Problem Formulation for Truth-Table Invariant
Cylindrical Algebraic Decomposition by
Incremental Triangular Decomposition

Matthew England (The University of Bath)

Joint work with: R. Bradford, J.H. Davenport & D. Wilson (University of Bath), C. Chen (CIGIT, Chinese Academy of Sciences), and M. Moreno Maza (Western University).

Conferences on Intelligent Computer Mathematics
Calculemus Track
University of Coimbra, Coimbra, Portugal
July 7-11 2014
Problem Formulation for Truth-Table Invariant Cylindrical Algebraic Decomposition by Incremental Triangular Decomposition

1 Background material
   • CAD
   • RC-TTICAD

2 Problem formulation
   • Constraint ordering
   • Other issues of formulation
Problem Formulation for Truth-Table Invariant
Cylindrical Algebraic Decomposition
by Incremental Triangular Decomposition

1. **Background material**
   - CAD
   - RC-TTICAD

2. **Problem formulation**
   - Constraint ordering
   - Other issues of formulation

England et al.  Problem Formulation for RC-TTICAD
Problem Formulation for Truth-Table Invariant
Cylindrical Algebraic Decomposition
by Incremental Triangular Decomposition

1 Background material
   - CAD
   - RC-TTICAD

2 Problem formulation
   - Constraint ordering
   - Other issues of formulation
Problem Formulation for Truth-Table Invariant Cylindrical Algebraic Decomposition by *Incremental Triangular Decomposition*

1. **Background material**
   - CAD
   - RC-TTICAD

2. **Problem formulation**
   - Constraint ordering
   - Other issues of formulation
Problem Formulation for Truth-Table Invariant Cylindrical Algebraic Decomposition by Incremental Triangular Decomposition

1 Background material
   - CAD
   - RC-TTICAD

2 Problem formulation
   - Constraint ordering
   - Other issues of formulation
Outline

1. Background material
   - CAD
   - RC-TTICAD

2. Problem formulation
   - Constraint ordering
   - Other issues of formulation
A **Cylindrical Algebraic Decomposition (CAD)** is:

- a decomposition meaning a partition of $\mathbb{R}^n$ into connected subsets called **cells**;
- (semi)-algebraic meaning that each cell can be defined by a sequence of polynomial equations and inequations.
- cylindrical meaning the cells are arranged in a useful manner - their projections are either equal or disjoint.
Example - Cylindrical Algebraic Decomposition

A CAD of $\mathbb{R}^2$ is given by the following collections of 13 cells:

$$
[ x < -1, y = y ],
[ x = -1, y < 0 ], [ x = -1, y = 0 ], [ x = -1, y > 0 ],
[-1 < x < 1, y^2 + x^2 - 1 > 0, y > 0 ],
[-1 < x < 1, y^2 + x^2 - 1 = 0, y > 0 ],
[-1 < x < 1, y^2 + x^2 - 1 < 0 ],
[-1 < x < 1, y^2 + x^2 - 1 = 0, y < 0 ],
[-1 < x < 1, y^2 + x^2 - 1 < 0 ],
[ x = 1, y < 0 ], [ x = 1, y = 0 ], [ x = 1, y > 0 ],
[ x > 1, y = y ]
$$
A CAD of $\mathbb{R}^2$ is given by the following collections of 13 cells:

- **$x < -1$**
  - $[x < -1, y = y]$, 
  - $[-1 < x < 1, y^2 + x^2 - 1 > 0, y > 0]$, 
  - $[-1 < x < 1, y^2 + x^2 - 1 = 0, y > 0]$, 
- **$x = -1$**
  - $[x = -1, y < 0]$, 
  - $[x = -1, y = 0]$, 
  - $[x = -1, y > 0]$, 
- **$-1 < x < 1$**
  - $[-1 < x < 1, y^2 + x^2 - 1 < 0]$, 
  - $[-1 < x < 1, y^2 + x^2 - 1 = 0, y < 0]$, 
  - $[-1 < x < 1, y^2 + x^2 - 1 < 0, y < 0]$, 
- **$x = 1$**
  - $[x = 1, y < 0]$, 
  - $[x = 1, y = 0]$, 
  - $[x = 1, y > 0]$, 
- **$x > 1$**
  - $[x > 1, y = y]$
Traditionally a CAD is produced from a set of polynomials such that each polynomial has constant sign (positive, zero or negative) in each cell. Such a CAD is said to be sign-invariant.

