimination, Cylindrical Algebraic Decomposition (CAD) is also a practical method of analysing such systems.

Definition 1.
Let $F \subset \mathbb{Z}[x_1, \ldots, x_n]$ be a finite set of polynomials. An **F-invariant Cylindrical Algebraic Decomposition** of $\mathbb{R}^n$ is a finite partition of $\mathbb{R}^n$ into cells such that:

- each cell is **algebraic**: can be described by a sequence of polynomial equalities and inequalities
- the cells are arranged **cylindrically**: the projections of any two cells onto the first k coordinates (for $k = 1, \ldots, n-1$) are equal or disjoint
- each cell is **F-invariant**: for each $f \in F$ and cell D, f is uniformly positive, negative or zero across D.

Whilst other methods ([CMXY09]) have been devised to construct CADs, we will concentrate on Collins' original algorithm, and its extensions. The algorithm consists of three main stages: projection, base case, and lifting.

Projection: A projection operator, **PROJ**, is used to repeatedly project out variables from the set of polynomials F. These polynomials are constructed to be delineable over lower dimensional CADs.

Base Case: After applying the projection operator $n-1$ times, the univariate polynomials are used to decompose $\mathbb{R}^1$ according to their real roots.

Lifting: We lift over a k-dimensional CAD to a $(k+1)$-dimensional CAD by taking a cell, D', and decomposing its cylinder, $D' \times \mathbb{R}$, according to the polynomials in $\mathbf{PROJ}^{(n-(k+1))}(F)$.

Rather than use use Collins' original projection operator, we will consider an improvement by McCallum (discussed in detail in [McC98]). Whilst all the theory which follows also holds for Collins' operator, using McCallum's allows for more efficient implementation (although a little care is needed to avoid 'non well-oriented' examples).

Whilst CAD is a very useful tool, it was shown in [BD07] that it has doubly-exponential complexity in the number of variables, both for computation time and output size. Indeed, the doubly-exponential complexity appears in the base case and is therefore inherent to the problem.

In an effort to compensate for this complexity, and make CAD construction more feasible, various techniques have been investigated to improve the CAD algorithm. For example, partial CAD ([CH91]) truncates the lifting stage of CAD construction, preconditioning by Gröbner bases ([BH91], [WBD12b]) reformulates the question, and truth table invariant CAD ([BDE+13]) asks a different question altogether. Singly-exponential algorithms exist for analysis of semi-algebraic sets and quantifier elimination over the real numbers but it was shown in [Hon91] that, in general, implementations of CAD are much more feasible, althought in special cases alternative algorithms can be used to great effect.

In this paper we investigate a new method to look at CAD: layered CADs. These concentrate on only cells of high dimension within a CAD and we give examples of applications where layered CADs are useful. We define a related topological property and also combine layered CADs with the ideas of [BDE+13]. All of this research is corroborated with experimental results.

2. Full-dimensional Cells

Let $\mathcal{D}$ be a CAD of $\mathbb{K}^n$ defined by the polynomials $F \subset \mathbf{k}[x_1, \ldots, x_n]$.

There are certain applications of CAD where lower-dimensional cells are of lesser importance than full-dimensional cells, such as branch cut analysis and robot motion planning (see Section 8 for a full discussion). For convenience we will refer to full-dimensional cells as **full**, and less than full-dimensional cells as **deficient**.

A simple way to obtain only full cells in a CAD is to only lift over full cells, discarding any deficient cells at each level during the lifting stage. This is described in Algorithm 1 and was implemented by the author in the `LayeredCAD` package in Maple which is described in further detail in Appendix A.

Algorithm 1: Basic generic CAD algorithm: `CADOneLayered(F,vars)`

Input : A set of polynomials F, a list of variables $vars$
Output: A generic CAD, D, for F

$\mathcal{D} \leftarrow [\,]$, $\mathbf{P} \leftarrow [F]$;
for $i = 1, \ldots, n-1$ **do**
  $\mathbf{P}[i+1] \leftarrow \mathbf{Proj}(\mathbf{P}[i])$;
$\mathcal{D}[1] \leftarrow$ `SplitR`$(\mathbf{P}[n])$;
for $i = 2, \ldots, n$ **do**
  $\mathcal{D}[i] \leftarrow [\,]$;
  for $D \in \mathcal{D}[i-1]$ **do**
    $dim \leftarrow \sum_{\alpha \in D.\text{index}} (\alpha \mod 2)$;
    if $dim == i$ **then**
      $\mathcal{D}[i]$.append(`GenerateStack`$(\mathbf{P}[n+1-i], D)$);
$\mathcal{D}' \leftarrow [\,]$;
for $D \in \mathcal{D}[n]$ **do**
  $dim \leftarrow \sum_{\alpha \in D.\text{index}} (\alpha \mod 2)$;
  if $dim == n$ **then**
    $\mathcal{D}'$.append(D);
return $\mathcal{D}'$;

This idea has already been considered by McCallum ([McC93]), Strzebonski ([Str94] and further in [Str00]), and Brown ([Bro13]). It is referred to by a multitude of names in the literature: **CADMD** (CAD of Maximal Dimension), **open CAD**, and **generic CAD**. We will initially refer to these as **generic CADs** in this paper (although later we will use the new terminology: 1-layered).

In both [McC93] and [Str94] the authors describe an algorithm similiar in essence to Algorithm 1. Their main motivation is solving systems of strict inequalities where any solution set must be, by necessity, full dimensional and therefore a generic CAD suffices. Another nice property of a generic CAD is that all sample points computed can be rational numbers: picking a point from an interval during each lifting stage does not require costly calculations involving algebraic numbers.

Asymptotically, computing a generic CAD is much faster than computing a full CAD (for very small examples the cost of checking dimensionality may outweigh the benefit), but it was pointed out in [McC93] that it is still worst-case doubly-exponential: the doubly-exponential complexity can appear in the base case of splitting $\mathbb{R}^1$. A deeper complexity analysis is given in Section 2.2.

2.1. **Order Invariance.**
We note a simple, but useful consequence of a generic CAD. The following lemma is needed.

Lemma 1.
Let $f \in \mathbb{R}[x_1, \ldots, x_n]$ vanish identically on a cell, D, of dimension n. Then f is identically zero on the whole of $\mathbb{R}^n$.

Proof.
We proceed by induction.

Let $n = 1$, and $D \subset \mathbb{R}^1$ an interval. As f vanishes on the whole of D, there exists at least $\deg(f) + 1$ points α_i such that $f(\alpha_i) = 0$ (there are infinitely many such points). Then by the Fundamental Theorem of Algebra, f must be the zero polynomial.

Now let $n > 1$. We view f as a univariate polynomial in x_n with coefficients in $\mathbb{R}[x_1, \ldots, x_{n-1}]$. As D is an n-dimensional cell, it must be in the cylinder of an $n-1$-dimensional cell D' in $\mathbb{R}^{n-1}$. Using the Fundamental Theorem of Algebra we can see that for every point $\alpha' \in D'$ every coefficient in f vanishes. But by induction, these coefficients must be zero on the whole of $\mathbb{R}^{n-1}$ and so $f \equiv 0$ on all of $\mathbb{R}^n$.

□

Using this lemma we can prove that any sign-invariant generic CAD is, automatically, order-invariant.

Theorem 1.
Let $F \subset \mathbf{k}[x_1, \ldots, x_n]$ and let $\mathcal{D}$ be a F-invariant generic CAD of $\mathbb{K}^n$. Then $\mathcal{D}$ is order-invariant.

Proof.
We will prove that the order of every polynomial in F on every cell in D is zero, and therefore $\mathcal{D}$ is trivially order-invariant.

For a non-zero polynomial $f \in F$ to have a non-zero order on a cell it must vanish in that cell and, as $\mathcal{D}$ is sign invariant with respect to F, it must vanish identically on that cell. For a cell $D \in \mathcal{D}$ this means that f vanishes identically on a cell of dimension n and so, by Lemma 1, f is the zero polynomial.

□

2.2. **Complexity.**
Extending the work of [Col75], [McC93] gave a thorough analysis on the computational complexity for constructing a generic CAD using the algorithm **CADMD** (similar to Algorithm 1 but using Collins' full projection operator).

Let F be a set of polynomials in n variables, let $|F| = m$, let d be the maximum degree of any polynomial in F in any variable, and let l be the maximum norm length (sum of the absolute values of its integer coefficients) of any such polynomial.

