Constraint Programming

Propagation algorithms for specific constraints
Previously

- MAC (Maintaining Arc Consistency)
  - Combines search and enforcing global arc consistency
  - **Incremental**: smart about which arcs (constraints) are added to the queue following an assignment
  - After every assignment $x_i = d_i$, initialise the queue with the consequences of removing all but $d_i$ from $D_i$
  - Propagate these changes to re-establish global arc consistency.
Previously

- Exploiting properties of constraints
- e.g. functional constraints
  - Constraint $c(x_i, x_j)$ is functional wrt $D_i$:
  - For all $d_i$ in $D_i$ there exists at most one $d_j$ in $D_j$ such that $c(x_i = d_i, x_j = d_j)$ is satisfied.
  - Example: $x_i = x_j + c$
  - For each value of $x_i$, only need to check one value of $x_j$ to determine support
This Lecture

- Propagators for intensional non-binary constraints
  - Less-than
  - Lexicographic (dictionary) ordering of two sequences
  - AllDifferent
- All enforce GAC
Brief note about Triggers

- AC6 has list $L$ of removed domain values to be processed
- Contains pairs $\langle x_4, 2 \rangle$
- Processes all arcs incident to $x_4$
- In a modern solver, constraints ask to be notified of events (they place triggers), e.g.:
  - Upper/lower bound moved on variable $x_i$
  - Value $a$ removed from $D_i$
  - Any change to $D_i$
Less-than

- \( x_i \leq x_j + c \)
- Places triggers on bounds
  - Which bounds?
Less-than

- $x_i \leq x_j + c$

- Places triggers on bounds
  - Upper bound of $x_j$
    - What happens when this fires?
  - Lower bound of $x_i$
    - What happens when this fires?
Less-than

- $x_i \leq x_j + c$
- Places triggers on bounds
  - Upper bound of $x_j$
    - $x_i \leq UB(x_j) + c$
  - Lower bound of $x_i$
    - $x_j \geq LB(x_i) - c$
- After updating, all values in $D_i$ are supported by $UB(x_j)$ – Monotonic
Less-than

- \( x_i \leq x_j + c \)
- Uses bound triggers to exploit **monotonicity**
  - Propagator is never called unless a supporting value is removed
- Uses monotonicity again to remove multiple values at once
Lex ordering

- Lexicographic (dictionary) ordering of two vectors of variables.
- $\text{lexleq}: [x_1, x_2, \ldots, x_n] \leq_{\text{lex}} [y_1, y_2, \ldots, y_n]$
- $x_1 \leq y_1$
- if $x_1 = y_1$ then $x_2 \leq y_2$
- For all $i < n$, if $[x_1, \ldots, x_i] = [y_1, \ldots, y_i]$ then $x_{i+1} \leq y_{i+1}$
- $\text{lexless}: [x_1, x_2, \ldots, x_n] <_{\text{lex}} [y_1, y_2, \ldots, y_n]$
Lexicographic Ordering: A Filtering Algorithm

A new family of global constraints

Linear time complexity

Ensures that a pair of vectors of variables are lexicographically ordered.

\[
\begin{array}{cccc}
0 & 1 & 4 & 2 \\
\end{array}
\leq_{\text{lex}}

\begin{array}{cccc}
2 & 9 & 8 & 7 \\
\end{array}
\]
Motivation: Symmetry

Symmetry: transformation of an entity that preserves the properties of the entity

Example:

![Checkerboard pattern with a 180° rotation](image)
Motivation: Symmetry

Frequently occurs

Combinatorial problems like covering arrays

*Rows and columns can be permuted*

Messy real world problems like nurse rostering

*Nurses with same skills can be swapped*
Motivation

- Many problems can be modelled by matrices of decision variables.
- E.g. Combinatorial Problems
  - Balanced Incomplete Block Design.
- Configuration Problems:
  - Rack Configuration.
- Design Problems:
  - Steel Mill Slab Design.
Motivation

An important class of symmetries in CP matrices of decision variables:
rows/columns represent indistinguishable objects, hence symmetric.

Rows and columns can be permuted without affecting satisfiability.