The example from the previous slide was a sign-invariant CAD for the polynomial $x^2 + y^2 - 1$. 
Sign-invariance

Traditionally a CAD is produced from a set of polynomials such that each polynomial has constant sign (positive, zero or negative) in each cell. Such a CAD is said to be sign-invariant.

The example from the previous slide was a sign-invariant CAD for the polynomial $x^2 + y^2 - 1$. 

England et al. Problem Formulation for RC-TTICAD
Traditionally a CAD is produced from a set of polynomials such that each polynomial has constant sign (positive, zero or negative) in each cell. Such a CAD is said to be sign-invariant.

The example from the previous slide was a sign-invariant CAD for the polynomial $x^2 + y^2 - 1$. 
Traditionally a CAD is produced from a set of polynomials such that each polynomial has constant sign (positive, zero or negative) in each cell. Such a CAD is said to be sign-invariant.

The example from the previous slide was a sign-invariant CAD for the polynomial $x^2 + y^2 - 1$. 
Traditionally a CAD is produced from a set of polynomials such that each polynomial has constant sign (positive, zero or negative) in each cell. Such a CAD is said to be **sign-invariant**.

The example from the previous slide was a sign-invariant CAD for the polynomial $x^2 + y^2 \ - \ 1$. 
Traditionally a CAD is produced from a set of polynomials such that each polynomial has constant sign (positive, zero or negative) in each cell. Such a CAD is said to be **sign-invariant**.

The example from the previous slide was a sign-invariant CAD for the polynomial $x^2 + y^2 - 1$.

Sign-invariance means we need only test one sample point per cell to determine behaviour of the polynomials. Various applications: quantifier elimination, optimisation, theorem proving, ...
Most CAD algorithms work by a process of:

- **Projection**: to derive a set of polynomials from the input which can define the decomposition
Most CAD algorithms work by a process of:

- **Projection**: to derive a set of polynomials from the input which can define the decomposition
- **Lifting** to incrementally build CADs by dimension.
Most CAD algorithms work by a process of:

- **Projection**: to derive a set of polynomials from the input which can define the decomposition
- **Lifting**: to incrementally build CADs by dimension.

![Graph depicting projection and lifting](image)
Most CAD algorithms work by a process of:

- **Projection**: to derive a set of polynomials from the input which can define the decomposition
- **Lifting**: to incrementally build CADs by dimension.
Most CAD algorithms work by a process of:

- **Projection**: to derive a set of polynomials from the input which can define the decomposition
- **Lifting**: to incrementally build CADs by dimension.
Most CAD algorithms work by a process of:

- **Projection**: to derive a set of polynomials from the input which can define the decomposition
- **Lifting**: to incrementally build CADs by dimension.
Most CAD algorithms work by a process of:

- **Projection**: to derive a set of polynomials from the input which can define the decomposition
- **Lifting**: to incrementally build CADs by dimension.
Given a sequence of quantifier free formulae (QFFs) we define a truth table invariant CAD (TTICAD) as a CAD such that each formula has constant Boolean truth value on each cell.
Truth table invariance

Given a sequence of quantifier free formulae (QFFs) we define a truth table invariant CAD (TTICAD) as a CAD such that each formula has constant Boolean truth value on each cell.

ISSAC 2013: Bath team + Scott McCallum derived a reduced projection operator to build TTICADs for a sequence of formulae which each have an equational constraint (EC): an equation logically implied by the formula.