Theorem 2 ([Col75])**.**
The computational complexity of constructing an F-invariant CAD using the original Collins algorithm is dominated by:

$$(2d)^{2^{2n+8}} m^{2^{n+6}} l^3. \tag{1}$$

Therefore for fixed n this is a polynomial in d,m and l with degrees 4^{n+4},2^{n+6} and 3, respectively.

Theorem 3 ([McC93])**.**
The computational complexity of constructing an F-invariant generic CAD using

the algorithm **CADMD** *is dominated by:*

$$(2d)^{3^{n+4}} m^{2^{n+4}} l^3. \tag{2}$$

Therefore for fixed n this is a polynomial in d,m and l with degrees 3^{n+4},2^{n+4} and 3, respectively.

Proofs for both of these bounds start in the same way: let $\mathbf{A}_k := \mathbf{PROJ}^{(k-1)}(F)$, let $m_k := |\mathbf{A}_k|$, d_k be the maximum degree for $\mathbf{A}_k$, and l_k the norm length maximum for A_k. Then the following bounds ([Col75]) hold:

$$m_k \leq (2d)^{3^k} m^{2^{k-1}} \qquad d_k \leq \frac{1}{2}(2d)^{2^{k-1}} \qquad l_k \leq (2d)^{2^k} l. \tag{3}$$

These can then be extended to show that the projection phase of CAD (unchanged between full and generic CADs) is dominated by

$$(2d)^{3^{n+1}} m^{2^n} l^2 \tag{4}$$

and the base case (also unchanged between full and generic CADs) is dominated by

$$(2d)^{3^{n+3}} m^{2^{n+2}} l^3. \tag{5}$$

Part of the following saving in complexity for a generic CAD arises from the fact that sample points generated will be rational points from within intervals defining each cell. This saves the need to work with algebraic numbers which can be costly.

The high complexity in the base case shows that it is futile to consider breaking the doubly-exponential complexity barrier when using a projection and lifting method. However for a fixed n, there is a hope to improve the degree of the polynomial in d, m and l.

3. Layered CADs

There is an obvious extension to the idea of a generic CAD where you produce not only the maximal dimensional cells but also those of one dimension lower, and so forth. We formalise this idea:

Definition 2.
Let $F \subset \mathbf{k}[x_1, \ldots, x_n]$ be a finite set of polynomials and let l be an integer with $1 \leq l \leq n+1$. An **F-invariant l-layered CAD** is the subset of an F-invariant CAD consisting of all cells of dimension $n-i$ for $0 \leq i < l$.

We refer to a standard, non-layered CAD (consisting of all cells of all dimensions), as a **complete** CAD.

Remark 1.
Thus a 1-layered CAD is a generic CAD, and an $(n+1)$-layered CAD is a complete CAD.

A simple generalisation of Algorithm 1 can be used to compute an l-layered CAD: during the lifting process if a cell at a given level has too low a dimension to result in an l-dimensional (or higher-dimensional) cell then it is discarded. This is shown in Algorithm 2 and implemented in the `LayeredCAD` Maple package (further details in Appendix A).

3.1. Order-Invariance of 2-Layered CADs.

In Theorem 1 it was shown that one-layered CADs are, trivially, order-invariant. We show that dropping only one layer also provides order-invariance.

Theorem 4.
Let $F \subset \mathbb{Z}[x_1, \ldots, x_n]$ and let $\mathcal{D}$ be a F-invariant two-layered CAD of $\mathbb{R}^n$. Then $\mathcal{D}$ is order-invariant.

Algorithm 2: l-Layered CAD algorithm: `CADNLayered(F,l,vars)`

Input : A set of polynomials F, an integer l, a list of variables $vars$
Output: An l-layered CAD, D, for F

$\mathcal{D} \leftarrow [\,]$, $\mathbf{P} \leftarrow [F]$;
for $i = 1, \ldots, n-1$ **do**
  $\mathbf{P}[i+1] \leftarrow \mathbf{Proj}(\mathbf{P}[i])$;
$\mathcal{D}[1] \leftarrow$ `SplitR`$(\mathbf{P}[n])$;
for $i = 2, \ldots, n$ **do**
  $\mathcal{D}[i] \leftarrow [\,]$;
  for $D \in \mathcal{D}[i-1]$ **do**
    $dim \leftarrow \sum_{\alpha \in D.\text{index}} (\alpha \mod 2)$;
    if $dim > i - l$ **then**
      $\mathcal{D}[i]$.append(`GenerateStack`$(\mathbf{P}[n+1-i], D)$);
$\mathcal{D}' \leftarrow [\,]$;
for $D \in \mathcal{D}[n]$ **do**
  $dim \leftarrow \sum_{\alpha \in D.\text{index}} (\alpha \mod 2)$;
  if $dim > n - l$ **then**
    $\mathcal{D}'$.append(D);
return $\mathcal{D}'$;

Proof.
Applying Theorem 1 we see immediately that each $f \in F$ is order-invariant on all n-dimensional cells in D.

Now let D' be an $(n-1)$-dimensional cell, let x_i be the dimension for which it is deficient and assume $f \equiv 0$ on the whole of D', but not identically zero on $\mathbb{R}^n$. We will show that f cannot be identically zero, and therefore its order is trivially zero. View f as univariate in x_i and factor into its content and primitive part: $\text{cont}_{x_i}(f) \cdot \text{pp}_{x_i}(f)$. Then if $f \equiv 0$ on D' then $\text{cont}_{x_i}(f)$ must be identically zero on D' (as $\text{pp}_{x_i}(f)$ can only be zero at a finite set of points). But $\text{cont}_{x_i}(f)$ is a polynomial in $n-1$ variables that vanishes on an $(n-1)$-dimensional cell and so, by Lemma 1, it must be identically zero on $\mathbb{R}^{(n-1)}$. Therefore $f \equiv 0$ on $\mathbb{R}^n$ and we have the desired contradiction.

□

3.2. **Complexity.**

We would wish to give a dominating complexity for Algorithm 2 in terms of the number of variables, n, the number of polynomials in F, denoted by m, the maximum degree, d, of any polynomial in any variable, the maximum norm length, l, of any such polynomial along with N, the number of layers needed.

As in Section 2.2, we have a doubly exponential bound on then projection phase:

$$(2d)^{3^{n+1}} m^{2^n} l^2, \tag{6}$$

and the base case:

$$(2d)^{3^{n+3}} m^{2^{n+2}} l^3. \tag{7}$$

We cannot therefore escape the double exponential complexity in the worst case, and we cannot hope for a huge saving on the exponents of d, m and l for fixed n. Further, any N-layered CAD for $N > 1$ will necessitate the need for working with algebraic numbers which is known to be costly. Therefore it is likely a bound on

Example	Full-CAD		1-layered		2-layered	
	Cells	Time	Cells	Time	Cells	Time
CMXY 1	115	0.549	28	0.028	80	0.472
CMXY 2	895*	3.753	156	0.376	512	2.141
CMXY 3	333	2.770	64	0.097	204	1.030
CMXY 4	509	4.645	104	0.923	324	3.358
CMXY 5	55	0.192	18	0.026	45	0.100
CMXY 6	41	0.292	14	0.078	34	0.161
CMXY 7	897*	6.149	168	0.915	540	3.692
CMXY 8	365	3.137	66	0.507	218	1.543
CMXY 9	1099	6.094	228	1.288	712	4.304
CMXY 10	3673	38.008	680	9.086	2246	27.547
CMXY 13	4949	18.096	760	2.930	2684	12.537
Other 1	4569	132.487	778	9.922	2674	40.965
Other 2	41	0.130	13	0.035	33	0.084

TABLE 1. Results of Layered CADs (an $*$ indicates that the CAD produced is not guaranteed to be order invariant)

the complexity for any N-layered CAD with $N > 1$ will be close to that of a full CAD; for example, Collins' bound of:

$$(2d)^{2^{2n+8}} m^{2^{n+6}} l^3. \tag{8}$$

Exact complexity computations are currently being investigated in the hope to gain a bond dependent on n, m, d, l and N.

3.3. **Experimental Data.**

Using the `LayeredCAD` package for MAPLE the experimental data given in Table 1 was computed. The examples used were sourced from the CAD Repository detailed in [WBD12a] (available freely at `http://opus.bath.ac.uk/29503`).

This table demonstrates the obvious reduction in cells by taking a 1-layered or 2-layered CAD, and the subsequent saving in time. It also demonstrates that a 1-layered and 2-layered CAD can be order-invariant when a full CAD is not (CMXY 2 and 7).

It seems natural to ask whether it is possible to predict the number of cells in an l-layered CAD if we know the size of the Full CAD and, perhaps of more use, conversely. This is investigated in Section 6.