Encountered frequently.
Example: Sports Scheduling

Schedule games between \( n \) teams over \( n-1 \) weeks
Each week is divided into \( n/2 \) periods
Each period has 2 slots: home and away
Find a schedule such that

- every team plays exactly once a week
- every team plays against every other team
- every team plays at most twice in the same period over the tournament
Example: Sport Scheduling

We need a table of meetings!

<table>
<thead>
<tr>
<th></th>
<th>Week 1</th>
<th>Week 2</th>
<th>Week 3</th>
<th>Week 4</th>
<th>Week 5</th>
<th>Week 6</th>
<th>Week 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Period1</td>
<td>0 vs 1</td>
<td>0 vs 2</td>
<td>4 vs 7</td>
<td>3 vs 6</td>
<td>3 vs 7</td>
<td>1 vs 5</td>
<td>2 vs 4</td>
</tr>
<tr>
<td>Period2</td>
<td>2 vs 3</td>
<td>1 vs 7</td>
<td>0 vs 3</td>
<td>5 vs 7</td>
<td>1 vs 4</td>
<td>0 vs 6</td>
<td>5 vs 6</td>
</tr>
<tr>
<td>Period3</td>
<td>4 vs 5</td>
<td>3 vs 5</td>
<td>1 vs 6</td>
<td>0 vs 4</td>
<td>2 vs 6</td>
<td>2 vs 7</td>
<td>0 vs 7</td>
</tr>
<tr>
<td>Period4</td>
<td>6 vs 7</td>
<td>4 vs 6</td>
<td>2 vs 5</td>
<td>1 vs 2</td>
<td>0 vs 5</td>
<td>3 vs 4</td>
<td>1 vs 3</td>
</tr>
</tbody>
</table>
Example: Sport Scheduling

We need a table of meetings!

<table>
<thead>
<tr>
<th></th>
<th>Week1</th>
<th>Week 2</th>
<th>Week 3</th>
<th>Week 4</th>
<th>Week 5</th>
<th>Week 6</th>
<th>Week 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Period1</td>
<td>0 vs 1</td>
<td>0 vs 2</td>
<td>4 vs 7</td>
<td>3 vs 6</td>
<td>3 vs 7</td>
<td>1 vs 5</td>
<td>2 vs 4</td>
</tr>
<tr>
<td>Period2</td>
<td>2 vs 3</td>
<td>1 vs 7</td>
<td>0 vs 3</td>
<td>5 vs 7</td>
<td>1 vs 4</td>
<td>0 vs 6</td>
<td>5 vs 6</td>
</tr>
<tr>
<td>Period3</td>
<td>4 vs 5</td>
<td>3 vs 5</td>
<td>1 vs 6</td>
<td>0 vs 4</td>
<td>2 vs 6</td>
<td>2 vs 7</td>
<td>0 vs 7</td>
</tr>
<tr>
<td>Period4</td>
<td>6 vs 7</td>
<td>4 vs 6</td>
<td>2 vs 5</td>
<td>1 vs 2</td>
<td>0 vs 5</td>
<td>3 vs 4</td>
<td>1 vs 3</td>
</tr>
</tbody>
</table>

- Weeks are indistinguishable
- Periods are indistinguishable
Example: Sport Scheduling

- Weeks are indistinguishable
- Periods are indistinguishable

<table>
<thead>
<tr>
<th></th>
<th>Week 1</th>
<th>Week 2</th>
<th>Week 3</th>
<th>Week 4</th>
<th>Week 5</th>
<th>Week 6</th>
<th>Week 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Period 1</td>
<td>0 vs 1</td>
<td>0 vs 2</td>
<td>4 vs 7</td>
<td>3 vs 6</td>
<td>3 vs 7</td>
<td>1 vs 5</td>
<td>2 vs 4</td>
</tr>
<tr>
<td>Period 2</td>
<td>2 vs 3</td>
<td>1 vs 7</td>
<td>0 vs 3</td>
<td>5 vs 7</td>
<td>1 vs 4</td>
<td>0 vs 6</td>
<td>5 vs 6</td>
</tr>
<tr>
<td>Period 3</td>
<td>4 vs 5</td>
<td>3 vs 5</td>
<td>1 vs 6</td>
<td>0 vs 4</td>
<td>2 vs 6</td>
<td>2 vs 7</td>
<td>0 vs 7</td>
</tr>
<tr>
<td>Period 4</td>
<td>6 vs 7</td>
<td>4 vs 6</td>
<td>2 vs 5</td>
<td>1 vs 2</td>
<td>0 vs 5</td>
<td>3 vs 4</td>
<td>1 vs 3</td>
</tr>
</tbody>
</table>
## Example: Sport Scheduling