2014 (submitted): Extended to any sequence of formulae, with savings if at least one has an EC. Plus further savings identified in the lifting phase from presence of EC.
< 2009  All CAD research broadly within Collin’s projection and lifting framework.

\[ \mathbb{R}^n \rightarrow \mathbb{R}^{n-1} \rightarrow \mathbb{R}^1 \]

Projection

\[ \mathbb{R}^n \uparrow \leftarrow \mathbb{R}^{n-1} \leftarrow \mathbb{R}^1 \]

Lifting

England et al.
A new framework for CAD

< 2009: All CAD research broadly within Collin’s projection and lifting framework.
ISSAC 2009: Chen, Moreno Maza, Xia & Yang give new approach.
A new framework for CAD

< 2009: All CAD research broadly within Collin’s projection and lifting framework.

ISSAC 2009: Chen, Moreno Maza, Xia & Yang give new approach.
RC-CAD starts with a complex cylindrical decomposition (CCD). The tree below represents a sign-invariant CCD for $p := x^2 + bx + c$ under variable ordering $c \prec b \prec x$.

The key advantage is case distinction: the polynomial $b$ is not sign-invariant for the whole decomposition, only when required.
RC–CAD starts with a complex cylindrical decomposition (CCD). The tree below represents a sign-invariant CCD for $p := x^2 + bx + c$ under variable ordering $c \prec b \prec x$.

The key advantage is case distinction: the polynomial $b$ is not sign-invariant for the whole decomposition, only when required.
CCDs may be calculated using the theory of triangular decomposition by regular chains:

**ISSAC 2009:** Utilising existing algorithms in the **RegularChains** Library.

**ASCM 2012:** Using custom built algorithms, now part of the **RegularChains** Library.

One practical difference is that the latter algorithms work **incrementally**: refining a tree one polynomial at a time.
CCDs may be calculated using the theory of triangular decomposition by regular chains:

**ISSAC 2009:** Utilising existing algorithms in the `RegularChains` Library.

**ASCM 2012:** Using custom built algorithms, now part of the `RegularChains` Library.

One practical difference is that the latter algorithms work incrementally: refining a tree one polynomial at a time. This allows for the easy implementation of **simplification in the presence of equational constraints:** If an EC is not satisfied on a branch then no need for further refinement.
CASC 2014: Algorithm presented to produce TTICADs in the RC-CAD framework through incremental triangular decomposition = RC-TTICAD.
RC-TTICAD

CASC 2014: Algorithm presented to produce TTICADs in the RC-CAD framework through incremental triangular decomposition = RC-TTICAD.

Advantages over PL-TTICAD:

- The case distinction built into the RC-CAD framework;
- Can take advantage of multiple ECs in a formula easily;
- No theoretical failure due to well-orientedness conditions.

Experimental results in CASC 2014 show this approach to be competitive with the state of the art in CAD.
CASC 2014: Algorithm presented to produce TTICADs in the RC–CAD framework through incremental triangular decomposition = RC-TTICAD.

Advantages over PL-TTICAD:
- The case distinction built into the RC–CAD framework;
- Can take advantage of multiple ECs in a formula easily;
- No theoretical failure due to well-orientedness conditions.

Experimental results in CASC 2014 show this approach to be competitive with the state of the art in CAD. However, the best implementation varied greatly by example. We now try to optimise both the implementation of RC-TTICAD, but also how we formulate problems for it.
Outline

1. **Background material**
   - CAD
   - RC-TTICAD

2. **Problem formulation**
   - Constraint ordering
   - Other issues of formulation
Problem formulation for CAD

Any CAD is produced relative to an ordering on the variables (defining which projections the cylindricity is defined against). Depending on the application we may have the freedom to choose.

Consider a sign-invariant CAD for $f := (x - 1)(y^2 + 1) - 1$. Then there are two possible minimal CADs:

- 3 cells (2 full dimensional)
- 11 cells (5 full dimensional)
Problem formulation for CAD

Any CAD is produced relative to an ordering on the variables (defining which projections the cylindricity is defined against). Depending on the application we may have the freedom to choose.