4. RECURSIVE ALGORITHM

In [CDM+10] an algorithm was given for the decomposition of a semi-algebraic system into regular chains representing the solutions. A regular chain is a particularly well-behaved polynomial system that is also used in [CMXY09] to construct CADs. It has been shown that real triangular decomposition, like CAD, has doubly-exponential complexity in the number of variables. The paper provides an algorithm for **lazy** triangular decomposition: outputting only regular chains of full (complex) dimension.

The true power of the algorithm is that it includes a recursive call to construct the next 'layer' of regular chains (of one less dimension) and a successive recursive call. This process can be repeated until a full decomposition is obtained. As they are truly working from full-dimension 'downwards' (unlike layered CAD which works upwards but discards surplus data along the way) this allows them to obtain a

singly-exponential complexity under certain conditions on the input. The doubly-exponential complexity still exists if you recurse fully, but the saving for a single layer is large.

We can adapt Algorithm 1 to produce a similarly recursive procedure. The general principle behind our method of constructing layered CADs is to stop lifting over a cell if it is produced as a section of a stack, rather than a sector. For convenience we will refer to this as **full-lifting**. Rather than simply discarding these sections, which we shall call **terminating sections**, when they are produced we could store them in a separate output variable.

When constructing a 1-layered CAD these terminating sections will have dimension $m-1$ where m is the number of variables lifted at the point the section is constructed: deficient by one dimension. Therefore in the final lifting stage the terminating sections will have dimension $n-1$ (and so are part of the 2-layered CAD). If we take one of these terminating sections and construct a stack over it by full-lifting we will construct cells that are of dimension $n-1$ and terminating sections that are deficient by two dimensions. Combining these $n-1$-dimensional cells with the 1-layered CAD produces a 2-layered CAD and the terminating sections can be used for a recursive call to construct a 3-layered CAD. In this manner we can recursively produce l-layered CADs.

Implementation is not quite as straightforward as Algorithm 1. The method is given in full in Algorithm 3 and implementation is recorded in Appendix A.

We require four inputs into our algorithm:

- `F`: The polynomials with which to construct the layered CAD with respect to.
- `vars`: The (ordered) variables.
- $\mathcal{C}$: A list of cells corresponding to terminating sections. If $\mathcal{C}$ is the empty list then a 1-layered CAD will be constructed, otherwise if $\mathcal{C}$ was produced computing an l-layered CAD an $(l+1)$-layered CAD will be created.
- $\mathcal{LD}$: A layered CAD as produced by this algorithm. If $\mathcal{LD}$ is the empty list than a one-layered CAD will be produced.

The algorithm will then output, for an l-layered CAD as input, both an $(l+1)$-layered CAD with respect to `F` and `vars` and a recursive call to produce an $(l+2)$-layered CAD if required.

Note the definition of $\mathbf{P}$ to be a global variable is a practical measure to prevent excessive recalculation of the projection polynomials which can sometimes be costly. A small amount of care is needed to ensure that the user does not try to call the algorithm with incompatible inputs.

The outputted recursive call should be in an inert form that can be evaluated. Maple provides this functionality by preceding a command with `%` which can be then evaluated using the `value` command. There is the added issue of finding a compact way to represent the inputs to the recursive calls but we are looking at achieving this through the introduction of new data types.

It is worth noting that it is not particularly inefficient to compute a full CAD by the recursive algorithm. The times given in Table 2 are the total time (including recursive calls) for producing layered CADs with the polynomials $x^2+y^2+z^2-1$ and $x+y+z-xy$ with variable order x, y, z. To compute a full CAD with Collins' original algorithm takes 5.044 seconds, although this includes much more diligent error checking than the layered algorithms, accounting for the greater computation time.

The implementation of Algorithm 2 is based on the general procedures of the `ProjectionCAD` package and so contains many more error checks which may explain why the time to compute a full CAD (4-layered CAD) differ. However, the real

Algorithm 3: Recursive layered CAD algorithm based on Algorithm 1: `CADRecursiveLayered(F, vars, C, LD)`

Input : A set of polynomials F; a list of variables $vars$; a list of lists of cells, $\mathcal{C}$, sorted into appropriate dimensions, to lift over (can be empty); a list of cells of the layered CAD already constructed, $\mathcal{LD}$, sorted into appropriate dimensions (can be empty).

Output: A layered CAD, $\mathcal{D}$, for F over $\mathcal{C}$ of one layer lower than $\mathcal{C}$; a recursive call to `CADOneLayeredRecursive` to construct the consecutive layer.

```
global P;
if P undefined then
    P ← [F];
    for i = 1, ..., n − 1 do
        P[i + 1] ← Proj(P[i]);
C' ← [ ];
if C == ∅ then
    D ← [ ];
    Base ← SplitR(P[n]);
    for i = 1, .., length(Base) do
        if (i mod 2) == 1 then
            D[1].append(Base[i]);
        else
            C'[1].append(Base[i]);
    D' ← [ ];
else
    D ← C;
    D' ← C[n];
for i = 2, ..., n do
    for D ∈ D[i − 1] do
        Dstack ← GenerateStack(P[n + 1 − i], D);
        for j = 1, ..., length(Dstack) do
            if (j mod 2) == 1 then
                D[i].append(Dstack[j]);
            else
                C'[i].append(Dstack[j]);
D' ← D' ∪ LD;
return [D', CADOneLayeredRecursive(F, vars, C', D')];
```

Layers	1	2	3	4
Algorithm 2	1.822	5.058	4.994	4.937
Algorithm 3	1.334	3.962	4.687	4.701

TABLE 2. Comparison of timings for Algorithms 2 and 3

power of recursive computation is demonstrated by the fact that if you have already computed a 3-layered CAD, computing a 4-layered CAD only takes 0.014 seconds with Algorithm 3 compared to 4.937 seconds using Algorithm 2.

5. Topology

We attempt to relate the concepts arising within layered CADs to topological properties. Recall the definition of an l-dimensional ball:

Definition 3.
The **l-dimensional ball** centered at the point $\alpha \in \mathbb{R}^l$ of radius $\epsilon > 0$, denoted by $B_\epsilon^{(l)}(\alpha)$, is defined to be:

$$B_\epsilon^{(l)}(\alpha) := \left\{x \in \mathbb{R}^l \mid |x - \alpha| < \epsilon\right\} \tag{9}$$

The idea of spaces being connected by paths is well-defined and studied within Topology.

Definition 4. [Wil04]
A space X is **pathwise connected** iff for any two points x and y in X, there is a continuous function $f : [0, 1] \to X$ such that $f(0) = x$, $f(1) = y$.

However, path-connectedness can be perhaps deceptive as the following example shows.

Example 1.
Consider the plane, $\mathbb{R}^2$, and the four open quadrants labelled clockwise from the positive quadrant. Let X be the first quadrant and Y the third quadrant.

The closures, $\overline{X}$ and $\overline{Y}$, are path connected. In particular, $\overline{X} \cap \overline{Y} = \{(0, 0)\}$ so any path between points in separate quadrants must pass through the origin.

For practical applications of CAD we often want something stronger. Unaware of a formal definition in Topology, we create such a definition here:

Definition 5.
A space X is said to be **l-dimensionally pathwise connected** if for any two points x and y in X, there is a continuous function $f : B_1^{(l)}(\mathbf{0}) \times [0, 1] \to X$ such that $f(\mathbf{0}, 0) = x$ and $f(\mathbf{0}, 1) = y$.

Clearly 0-dimensionally pathwise connected is equivalent to the definition of pathwise connected given in Definition 4.

Informally, 1-dimensionally and 2-dimensionally path-connected spaces are those in which you can connect any two points by a deformation of a non-trivial ribbon or cylinder, respectively.

Consider Example 1. We see that whilst the quadrants are 0-dimensionally path-connected, they are not 1-dimensionally path connected (unlike the closure of two adjacent quadrants).

With this definition we can consider connectivity of an l-layered CAD.

Theorem 5.
Let $\mathcal{D}$ be an l-layered CAD. Then each path-connected component of $\mathcal{D}$ is, at least, $(n - l + 1)$-dimensionally path connected.

Proof.
Let x and y be in the same path-connected component of $\mathcal{D}$, say X, and let $f : [0, 1] \to X$ be a path connecting x and y.

As $\mathcal{D}$ is l-layered, X is comprised of a union of a finite number of cells, each with dimension in the range $n - l + 1 \leq d \leq n$. For each cell D_i, let ϵ_i be the smallest distance from a point on the image of f to the boundary of the closure of D_i. As each cell is open this value is non-zero and an $(n - l + 1)$-dimensional ball of radius ϵ_i can surround every point of $f([0, 1])$ in D_i without intersecting the boundary of $\overline{D_i}$.