- Weeks are indistinguishable
- Periods are indistinguishable

<table>
<thead>
<tr>
<th></th>
<th>Week 1</th>
<th>Week 2</th>
<th>Week 3</th>
<th>Week 4</th>
<th>Week 5</th>
<th>Week 6</th>
<th>Week 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Period 1</td>
<td>0 vs 1</td>
<td>3 vs 7</td>
<td>4 vs 7</td>
<td>3 vs 6</td>
<td>0 vs 2</td>
<td>1 vs 5</td>
<td>2 vs 4</td>
</tr>
<tr>
<td>Period 2</td>
<td>2 vs 3</td>
<td>1 vs 4</td>
<td>0 vs 3</td>
<td>5 vs 7</td>
<td>1 vs 7</td>
<td>0 vs 6</td>
<td>5 vs 6</td>
</tr>
<tr>
<td>Period 3</td>
<td>4 vs 5</td>
<td>2 vs 6</td>
<td>1 vs 6</td>
<td>0 vs 4</td>
<td>3 vs 5</td>
<td>2 vs 7</td>
<td>0 vs 7</td>
</tr>
<tr>
<td>Period 4</td>
<td>6 vs 7</td>
<td>0 vs 5</td>
<td>2 vs 5</td>
<td>1 vs 2</td>
<td>4 vs 6</td>
<td>3 vs 4</td>
<td>1 vs 3</td>
</tr>
</tbody>
</table>
Example: Bin Packing

Consider 2 identical bins:

A

B
Example

Consider 2 identical bins:

- We must pack 6 items:

  
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

  A   B
Example

Here is one solution:

A

5

3

1

B

6

4

2
Example

Here is another:

```
   6
  4  2
```

```
   5
  3  1
```
Example

Is there any fundamental difference?

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>a)</td>
<td>5 3 1</td>
<td>6 4 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b)</td>
<td>6 4 2</td>
<td>5 3 1</td>
</tr>
</tbody>
</table>
Example

Consider a matrix model:

a) 

\[
\begin{array}{c|c}
A & B \\
\hline
5 & 6 \\
3 & 4 \\
1 & 2 \\
\end{array}
\]

b) 

\[
\begin{array}{c|c}
A & B \\
\hline
6 & 5 \\
4 & 3 \\
2 & 1 \\
\end{array}
\]
Example

Consider a matrix model:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

NB: ‘1’ means place this item in this bin:

a)  
A: 5, 3, 1  
B: 6, 4, 2

b)  
A: 6, 4, 2  
B: 5, 3, 1
Example

Consider a matrix model:

If we insist that row $A \leq_{\text{lex}} \text{ row } B$, we remove a) from the solution set.

b)
Example

Notice that items 3 and 4 are identical.
Example

Notice that items 3 and 4 are identical.

If we insist that col 3 $\leq$ lex col 4, we remove c) from the solution set.
Aims

Main Goal

Eliminate row and column symmetries effectively and efficiently.

Aims:

Investigate types of ordering constraints to break row and column symmetries.

Devise global constraints to easily pose and efficiently solve the ordering constraints.

Examine the effectiveness of the ordering constraint
Lexicographic ordering is total.

Forcing the rows to be lexicographically ordered breaks all row symmetry.

\[
\begin{bmatrix}
A & B & C \\
D & E & F \\
G & H & I \\
\end{bmatrix} \leq_{\text{lex}}
\begin{bmatrix}
D & E & F \\
G & H & I \\
A & B & C \\
\end{bmatrix}
\]
Breaking both row and column symmetries is difficult (NP-Hard)

Rows and columns intersect

After constraining the rows to be lexicographically ordered

we distinguish the columns

the columns are not symmetric anymore!
Each symmetry class of assignments has at least one element where both the rows and the columns are lexicographically ordered.

But there may be no element with rows lex ordered and columns anti-lex ordered.

To break row and column symmetries, we can insist that the rows and columns are both lexicographically ordered (double-lex).