Consider a sign-invariant CAD for \( f := (x - 1)(y^2 + 1) - 1 \). Then there are two possible minimal CADs:

- 3 cells (2 full dimensional)
- 11 cells (5 full dimensional)

The choice can change the tractability of a problem. There exist various heuristics to help. Some of these will be presented next along with a new approach utilising machine learning.
We focus on a choice particular to RC-TTICAD: the order in which constraints are presented to the algorithm.
Choosing the order of constraints

We focus on a choice particular to RC-TTICAD: the order in which constraints are presented to the algorithm.

- The algorithm processes all ECs first;
- It is logical to process ECs from the same formula one after the another;
- It makes no difference in what order you process non-ECs.

But there are open questions:

Q1: In what order to process the formulae?
Q2: In what order order to process the ECs within each formulae?
Choosing the order of constraints

We focus on a choice particular to RC-TTICAD: the order in which constraints are presented to the algorithm.

- The algorithm processes all ECs first;
- It is logical to process ECs from the same formula one after the another;
- It makes no difference in what order you process non-ECs.

But there are open questions:

**Q1:** In what order to process the formulae?

**Q2:** In what order order to process the ECs within each formulae?

We use $\rightarrow$ to denote the order constraints are processed. Existing heuristics for CAD do not discriminate between these orderings.
Assume the ordering $x \prec y$ and consider

\[ f_1 := x^2 + y^2 - 1, \quad f_2 := 2y^2 - x, \quad \phi_1 := f_1 = 0 \land f_2 = 0, \]
\[ f_3 := (x - 5)^2 + (y - 1)^2 - 1, \quad \phi_2 := f_3 = 0. \]
Assume the ordering $x \prec y$ and consider

$$f_1 := x^2 + y^2 - 1, \quad f_2 := 2y^2 - x,$$
$$f_3 := (x - 5)^2 + (y - 1)^2 - 1,$$
$$\phi_1 := f_1 = 0 \land f_2 = 0, \quad \phi_2 := f_3 = 0.$$
Assume the ordering \( x \prec y \) and consider

\[
\begin{align*}
  f_1 & := x^2 + y^2 - 1, \\
  f_2 & := 2y^2 - x, \\
  f_3 & := (x - 5)^2 + (y - 1)^2 - 1,
\end{align*}
\]

\[
\begin{align*}
  \phi_1 & := f_1 = 0 \land f_2 = 0, \\
  \phi_2 & := f_3 = 0.
\end{align*}
\]
Our implementation of RC-TTICAD computed:

\( \phi_1 \rightarrow \phi_2 \text{ and } f_1 \rightarrow f_2 \): 37 cells in 0.095 seconds.
\( \phi_1 \rightarrow \phi_2 \text{ and } f_2 \rightarrow f_1 \): 81 cells in 0.118 seconds.

\( \phi_2 \rightarrow \phi_1 \text{ and } f_1 \rightarrow f_2 \): 25 cells in 0.087 seconds.
\( \phi_2 \rightarrow \phi_1 \text{ and } f_2 \rightarrow f_1 \): 43 cells in 0.089 seconds

- Both constraint ordering and EC ordering made substantial differences to output size (and computation time);
Our implementation of RC-TTICAD computed:

\[ \phi_1 \rightarrow \phi_2 \text{ and } f_1 \rightarrow f_2: \text{ 37 cells in 0.095 seconds.} \]

\[ \phi_1 \rightarrow \phi_2 \text{ and } f_2 \rightarrow f_1: \text{ 81 cells in 0.118 seconds.} \]

\[ \phi_2 \rightarrow \phi_1 \text{ and } f_1 \rightarrow f_2: \text{ 25 cells in 0.087 seconds.} \]

\[ \phi_2 \rightarrow \phi_1 \text{ and } f_2 \rightarrow f_1: \text{ 43 cells in 0.089 seconds.} \]

- Both constraint ordering and EC ordering made substantial differences to output size (and computation time);
- It was always superior to process \( f_1 \rightarrow f_2 \);
Our implementation of RC-TTICAD computed:

- $\phi_1 \rightarrow \phi_2$ and $f_1 \rightarrow f_2$: 37 cells in 0.095 seconds.
- $\phi_1 \rightarrow \phi_2$ and $f_2 \rightarrow f_1$: 81 cells in 0.118 seconds.
- $\phi_2 \rightarrow \phi_1$ and $f_1 \rightarrow f_2$: 25 cells in 0.087 seconds.
- $\phi_2 \rightarrow \phi_1$ and $f_2 \rightarrow f_1$: 43 cells in 0.089 seconds.