Choosing $\epsilon > 0$ such that $\epsilon < \min(\epsilon_1, \ldots, \epsilon_m)$ therefore proves that X is $(n-l+1)$-dimensionally path-connected.

□

6. CAD Dimensional Distribution

The idea of a layered CAD motivates the investigation of the distribution of dimensions of cells within a given CAD. This seemingly innocuous question hides a surprisingly tricky problem.

Definition 6.
Let $\mathcal{D}$ be a CAD of $\mathbb{R}^n$. Let $\mathfrak{D}_l$ be the number of cells of $\mathcal{D}$ of dimension l, for $l = 0, \ldots, n$.

6.1. **Empirical.**

An initial investigation of cell distribution in a collection of examples exhibited very similar patterns.

Figures 1 and 2 provide graphs of dimensional data for examples in [CMXY09] and [BH91].

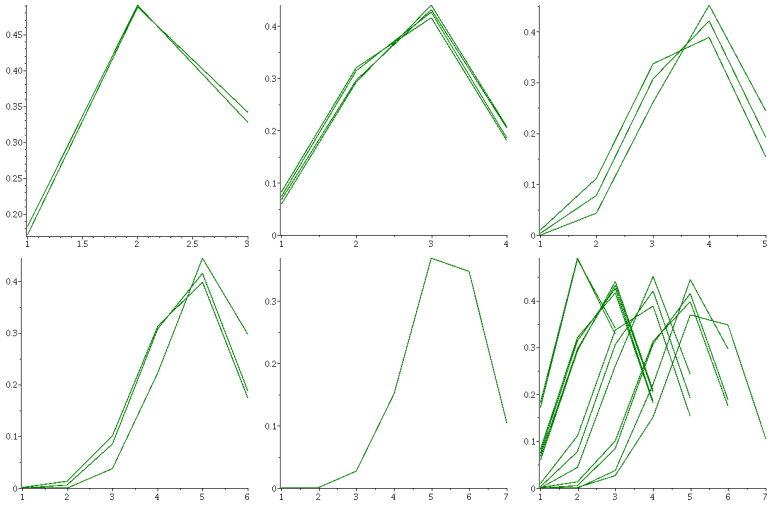

Figure 1. Dimensional distributions for a selection of examples from [CMXY09] split into 2-6 variables and all examples

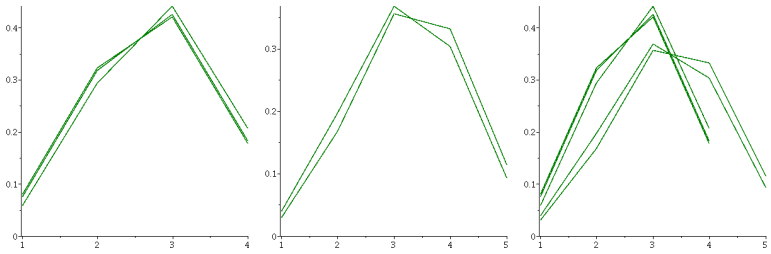

Figure 2. Dimensional distributions for a selection of examples from [BH91] split into 3 and 4 variables and all examples

There is a clear shape to most of the graphs: similar to a normal distribution that has been biased towards higher dimension cells. This can be emulated to a certain extent by a binomial distribution.

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

$$\mathbb{P}(X = x) := \begin{cases} \binom{n}{x} p^x (1-p)^{n-x} & 0 \leq x \leq n \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

Considering the binomial distribution with a high value of p gives a bias similar to what has been exhibited in the examples given. Figure 3 shows a binomial distribution with $n = 6$ and $p = 0.8$ alongside the 5-variable examples from [CMXY09] and their clear similarity.

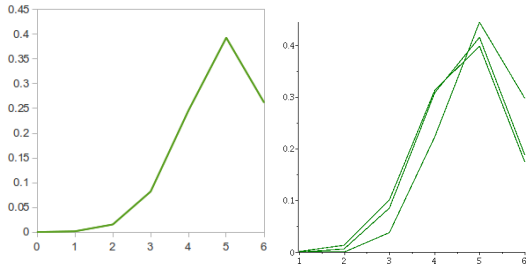

FIGURE 3. A binomial distribution with $n = 6$ and $p = 0.8$ with 5-variable examples from [CMXY09]

The fact that examples with the same number of variables gave similar distributions suggested that the distribution was unrelated to the specific polynomials used to generate the CAD. This was investigated by creating "random CADs": a random number of variables were chosen and each cylinder over a cell was split into a random number of cells. This was implemented in MAPLE using its inbuilt `rand` command to construct cell indices.

Figure 4 shows the output from 50 random CADs, each with variables chosen randomly from $\{0, \ldots, 6\}$ and each cylinder being split into a random number of parts chosen from $\{0, \ldots, 7\}$. This is given alongside the examples from [CMXY09] and it is clear that they have a strikingly similar distribution.

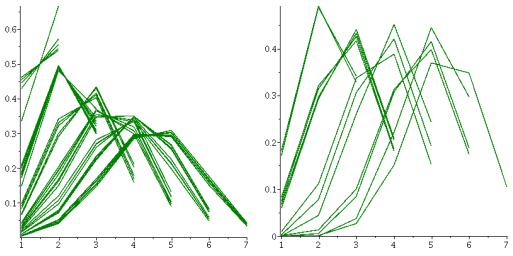

FIGURE 4. The cell distributions of `Random(6,7,50)` with the cell distributions from examples in [CMXY09]

It is worth noting that for the examples and random constructions the top two layers are generally the largest. This is unfortunate as often 2-layered CADs can be of great use (this will be discussed further Section 8).

6.2. **Theoretical Investigation.**
As the problem seemed to be independent of the real algebraic geometry it was worth considering the underlying combinatorial structure of a CAD. To do this we strip away the reliance on polynomials to construct a CAD and consider the following construction.

Lemma 2.
Let $\mathcal{D}$ be a partition of $\mathbb{R}^n$ constructed by the following:

1. *Decompose $\mathbb{R}^1$ using k_1 0-cells.*
2. *When lifting from the $\mathbb{R}^m$ to $\mathbb{R}^{m+1}$ decompose the cylinder over a cell of dimension m' using k_m (m')-cells.*

Decomposing $\mathbb{R}^n$ this way gives:

$$\mathfrak{D}_l = \sum_{\substack{L\subseteq[n]\\|L|=l}} \left(\prod_{i\in L}(k_i+1) \prod_{j\in[n]\setminus L} k_j \right), \tag{11}$$

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

In particular we have:

$$\mathfrak{D}_0 = \prod_{i=1}^{n} k_i \tag{12}$$

$$\mathfrak{D}_n = \prod_{i=1}^{n} (k_i+1) \tag{13}$$

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 l, it must therefore have l odd indices and $n-l$ even indices.

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

Therefore, for a fixed L, the number of cells that have an appropriate cell index is given by:

$$\prod_{i\in L}(k_i+1) \prod_{i\in[n]\setminus L} k_i. \tag{14}$$

All that remains is to sum over all possible subsets L of $[n]$ with cardinality l. □

This is a gross simplification of the problem at hand, however it does well to model standard CAD behaviour empirically.

It is very easy to see that if $k_i = k$ for all i then the sequence $\{\mathfrak{D}_l\}$ is simply the sequence of binomial coefficients of $(k+(k+1)x)^n$. In fact, we can state the following:

Lemma 3.
Let $\mathcal{D}$ be as given in Lemma 2. Then the generating function for $\mathfrak{D}_l$ is given by:

$$\Gamma(x) := \prod_{i=1}^{n} (k_i + (k_i+1)x). \tag{15}$$

That is, $\mathfrak{D}_l$ is the coefficient of x^l in the expansion of $\Gamma(x)$.

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

$$\prod_{i \in L} (k_i + 1) \prod_{i \in [n] \setminus L} k_i. \tag{16}$$

The coefficient of x^l in $\Gamma(x)$ is precisely the summation of all such coefficients:

$$\sum_{\substack{L \subseteq [n] \\ |L| = l}} \left(\prod_{i \in L} (k_i + 1) \prod_{j \in [n] \setminus L} k_j \right), \tag{17}$$

which was shown in Lemma 2 to be precisely $\mathfrak{D}_l$.

□

This lends some credibility to the idea that the dimension of cells in a CAD roughly obeys a binomial statistical distribution.