Extends to higher dimensions.
A symmetry class of assignments may have more than one element where both the rows and the columns are lexicographically ordered.

Double-lex does not break all row and column symmetries.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

swap the columns
swap row 1 and row 3

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Special Cases

All symmetry can be broken

When variables take distinct values
  Simply push largest value to a particular corner
Order 1st row and 1st col

0/1 variables, 1 occurs once in each row/col
  Double LEX then leaves an unique solution
How GACLex Works

- Consider the following example.
- We have two vectors of decision variables:

<table>
<thead>
<tr>
<th>x</th>
<th>{2}</th>
<th>{1,3,4}</th>
<th>{1,2,3,4,5}</th>
<th>{1,2}</th>
<th>{3,4,5}</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>{0,1,2}</td>
<td>{1}</td>
<td>{0,1,2,3,4}</td>
<td>{0,1}</td>
<td>{0,1,2}</td>
</tr>
</tbody>
</table>
How GACLex Works

- Consider the following example.
- We have two vectors of decision variables:

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th></th>
<th>y</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>{2}</td>
<td></td>
<td>{0,1,2}</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>{1,3,4}</td>
<td></td>
<td>{1}</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>{1,2,3,4,5}</td>
<td></td>
<td>{0,1,2,3,4}</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>{1,2}</td>
<td></td>
<td>{0,1}</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>{3,4,5}</td>
<td></td>
<td>{0,1,2}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- We want to enforce GAC on: $x \prec_{\text{lex}} y$. 
A Tale of Two Pointers

- We use two pointers, $\alpha$ and $\beta$, to avoid repeatedly traversing the vectors.
A Tale of Two Pointers

- We use two pointers, $\alpha$ and $\beta$, to avoid repeatedly traversing the vectors.
- We index the vectors as follows:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>x</strong></td>
<td>{2}</td>
<td>{1,3,4}</td>
<td>{1,2,3,4,5}</td>
<td>{1,2}</td>
<td>{3,4,5}</td>
</tr>
<tr>
<td><strong>y</strong></td>
<td>{0,1,2}</td>
<td>{1}</td>
<td>{0,1,2,3,4}</td>
<td>{0,1}</td>
<td>{0,1,2}</td>
</tr>
</tbody>
</table>
A Tale of Two Pointers

- We use two pointers, $\alpha$ and $\beta$, to avoid repeatedly traversing the vectors.
- We index the vectors as follows:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>{2}</td>
<td>{1,3,4}</td>
<td>{1,2,3,4,5}</td>
<td>{1,2}</td>
<td>{3,4,5}</td>
</tr>
<tr>
<td>y</td>
<td>{0,1,2}</td>
<td>{1}</td>
<td>{0,1,2,3,4}</td>
<td>{0,1}</td>
<td>{0,1,2}</td>
</tr>
</tbody>
</table>

Most Significant Index
A Tale of Two Pointers

- We use two pointers, $\alpha$ and $\beta$, to avoid repeatedly traversing the vectors.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>{2}</td>
<td>{1,3,4}</td>
<td>{1,2,3,4,5}</td>
<td>{1,2}</td>
<td>{3,4,5}</td>
</tr>
<tr>
<td>y</td>
<td>{0,1,2}</td>
<td>{1}</td>
<td>{0,1,2,3,4}</td>
<td>{0,1}</td>
<td>{0,1,2}</td>
</tr>
</tbody>
</table>

- $\alpha$: index such that all variables at more significant indices are ground and equal.
We use two pointers, $\alpha$ and $\beta$, to avoid repeatedly traversing the vectors.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>{2}</td>
<td>{1,3,4}</td>
<td>{1,2,3,4,5}</td>
<td>{1,2}</td>
<td>{3,4,5}</td>
</tr>
<tr>
<td>y</td>
<td>{0,1,2}</td>
<td>{1}</td>
<td>{0,1,2,3,4}</td>
<td>{0,1}</td>
<td>{0,1,2}</td>
</tr>
</tbody>
</table>

- $\alpha$: index such that all variables at more significant indices are ground and equal.
- $\beta$: most significant index from which the two vectors' tails necessarily violate the constraint.
A Tale of Two Pointers