- Both constraint ordering and EC ordering made substantial differences to output size (and computation time);
- It was always superior to process $f_1 \rightarrow f_2$;
- It was always superior to process $\phi_2 \rightarrow \phi_1$. 
Figures on the top have $\phi_1 \rightarrow \phi_2$ and on the bottom $\phi_2 \rightarrow \phi_1$. Figures on the left have $f_1 \rightarrow f_2$ and on the right $f_2 \rightarrow f_1$. 
Lessons for EC ordering choice

Illustrated by example and verified by algorithm specification, the output of RC-TTICAD is

- SI for discriminant of first EC in each formula;
- SI for resultants of pairs of first ECs in each formula;

So identify these polynomials as the constraint ordering set and use a measure on it as a heuristic for making the choice of EC ordering.
Lessons for EC ordering choice

Illustrated by example and verified by algorithm specification, the output of RC-TTICAD is

- SI for discriminant of first EC in each formula;
- SI for resultants of pairs of first ECs in each formula;

So identify these polynomials as the constraint ordering set and use a measure on it as a heuristic for making the choice of EC ordering.

Previous research would suggest sotd as the measure, but this measure of sparsity it only suitable for entire projection sets, not subsets. Instead we suggest sum of degrees in the main variable. (Illustrated by 2012 x-axis ellipse problem in the paper). Refer to this as Heuristic 1.
Illustrated by example and verified by algorithm specification, the output of RC-TTICAD is

- SI for the first polynomial considered.

So beneficial to place a formulae first if it has only one EC. But what to do otherwise?
Lessons for formula ordering choice

Illustrated by example and verified by algorithm specification, the output of RC-TTICAD is

- SI for the first polynomial considered.

So beneficial to place a formulae first if it has only one EC. But what to do otherwise?

**Heuristic 2:** Construct CCDs; extract the polynomials within; refine to a CAD the one with the lowest sum of degree of polynomials (each taken in the main variable of the polynomial).
Illustrated by example and verified by algorithm specification, the output of RC-TTICAD is

- SI for the first polynomial considered.

So beneficial to place a formulae first if it has only one EC. But what to do otherwise?

**Heuristic 2:** Construct CCDs; extract the polynomials within; refine to a CAD the one with the lowest sum of degree of polynomials (each taken in the main variable of the polynomial).

**Heuristic 3:** Use Heuristic 1 and split ties with Heuristic 2.
Generated 100 random systems consisting of two formulae, each with two ECs and one non-EC, defined by random sparse polynomials in three variables. For each there are $2 \times 2 \times 2 = 8$ choices of constraint ordering.
Experiments on the heuristics

Generated 100 random systems consisting of two formulae, each with two ECs and one non-EC, defined by random sparse polynomials in three variables. For each there are $2 \times 2 \times 2 = 8$ choices of constraint ordering.

Average TTICAD had 2018 cells, computed in 37 seconds. But median CAD had 1554 cells, computed in 6.1 seconds. A lot of variance both between and within individual problems.

For each problem we compared how the heuristics’ choices compared to the problem average.
### Experimental results

<table>
<thead>
<tr>
<th>H</th>
<th>Cell Count</th>
<th>Saving</th>
<th>% Saving</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1589.67</td>
<td>428.61</td>
<td>26.73</td>
</tr>
<tr>
<td>2</td>
<td>1209.10</td>
<td>809.18</td>
<td>47.70</td>
</tr>
<tr>
<td>3</td>
<td>1307.63</td>
<td>710.65</td>
<td>40.97</td>
</tr>
</tbody>
</table>