It may seem at first like this construction has somehow avoided the doubly-exponential complexity of CAD. This is not the case: taking a resultant at each level in projection results in an increase of each degree by a factor of 2^n.

6.3. **Prediction of CAD size.**

This consistency in dimensional distribution can be of great use. As a one-layered CAD is generally easy to compute we can use it to predict the size of a complete CAD.

The obvious way to do this is to compute the dimensional distribution for a collection of examples with the same number of variables and take the average of the fraction of full cells in each. Multiplying the number of full cells for the layered CAD in question by the reciprocal of this average should give a reasonable estimate for a complete CAD.

Note that can also create 'random' distribution fractions by using the formulae obtained in Lemma 2 so that:

$$\frac{\mathfrak{D}_l}{\sum_{i=1}^n \mathfrak{D}_i} = \frac{\sum_{\substack{L \subseteq [n] \\ |L| = l}} \left(\prod_{i \in L} (k_i + 1) \prod_{j \in [n] \setminus L} k_j \right)}{\prod_{i=1}^n (2k_i + 1)} \tag{18}$$

and in particular the fraction the top layer constitutes is given by the simple formula:

$$\frac{\mathfrak{D}_n}{\sum_{i=1}^n \mathfrak{D}_i} = \frac{\prod_{i=1}^n (k_i + 1)}{\prod_{i=1}^n (2k_i + 1)}. \tag{19}$$

This simplifies even further when all the k_i are equal, simply giving:

$$\frac{\mathfrak{D}_n}{\sum_{i=1}^n \mathfrak{D}_i} = \left(\frac{k+1}{2k+1} \right)^n. \tag{20}$$

Computing values for examples sourced from [WBD12a] we can also calculate average fractions of full cells in distributions for a given number of variables. These results are shown in Table 3.

Variables	2	3	4	5
Fraction	0.334	0.192	0.161	0.181

TABLE 3. Average distribution of full cells from [WBD12a]

Obviously using an averaged value to predict the size of a complete CAD will not work universally but initial results are promising, as shown in the following example.

Example 2.
Two random polynomials in x, y, z were created, each with total degree 2:

$$g := 81zy - 3zx - 88yx + 99x^2 - 12y - 48; \tag{21}$$

$$h := 80z^2 + 40zy - 41y^2 + 44yx - 28z + 11. \tag{22}$$

These were inputted into the CAD algorithms under the two variable orderings: $x \prec y \prec z$ and $x \succ y \succ z$.

With variable ordering $x \prec y \prec z$ a 1-layered CAD was produced with 372 cells which give a predicted full CAD of 1941. In reality, a full CAD produced by McCallum's projection contains 2057 which is a difference of 116 (around 5%).

With variable ordering $x \succ y \succ z$ a 1-layered CAD was produced with 454 cells which give a predicted full CAD of 2368. In reality, a full CAD produced by McCallum's projection contains 2373 which is a difference of only 5 cells (around 0.2%).

It would seem, as to be expected, that larger CADs have more accurate predictions. For example when dealing with only a single polynomial with a CAD of at most 200 cells the results can be inaccurate (up to a factor of 100%) but with more polynomials and CADs over 1000 cells the predictions seem to mostly be within 5%. Arguably, the larger examples are those for which you would want to have a prediction (and avoid computing a full CAD if necessary) so this is not a huge problem.

Indeed, it seems the most appropriate use of this approximation is estimating feasibility of problems, an example of which is given in Section 8.1.

6.4. **Variable Ordering.**

It is well known that variable ordering is fundamental to CAD problems. As shown in [BD07] there are examples for which variable ordering is of key importance: producing a constant number of cells (as few as 3) with one ordering and a doubly exponential (in the number of variables) number of cells with another variable ordering. This obviously means that different variable orderings can produce differing distributions.

We do not take this into account with our estimations and it is quite possible that for a given problem the estimate with one variable ordering may be highly accurate whilst another ordering is inaccurate. However averaged out over many problems we believe that the CAD dimension distributions should be fairly consistent.

Therefore constructing a 1-layered CAD and predicting overall CAD size could be used as a heuristic for choosing variable ordering. Although much more costly than existing heuristics (see, for example, [DSS04], [BDEW13], and [Eng13b]) it could be used as a 'tie-breaker' or when a truly accurate prediction is needed.

7. Extension to TTICAD

In [BDE+13] the authors suggested that often in CAD problems we ask the incorrect question. It is often not the behaviour of specific polynomials that we care about but rather the truth of Tarski quantifier-free formulae (QFFs) comprised of them.

Given a sentence Φ comprised of quantifier-free formulae ϕ_i they propose that rather than compute a full CAD for all polynomials in Φ a **Truth Table Invariant CAD (TTICAD)** should be computed instead. This is a CAD where the truth

value of each ϕ_i is constant over each cell. Generalising work in [McC99], an algorithm is given for the case that each quantifier free-formula contains a designated **equational constraint** (an equation of the form $f = 0$ which is logically implied by the rest of the clause).

Example 3.
Consider the following single-clause formula:

$$(xyz - \frac{1}{4} = 0) \wedge (x^2 + y^2 + z^2 - 1 < 0) \wedge (x + y + z - 1 > 0). \tag{23}$$

As there is only a single clause, TTICAD works much the same as equational constraints here apart from an improvement in the lifting stage described in [Eng13b]. Instead of including $x^2 + y^2 + z^2 - 1$ and $x + y + z - 1$, a CAD is constructed using $xyz - \frac{1}{4}$ (the designated equational constraint) and its resultants with the other two polynomials. This is sufficient as we only need care about the behaviour of the inequalities when the first equation is satisfied.

In this case a projection based CAD, using McCallum's projection operator, produces 1069 cells, and a CAD using the technology of [CMXY09] produces 1077 cells. Using TTICAD produces just 127 cells, encapsulating the behaviour of the clause.

Example 4.
Consider the following two-clause formula:

$$\left(x^2 + y^2 - 1 = 0 \wedge xy - \frac{1}{4} < 0\right) \vee \left((x-4)^2 + (y-1)^2 - 1 = 0 \wedge (x-4)(y-1) - \frac{1}{4} < 0\right). \tag{24}$$

In this example, as the two QFFs are disjuncted together, the traditional theory of equational constraints seems to not be applicable. In fact, an implicit equational constraint is present: the product of the two equations. The power of TTICAD is due to the observation that we can apply the theory of equational constraints separately to each QFF, worrying only about interaction between the equational constraints of each QFF.

A classical projection-based CAD, using McCallum's operator, for this problem produces 377 cells, as does a CAD using the technology of [CMXY09]. Using the implicit equational constraint results in 249 cells being produced. Applying TTICAD reduces the number of cells to 153 while still being able to fully determine the behaviour of the formula.

A natural idea is to combine TTICAD with layered CAD. This is surprisingly easy and conducted almost identically to Algorithm 2 but with the TTICAD projection and lifting operators. A full implementation is given by the author in the `LayeredTTICAD` Maple package and is discussed in Appendix A.

Table 4 gives results for using `LayeredTTICAD` with Example 3 and 4. As is to be expected, taking a layered TTICAD offers savings over a complete TTICAD.

It is also worth asking if the ideas in Section 6 transfer to TTICADs. In particular, we can ask if the dimensional distribution of a TTICAD differs from the distribution of a standard CAD. Random inputs in 2 and 3 variables were considered: a formula was constructed with a random number of clauses between 1 and 4, each had a random equational constraint (using the `randpoly` command in Maple) with a random degree between 1 and 4, and a random number (between 1 and 4) of random non-equational constraints with a random degree between 1 and 4. Running 10 tests for each set of variables gave the average distributions shown in Table 5.

	Example 3		Example 4	
	Cells	Time	Cells	Time
MapleCAD	1077	14.887	377	4.300
CADFull	1069	115.658	377	3.648
2-layerCAD	684	9.558	301	2.377
1-layerCAD	216	2.987	113	1.386
TTICAD	127	0.692	153	0.473
2-layerTTICAD	96	0.642	121	0.528
1-layerTTICAD	36	0.272	45	0.462

TABLE 4. Various results for Layered TTICAD

Variables	0-cells	1-cells	2-cells	3-cells
x, y	0.183	0.488	0.329	—
x, y, z	0.052	0.256	0.436	0.256

TABLE 5. TTICAD Distributions

Comparing with the results in Table 3 we can see that although the 2-dimensional distribution agrees fairly well, the 3-dimensional distribution differs somewhat. In the standard CAD testing a proportion of 0.192 cells were 3-dimensional, whereas over a quarter of TTICAD cells are 3-dimensional. This could be a manifestation of the random nature of this testing, a consequence of the construction method of TTICAD, or, more likely, a combination of the two factors.

Further research into layered TTICADs and their applications is being conducted.

8. Applications

In [McC93] the author pointed out the most obvious use of 1-layered CADs (and, their inspiration) for solving systems of strict inequalities (which can only be satisfied by full cells).

Later, in [Str00] discussed how considering only full cells can be useful for integration. If two sets, A and B, differ only by a set of n-dimensional Lebesgue measure zero then:

$$\int_A f \mathrm{dm} = \int_B f \mathrm{dm}. \tag{25}$$

Therefore integrating only over full cells in a certain region is equivalent to integrating over all cells in that region (as a deficient cell has is Lebesgue-null). The author also notes that open cells have further use in graphical visualization of sets defined by strict inequalities.

8.1. Cyclic-n Polynomials.

The Cyclic-n problem defines a set of n polynomials symmetric in n variables as follows:

$$x_1 + x_2 + \cdots + x_n = 0 \tag{26}$$

$$x_1x_2 + x_2x_3 + \cdots + x_{n-1}x_n + x_nx_1 = 0 \tag{27}$$

$$\vdots \tag{28}$$

$$x_1x_2 \cdots x_{n-1}x_n - 1 = 0. \tag{29}$$

We consider the Cyclic-4 problem in the following example:

Example 5.
The Cyclic-4 *problem involves the following polynomials:*

$$a + b + c + d \tag{30}$$

$$ab + bc + cd + da \tag{31}$$

$$abc + bcd + cda + dab \tag{32}$$

$$abcd - 1. \tag{33}$$

Under any variable ordering a 1-layered CAD contains 10,440 cells. Using our current estimations this predicts a complete CAD with 64,845 cells - well outside our current feasible computational range.

This is indeed the case, as shown in [WBD12b]*, where computing a full CAD timed out after 1000s. It was found to be computable within a minute with preconditioning by Gröbner Bases. Indeed after this preconditioning a 1-layered CAD contains 252 cells which gives a much more manageable estimate of 1565 cells in a complete CAD. With McCallum's projection operator a cell count of 1569 is obtained showing the accuracy of the prediction. In fact, with better projection operators or the technology of* [CMXY09] *a cell count of 621 post pre-conditioning can be obtained.*

The following result is known and a proof is given in [Fau00]:

Lemma 4. *If n is square free then there are a finite number of solutions to Cyclic-n. If there exists $m > 1$ such that $m^2|n$ then there are an infinite number of solutions to Cyclic-n and they have dimension at least $m - 1$.*

Therefore the Cyclic-4 problem has an infinite number of solutions and the solution set has dimension at least 1. Therefore a full CAD is not necessary to find a set of solutions: we can construct a 4-layered CAD to only produce cells of dimension 1 and higher. Obviously this is not a great saving, but for larger n (and larger m) this could constitute a significant saving.

8.2. **Intersection of surface and regions.** Consider the following problem:

$$x^2 + \left(y - \frac{1}{2}\right)^2 + z^2 < 1 \wedge x^2 + \left(y + \frac{1}{2}\right)^2 + z^2 < 1 \wedge y^2 - x^3 + z = \frac{1}{10}$$

Which is the intersection of a surface with the interior of two spheres shown in $\mathbb{R}^3$, as shown in Figure 5.

We may wish to know if there is a solution to this problem and a description of the region without worrying about the boundaries and particular points, or we may wish to integrate over this surface and so need to know the cells of greatest dimension (as discussed in [Str00]). We can see that any solution region will have a co-dimension of at least 1, so a 2-layered CAD should be created. We will use the variable ordering $z \prec y \prec x$.

	Cells	Time (s)
1-layered CAD	544	44.939
2-layered CAD	1838	479.707
Full CAD	3063	6624.407

TABLE 6. Results for computing various CADs under the variable ordering $z \prec y \prec x$

The results of computing a 1-layered, 2-layered, and Full CAD are given in Table 6. Although the 2-layered CAD contains over half the total number of cells, computing a full CAD is nearly 14 times slower.

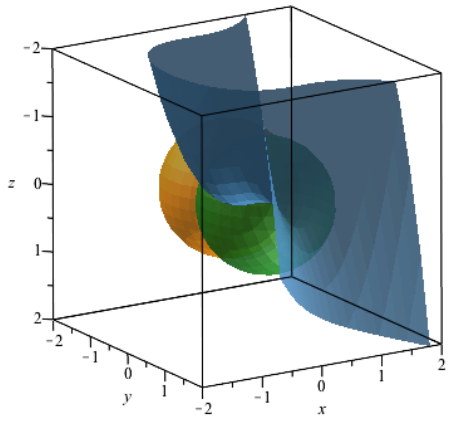

FIGURE 5. Intersection of the surface $z = x^3 - y^2 + \frac{1}{10}$ with unit spheres centered at $(0, \frac{1}{2}, 0)$ and $(0, -\frac{1}{2}, 0)$

As it consists of a conjunction of equations and inequalities, this problem is a candidate for treatment by equational constraints or TTICAD. As we have only one clause, TTICAD is in essence working the same as equational constraints (there are some differences in the particulars related to the lifting but they are relatively inconsequential). We have to select which of the spheres to designate as the equational constraint and to do this we can use the measures `sotd` and `ndrr`; this idea is discussed in [BDEW13]. As the problem is essentially symmetric in y, there is no difference between the choice of equational constraints.

	Cells	Time (s)
1-layered TTICAD	112	1.949
2-layered TTICAD	363	8.521
Full TTICAD	599	23.707

TABLE 7. Results for computing various TTICADs under the variable ordering $z \prec y \prec x$

The results of computing layered and full TTICADs are shown in Table 7. A 2-layered TTICAD is the minimal CAD we can currently construct to solve the problem (this may not be the smallest possible) and so through TTICAD and layered CADs we have reduced the number of cells produced by a factor of 8 from 3063 to 363, and reduced the time taken by a factor of 777 from nearly two hours to under 9 seconds.

8.3. **Branch Cut Analysis.**

Many elementary functions on the complex numbers are multi-valued; for example the natural logarithm function produces a set of valid answers for a complex number z:

$$\log(z) = \{\log(|z|) + 2\pi i k \mid k \in \mathbb{Z}\}. \tag{34}$$

Most computer algebra systems deal with this by defining a single valued counterpart for the function along with choosing a set of branch cuts defining where the

function is discontinous. Whilst the cuts for the elementary functions are simple (for the logarithm and square root the standard choice is the negative real axis) an expression involving multiple, possibly nested, functions can have complicated branch cuts.

CAD can be used to analyse the branch cuts for an expression and deduce where it holds in complex space. This was discussed in [BDPB07] and has recently been implemented in Maple 17 [EBDW13]. The method takes an expression in n complex variables and, by separating real and imaginary parts, considers it as an expression in $2n$ real variables.

Recall the following definition:

Definition 7.
Two functions, f_1 and f_2, defined on a domain X are said to be **equal almost everywhere** with respect to a measure μ if the subset $E \subseteq X$ of values where f_1 and f_2 differ is a null set (i.e. $\mu(E) = 0$).

Note that with respect to the Lebesgue measure (the standard measure on Euclidean space) any subset of $\mathbb{R}^n$ of deficient dimension has null measure.

We can therefore use layered CADs to decide if a complex expression is valid almost everywhere in complex space. Note that if we want a formula to hold almost everywhere with respect to complex dimension we would need to construct a 2-layered real CAD to correspond to a 1-layered complex CAD. However we only need to construct a 1-layered CAD of real space as a $2n-1$-real-dimensional cell cannot represent an n-complex-dimensional cell.

It is possible to use layered CADs to do better than just equality almost everywhere. In [BBDP05] the concept of adherence was investigated within the context of branch cut analysis.

Definition 8 ([BBDP05])**.** Suppose that we have a CAD for $B(h)$ (a set of branch cuts for h) and c is a section of a stack representing part of, or all of, a particular branch cut. Let s be an adjacent sector cell to c. Then we say that the branch cut c **adheres** to s if c belongs to the same sheet of $R_S(H)$ (the associated Riemann surface) as does s.

If c adheres to s, then the validity of a given identity is the same on c and s. As it is easier to work on a cell of higher dimension (especially full dimension, thanks to rational sample points and no risk of computational errors) this can be of great advantage.

Therefore, it would seem that the full dimensional cells are of greatest importance, and 2-layered cells can be analysed through adherence. However, it should be noted that computing adherences is highly non-trivial: the sub-procedure of computing interstack adjacencies alone is rather complicated.