England et al. Problem Formulation for RC-TTICAD
### Experimental results

<table>
<thead>
<tr>
<th>H</th>
<th>Cell Count</th>
<th>Saving</th>
<th>% Saving</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1589.67</td>
<td>428.61</td>
<td>26.73</td>
</tr>
<tr>
<td>2</td>
<td>1209.10</td>
<td>809.18</td>
<td>47.70</td>
</tr>
<tr>
<td>3</td>
<td>1307.63</td>
<td>710.65</td>
<td>40.97</td>
</tr>
</tbody>
</table>

England et al. Problem Formulation for RC-TTICAD
## Experimental results

<table>
<thead>
<tr>
<th>H</th>
<th>Cell Count</th>
<th>Saving</th>
<th>% Saving</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1589.67</td>
<td>428.61</td>
<td>26.73</td>
</tr>
<tr>
<td>2</td>
<td>1209.10</td>
<td>809.18</td>
<td>47.70</td>
</tr>
<tr>
<td>3</td>
<td>1307.63</td>
<td>710.65</td>
<td>40.97</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>H</th>
<th>Time</th>
<th>Saving</th>
<th>% Saving</th>
<th>Net Saving</th>
<th>% Net Saving</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14.48</td>
<td>22.02</td>
<td>37.17</td>
<td>22.01</td>
<td>37.12</td>
</tr>
<tr>
<td>2</td>
<td>9.02</td>
<td>27.47</td>
<td>49.45</td>
<td>-150.59</td>
<td>-215.31</td>
</tr>
<tr>
<td>3</td>
<td>9.42</td>
<td>27.08</td>
<td>43.84</td>
<td>-20.02</td>
<td>0.77</td>
</tr>
</tbody>
</table>
### Experimental results

<table>
<thead>
<tr>
<th>H</th>
<th>Cell Count</th>
<th>Saving</th>
<th>% Saving</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1589.67</td>
<td>428.61</td>
<td>26.73</td>
</tr>
<tr>
<td>2</td>
<td>1209.10</td>
<td>809.18</td>
<td>47.70</td>
</tr>
<tr>
<td>3</td>
<td>1307.63</td>
<td>710.65</td>
<td>40.97</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>H</th>
<th>Time</th>
<th>Saving</th>
<th>% Saving</th>
<th>Net Saving</th>
<th>% Net Saving</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14.48</td>
<td>22.02</td>
<td>37.17</td>
<td>22.01</td>
<td>37.12</td>
</tr>
<tr>
<td>2</td>
<td>9.02</td>
<td>27.47</td>
<td>49.45</td>
<td>-150.59</td>
<td>-215.31</td>
</tr>
<tr>
<td>3</td>
<td>9.42</td>
<td>27.08</td>
<td>43.84</td>
<td>-20.02</td>
<td>0.77</td>
</tr>
</tbody>
</table>

England et al. Problem Formulation for RC-TTICAD
Experimental results

<table>
<thead>
<tr>
<th>H</th>
<th>Cell Count</th>
<th>Saving</th>
<th>% Saving</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1589.67</td>
<td>428.61</td>
<td>26.73</td>
</tr>
<tr>
<td>2</td>
<td>1209.10</td>
<td>809.18</td>
<td>47.70</td>
</tr>
<tr>
<td>3</td>
<td>1307.63</td>
<td>710.65</td>
<td>40.97</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>H</th>
<th>Time</th>
<th>Saving</th>
<th>% Saving</th>
<th>Net Saving</th>
<th>% Net Saving</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14.48</td>
<td>22.02</td>
<td>37.17</td>
<td>22.01</td>
<td>37.12</td>
</tr>
<tr>
<td>2</td>
<td>9.02</td>
<td>27.47</td>
<td>49.45</td>
<td>-150.59</td>
<td>-215.31</td>
</tr>
<tr>
<td>3</td>
<td>9.42</td>
<td>27.08</td>
<td>43.84</td>
<td>-20.02</td>
<td>0.77</td>
</tr>
</tbody>
</table>

England et al. Problem Formulation for RC-TTICAD
### Experimental results

<table>
<thead>
<tr>
<th>H</th>
<th>Cell Count</th>
<th>Saving</th>
<th>% Saving</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1589.67</td>
<td>428.61</td>
<td>26.73</td>
</tr>
<tr>
<td>2</td>
<td>1209.10</td>
<td>809.18</td>
<td>47.70</td>
</tr>
<tr>
<td>3</td>
<td>1307.63</td>
<td>710.65</td>
<td>40.97</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>H</th>
<th>Time</th>
<th>Saving</th>
<th>% Saving</th>
<th>Net Saving</th>
<th>% Net Saving</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14.48</td>
<td>22.02</td>
<td>37.17</td>
<td>22.01</td>
<td>37.12</td>
</tr>
<tr>
<td>2</td>
<td>9.02</td>
<td>27.47</td>
<td>49.45</td>
<td>-150.59</td>
<td>-215.31</td>
</tr>
<tr>
<td>3</td>
<td>9.42</td>
<td>27.08</td>
<td>43.84</td>
<td>-20.02</td>
<td>0.77</td>
</tr>
</tbody>
</table>
Conclusions

We can see situations where all three heuristics could be useful:

**Use Heuristic 1** if lowest computation time is prioritised, for example if many CADs must be computed or this is just a small step in a larger calculation.

**Use Heuristic 2** if lowest cell count is prioritised, for example if only one CAD must be computed but then much work will be performed with its cells.
We can see situations where all three heuristics could be of use:

**Use Heuristic 1** if lowest computation time is prioritised, for example if many CADs must be computed or this is just a small step in a larger calculation.

**Use Heuristic 2** if lowest cell count is prioritised, for example if only one CAD must be computed but then much work will be performed with its cells.

**Use Heuristic 3** for a mixed approach, for example if a low cell count is required but the problem size makes Heuristic 2 infeasible.
We can see situations where all three heuristics could be of use:

**Use Heuristic 1** if lowest computation time is prioritised, for example if many CADs must be computed or this is just a small step in a larger calculation.

**Use Heuristic 2** if lowest cell count is prioritised, for example if only one CAD must be computed but then much work will be performed with its cells.

**Use Heuristic 3** for a mixed approach, for example if a low cell count is required but the problem size makes Heuristic 2 infeasible.

Heuristic 1 likely to become part of the default implementation.
In CICM 2013 the Bath team looked at issues of problem formulation for PL-TTICAD, revisited now for RC-TTICAD:

**EC designation:** This problem is analogous to picking which EC to process first in RC-TTICAD.

**Composing sub-formula:** This issue no longer needs to be considered since RC-TTICAD can use multiple ECs per formula. The best case is to always break into conjunctive sub-formulae.

**Another key issue:** Variable ordering: Recently we have developed a new heuristic more tuned to RC-TTICAD for picking this. But should we decide this before or after the constraint ordering?
In CICM 2013 the Bath team looked at issues of problem formulation for PL-TTICAD, revisited now for RC-TTICAD:

**EC designation:** This problem is analogous to picking which EC to process first in RC-TTICAD.

**Composing sub-formula:** This issue no longer needs to be considered since RC-TTICAD can use multiple ECs per formula. The best case is to always break into conjunctive sub-formulae.
In CICM 2013 the Bath team looked at issues of problem formulation for PL-TTICAD, revisited now for RC-TTICAD:

**EC designation:** This problem is analogous to picking which EC to process first in RC-TTICAD.

**Composing sub-formula:** This issue no longer needs to be considered since RC-TTICAD can use multiple ECs per formula. The best case is to always break into conjunctive sub-formulae.

Another key issue:

**Variable ordering:** Recently we have developed a new heuristic more tuned to RC-TTICAD for picking this.

But should we decide this before or after the constraint ordering?
This CICM paper examined issues of problem formulation for RC-TTICAD. The algorithms itself is presented in:


*Truth table invariant cylindrical algebraic decomposition by regular chains.*


Contact Details

M.England@bath.ac.uk
http://www.cs.bath.ac.uk/~me350/

Thanks for listening!