8.4. **Motion Planning Examples.**

Consider the problem of moving a 3m ladder along a hallway of width 1m that has a 90° bend, as shown in Figure 6.

As shown in [Dav86] we can describe this problem by cosidering the configuration space of the two endpoints (x, y) and (x', y') in the following manner:

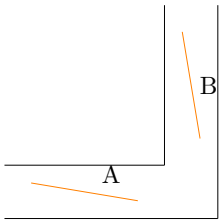

FIGURE 6. Is it possible to move the ladder from position A to position B?

$$
\begin{aligned}
(35)\quad \Big[\ & [(x-x')^2+(y-y')^2-9=0]\ \wedge \\
& [[yy'\geq 0]\ \vee\ [x(y-y')^2+y(x'-x)(y-y')\geq 0]]\ \wedge \\
& [[(y-1)(y'-1)\geq 0]\ \vee\ [(x+1)(y-y')^2+(y-1)(x'-x)(y-y')\geq 0]]\ \wedge \\
& [[xx'\geq 0]\ \vee\ [y(x-x')^2+x(y'-y)(x-x')\geq 0]]\ \wedge \\
& [[(x+1)(x'+1)\geq 0]\ \vee\ [(y-1)(x-x')^2+(x+1)(y'-y)(x-x')\geq 0]]\ \Big].
\end{aligned}
$$

This problem is infeasible with all of the current implementations. Even constructing a 1-layered CAD is still infeasible with current work. An alternative reformulation of the problem that is feasible with current technology is the topic of a future paper.

However, this is a useful example to demonstrate how layered CADs can be useful for motion planning. Moving a 2-dimensional object through a 1-dimensional cell is, in reality, impossible; it is akin to walking along an infinitely thin tightrope.

In fact, l-dimensional path connectedness is the appropriate property needed. Consider moving a sphere through 3-dimensional space. Certainly a hole the size of the "shadow" of the sphere (i.e. a circle) is need to pass the sphere between regions. This can be generalised into a necessary condition: to move an $(l+1)$-dimensional object between two regions they must be l-dimensionally path connected. Obviously it still may not be possible to move the object between regions (you cannot pass a sphere of radius 10m through a gap that is only 1cm wide!) but this condition is still very useful.

It is worth noting that it may not be obvious how many layers of a CAD are needed to correspond to a given dimension issue. For motion planning problems we generally work in a configuration space rather than the physical space which may be of higher dimension. In the ladder problem above, we work in a four-dimensional space (in the variables x, y, x', y') although we consider a subspace of co-dimension 1 (due to the equation defining the ladder's length).

9. Conclusions and Further Work

Although the idea of looking at just the full-dimensional cells of a CAD has been investigated before, we have shown that constructing multiple layers can also be of benefit. Trivially, 1-layered CADs are order-invariant, but the same can be said for 2-layered CADs as well. Algorithms have been given to compute l-layered CADs both directly and recursively: both are fully implemented in the `LayeredCAD` package for Maple described in Appendix A.

A new topological concept of l-dimensional path connectedness was introduced, motivated by motion planning where this property is necessary (although not sufficient). An l-layered CAD can be used to decide whether cells are $(n-l+1)$-dimensionally path connected. This can save the need to construct a complete CAD.

Layered CADs can also be used to predict the size of the equivalent complete CAD. There is evidence that the distribution of dimensions of cells within a CAD is approximately binomial (with a high p value) and this can be used as a crude estimate. This can be used as an important test for the feasability of problems.

The idea behind a layered CAD can also be combined with the recent idea of TTICAD (which, itself, extends the theory of equational constraints). A major application of TTICAD is in branch cut analysis, for which layered CADs can be used to prove expressions almost everywhere. The concept of adherence can also be used to determine the validity of identities on the branch cuts from a 1-layered CAD.

Future work will further investigate properties and applications of layered CADs and a large aim is to construct them in a more efficient way. It may be possible to construct layered CADs from the algorithm given in [CMXY09] and emulate the work of [CDM+10] to produce 1-layered CADs in singly-exponential time under certain conditions.

Appendix A. Implementation

The ideas contained in this paper have been implemented by the first author in the Maple package `LayeredCAD`, available freely for download alongside this paper or from the author's website[1].

The `LayeredCAD` package is an extension for the `ProjectionCAD` package (version 2.0+) created by the second author for the University of Bath *Real Geometry and Connectedness* research group. The original package allows users to construct CADs with a range of projection operators and is described in [Eng13a] and [Eng13b]. The package is also freey available on the author's website.

The `LayeredCAD` package contains eleven new top-level functions for constructing layered CADs:

- `CADOneLayered(F,vars)`, `CADNLayered(F,N,vars)`;
- `CADRecursiveLayered(F,vars,C,LD)`, `CADRecursiveLayeredLR(F,vars,C,LD)`;
- `CADDisplayLayered(CAD)`;
- `TTICADOneLayered(PHI,vars)`, `TTICADNLayered(PHI,N,vars)`;
- `CADDistribution(F,vars)`, `CADNormDist(F,vars)`;
- `TTICADDistribution(PHI,vars)`, `TTICADNormDist(PHI,vars)`;

These functions and their implementations are described in this Appendix along with an interactive Maple worksheet freely available with the package.

A.1. `CADOneLayered` and `CADNLayered`.

These functions compute their respective layered CADs using Algorithms 1 and 2 and output them as a list of cells represented by a list of indices and a regular chain representation. They take a list or set of polynomials, an (ordered) list of variables, and (for `CADNLayered`) a positive integer between 1 and $n+1$.

Further functionality can be provided with additional input specifiers. Providing `output=listwithrep` can request the output in a *listwithrep* format which provides for each cell a readable representation. Providing `method=McCallum` or `method=Collins` will specifiy the respective projection operator to be used.

The algorithms use the following simple test procedure:

```
IsCellNLayeredDim:=proc(cell::list,N::int) :: boolean:
    local cellindex, modtot, n, i:
    n:=nops(cellindex):
    modtot:=add(i mod 2, i in cellindex):
    if modtot >= (n+1-N) then
        return(true):
    elif modtot<(n+1-N) then
        return(false):
    fi:
end proc:
```

The simple idea behind `IsCellNLayeredDim` is that a cell's dimension can be easily determined by the parity of its indices:

$$\dim(D) = \sum_{i \in \mathrm{Ind}(D)} (i \mod 2)\,. \tag{36}$$

Therefore a cell is a valid cell for a N-layered CAD if this dimension is greater than or equal to $n+1-N$, where n is the number of indices of the cell (i.e. its maximal dimension).

[1] http://www.cs.bath.ac.uk/~djw42/triangular

A.2. `CADRecursiveLayered` **and** `CADRecursiveLayeredLR`**.**
Algorithm 3, and the ideas of Section 4, have been implemented in the two given procedures.

The two functions have the same specifications but differ in how the CADs are represented: the former as a list of cells (index and sample point), the latter also provides representations for each cell (equations and inequalities describing the cell). The input is a set or list of polynomials, an ordered list of variables along with a list of cells `C` and layered CAD `LD` - both of these can be empty lists. The list `C` should be those cells where the algorithm had terminated for the previous layer and the layered CAD `LD` should be the previous decomposition. It is intended that the user should never manually enter these but they should be automatically generated by previous calls to the algorithm.

The algorithms then propogate the next layer of the CAD from the cells in `C` and give a pair of outputs:

```
> A,B:= CADRecursiveLayered(F,vars,C,LD);
        LD', %CADRecursiveLayered(F,vars,C',LD')
```

The first output, `LD'`, is a layered CAD for `F` and `vars` of precisely one layer more than `LD`. The second output is a recursive call that can be used to produce a layered CAD with one more layer (and another recursive call). This recursive call is rendered inert by the Maple identifier `%`, but can be evaluated using the `value` command.

```
> A',B':= value(B);
        LD'', %CADRecursiveLayered(F,vars,C'',LD'')
```

In the current implementation the recursive call, `B`, is given in full detail, with `C'` and `LD'` explicitly printed. This is less than ideal, as the cells and layered CADs are often very large and complicated lists. To this extent it is recommended to assign the call 'quietly' (using `:` to terminate the statement rather than `;`) and then evaluate separately. It is hoped in future versions of the package this will be avoided through explicit type declaration (as is planned for the whole `ProjectionCAD` package).

A.3. `CADDisplayLayered`**.**
If a layered CAD has been constructed using the `listwithrep` formulation then it can be displayed in an informative tree-like structure using `CADDisplayLayered`. This outputs a Maple `piecewise` structure that gives an intuitive description of the layered CAD. Whenever a branch has been terminated a placeholder is given.

A simple example is given in Figure 7. This can be compared to Figure 8 which shows the equivalent full CAD in piecewise format. We see that the layered CAD displayed omits all information regarding the cylinder above $y = 0$ as this is unnecessary for a 1-layered CAD, along with the sections $x = 1 - y$ when $y > 0$ or $y < 0$.

A.4. `(TTI)CADDistribution` **and** `(TTI)CADNormDist`**.** ,
The package contains four analytical tools to generate the cell distributions of a Full CAD as used in Section 6. `CADDistribution` outputs a list with the number of cells for each dimension at its appropriate index in the list (regarding the list indices as starting at 0). `CADNormDist` returns a similar list but with each entry normed by dividing by the total number of cells to allow easy comparisons between problems. `TTICADDistribution` and `TTICADNormDist` perform the same tasks but for a TTICAD.

```
  > LD,RLD:=CADRecursiveLayeredLR({x+y-1,y},[x,y],[],[]):
  > CADDisplayLayered(LD);

{{ [regular_chain, [[-1, -1],[1, 1]]]       x < -y + 1
{{
{{            ["*********"]               branch = truncated       y < 0
{{
{{ [regular_chain, [[-1, -1],[3, 3]]]       -y + 1 < x
{
{                          ["*********"]                  branch = truncated
{
{{ [regular_chain, [[1, 1],[-1, -1]]]       x < -y + 1
{{
{{            ["*********"]               branch = truncated       0 < y
{{
{{ [regular_chain, [[1, 1],[1, 1]]]         -y + 1 < x
```

FIGURE 7. Usage of `CADDisplayLayered` for a piecewise layered CAD

```
> CADFull({x y-1,y},[x,y],method=McCallum,output=piecewise);

{{ [regular_chain, [[-1, -1],[1, 1]]]        x < -y + 1
{{
{{ [regular_chain, [[-1, -1],[2, 2]]]        x = -y + 1        y < 0
{{
{{ [regular_chain, [[-1, -1],[3, 3]]]        -y + 1 < x
{
{  { [regular_chain, [[0, 0],[0, 0]]]          x < 1
{  {
{  { [regular_chain, [[0, 0],[1, 1]]]          x = 1             y = 0
{  {
{  { [regular_chain, [[0, 0],[2, 2]]]          1 < x
{
{{ [regular_chain, [[1, 1],[-1, -1]]]        x < -y + 1
{{
{{  [regular_chain, [[1, 1],[0, 0]]]         x = -y + 1        0 < y
{{
{{  [regular_chain, [[1, 1],[1, 1]]]         -y + 1 < x
```

FIGURE 8. Full piecewise CAD

A.5. `TTICADOneLayered` **and** `TTICADNLayered`**.**
The ideas discussed in Section 7 are implemented within `LayeredCAD` with the folllowing five functions:

- `TTICADOneLayered(PHI,vars)`
- `TTICADTwoLayered(PHI,vars)`
- `TTICADNLayered(PHI,N,vars)`
- `TTICADDistribution(PHI,vars)`
- `TTICADNormDist(PHI,vars)`

These are built over the TTICAD procedures within the `ProjectionCAD` package and work in an almost identical way to the other `LayeredCAD` procedures. No recursive layered TTICAD algorithms are currently implemented - partly as TTICAD

works differently for the first projection (and therefore the final lifting) so recursion would be a bit trickier.

A.6. **Distribution of the `LayeredCAD` package.**
The `LayeredCAD` package is freely available alongside this paper or from the author's website: http://www.cs.bath.ac.uk/~djw42/triangular. Any corrections, suggestions or comments would be gratefully received and should be sent to the author on D.J.Wilson@bath.ac.uk.

References

[BBDP05] James C Beaumont, Russell J Bradford, James H Davenport, and Nalina Phisanbut, *Adherence is better than adjacency: computing the Riemann index using CAD*, Proc. ISSAC '05, July 2005.

[BD07] Christopher W Brown and James H Davenport, *The complexity of quantifier elimination and cylindrical algebraic decomposition*, Proc. ISSAC '07, July 2007.

[BDE+13] Russell J Bradford, James H Davenport, Matthew England, Scott McCallum, and David J Wilson, *Cylindrical Algebraic Decompositions for Boolean Combinations*, Proc. ISSAC '13, July 2013.

[BDEW13] Russell J Bradford, James H Davenport, Matthew England, and David J Wilson, *Optimising Problem Formulations for Cylindrical Algebraic Decomposition*, Proc. Calculemus '13 (part of CICM 2013), 2013.

[BDPB07] Russell J Bradford, James H Davenport, Nalina Phisanbut, and James C Beaumont, *Testing elementary function identities using CAD*, Applicable Algebra in ... **18** (2007), pp. 513–543.

[Bro13] Christopher W Brown, *Constructing a single open cell in a cylindrical algebraic decomposition*, Proc. ISSAC '13, July 2013.

[BH91] Bruno Buchberger and Hoon Hong, *Speeding-up Quantifier Elimination by Gröbner Bases*, RISC Report Series (1991), pp. 1–21.

[CDM+10] Changbo Chen, James H Davenport, John P May, Marc Moreno Maza, Bican Xia, and Rong Xiao, *Triangular Decomposition of Semi-algebraic Systems*, arXiv.org **cs.SC** (2010).

[CH91] George E Collins and Hoon Hong, *Partial Cylindrical Algebraic Decomposition for quantifier elimination*, Journal of Symbolic Computation **12** (1991), no. 3, 299–328.

[CMXY09] Changbo Chen, Marc Moreno Maza, Bican Xia, and Lu Yang, *Computing Cylindrical Algebraic Decomposition via Triangular Decomposition*, Proc. ISSAC '09, July 2009, pp. 95–102.

[Col75] George E Collins, *Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition*, Automata Theory and Formal Languages 2nd GI ... (1975).

[Dav86] James H Davenport, *A "Piano Movers" Problem*, ACM SIGSAM Bulletin (1986).

[DSS04] Andreas Dolzmann, Andreas Seidl, and Thomas Sturm, *Efficient Projection Orders for CAD*, Proc. ISSAC 2004, July 2004, pp. 111–118.

[Eng13a] Matthew England, *An implementation of CAD in Maple utilising McCallum projection*, Department of Computer Science Technical Report series 2013-02, University of Bath. Available at `http://opus.bath.ac.uk/33180/`, 2013.

[Eng13b] Matthew England, *An implementation of cad in maple utilising problem formulation, equational constraints and truth-table invariance*, Department of Computer Science Technical Report series 2013-04, University of Bath. Available at `http://opus.bath.ac.uk/35636/`, 2013.

[EBDW13] Matthew England, Russell J Bradford, James H Davenport, and David J Wilson, *Understanding branch cuts of expressions*, Proc. MKM '13 (part of CICM 2013), 2013.

[Fau00] Jean-Charles Faugere, *How my computer find all the solutions of Cyclic–9*, Tech. report, 2000.

[Hon91] Hoon Hong, *Comparison of Several Decision Algorithms for the Existential Theory of the Reals*, Tech. report, 1991.

[McC93] Scott McCallum, *Solving Polynomial Strict Inequalities Using Cylindrical Algebraic Decomposition*, The Computer Journal **36** (1993), no. 5, pp. 432–438.

[McC98] ———, *An Improved Projection Operation for Cylindrical Algebraic Decomposition*, Quantifier Elimination and Cylindrical Algebraic Decomposition (Bob F Caviness and Jeremy R Johnson, eds.), 1998, pp. 242–268.

[McC99] ———, *On Projection in CAD-Based Quantifier Elimination with Equational Constraint*, Proc. ISSAC '99, July 1999, pp. 145–149.

[Str94] Adam Strzeboński, *An Algorithm for Systems of Strong Polynomial Inequalities*, Math. J. **4** (1994), pp. 74–77.

[Str00] Adam Strzeboński, *Solving systems of strict polynomial inequalities*, Journal of Symbolic Computation **29** (2000), no. 3.

[WBD12a] David John Wilson, Russell J Bradford, and James H Davenport, *A repository for CAD Examples*, ACM Communications in ... (2012).

[WBD12b] ———, *Speeding up Cylindrical Algebraic Decomposition by Gröbner Bases*, Lecture Notes in Computer Science **7362** (2012), no. Chapter 19, pp. 280–294.

[Wil04] Stephen Willard, *General Topology*, 1 ed., Dover Publications, 2004.