- We use two pointers, $\alpha$ and $\beta$, to avoid repeatedly traversing the vectors.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>{2}</td>
<td>{1,3,4}</td>
<td>{1,2,3,4,5}</td>
<td>{1,2}</td>
<td>{3,4,5}</td>
</tr>
<tr>
<td>$y$</td>
<td>{0,1,2}</td>
<td>{1}</td>
<td>{0,1,2,3,4}</td>
<td>{0,1}</td>
<td>{0,1,2}</td>
</tr>
</tbody>
</table>

- $\alpha$: index such that all variables at more significant indices are ground and equal.
- $\beta$: If tails never violate the constraint: $\infty$
Pointer Initialisation

- Needs one traversal of the vectors (linear).

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>{2}</td>
<td>{1,3,4}</td>
<td>{1,2,3,4,5}</td>
<td>{1,2}</td>
<td>{3,4,5}</td>
</tr>
<tr>
<td>y</td>
<td>{0,1,2}</td>
<td>{1}</td>
<td>{0,1,2,3,4}</td>
<td>{0,1}</td>
<td>{0,1,2}</td>
</tr>
</tbody>
</table>

• $\alpha$: index such that all variables at more significant indices are ground and equal.
• $\beta$: most significant index from which the two vectors’ tails necessarily violate the constraint.
Pointer Initialisation

- Needs one traversal of the vectors (linear).

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>{2}</td>
<td>{1,3,4}</td>
<td>{1,2,3,4,5}</td>
<td>{1,2}</td>
<td>{3,4,5}</td>
</tr>
<tr>
<td>y</td>
<td>{0,1,2}</td>
<td>{1}</td>
<td>{0,1,2,3,4}</td>
<td>{0,1}</td>
<td>{0,1,2}</td>
</tr>
</tbody>
</table>

- \( \alpha \): index such that all variables at more significant indices are ground and equal.
- \( \beta \): most significant index from which the two vectors’ tails necessarily violate the constraint.
Failure

- For lexless, inconsistent if $\beta \leq \alpha$.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>{2}</td>
<td>{1}</td>
<td>{2}</td>
<td>{1,2}</td>
<td>{3,4,5}</td>
</tr>
<tr>
<td>y</td>
<td>{2}</td>
<td>{1}</td>
<td>{2}</td>
<td>{0,1}</td>
<td>{0,1,2}</td>
</tr>
</tbody>
</table>

Impossible to satisfy $x <_{\text{lex}} y$.

- $\alpha$: index such that all variables at more significant indices are ground and equal.
- $\beta$: most significant index from which the two vectors’ tails necessarily violate the constraint.
How GACLex Works

- We maintain $\alpha$ and $\beta$ as assignments made.
  - $\alpha$: index such that all variables at more significant indices are ground and equal.
  - $\beta$: most significant index from which the two vectors’ tails necessarily violate the constraint.
How GACLex Works

- We maintain $\alpha$ and $\beta$ as assignments made.
- When $\beta = \alpha + 1$ we enforce arc consistency on: $x_\alpha < y_\alpha$

- $\alpha$: index such that all variables at more significant indices are ground and equal.
- $\beta$: most significant index from which the two vectors’ tails necessarily violate the constraint.
How GACLex Works

- We maintain $\alpha$ and $\beta$ as assignments made.
- When $\beta = \alpha + 1$ we enforce arc consistency on:
  $$x_\alpha < y_\alpha$$

- $\alpha$: index such that all variables at more significant indices are ground and equal.
- $\beta$: most significant index from which the two vectors’ tails necessarily violate the constraint.
How GACLex Works

- We maintain $\alpha$ and $\beta$ as assignments made.
- When $\beta = \alpha + 1$ we enforce arc consistency on: $x_\alpha < y_\alpha$
- When $\beta > \alpha + 1$ we enforce arc consistency on: $x_\alpha \leq y_\alpha$

- $\alpha$: index such that all variables at more significant indices are ground and equal.
- $\beta$: most significant index from which the two vectors’ tails necessarily violate the constraint.
How GACLex Works

- We maintain $\alpha$ and $\beta$ as assignments made.
- Key: we reduce GAC on vectors to AC on binary constraints.

- $\alpha$: index such that all variables at more significant indices are ground and equal.
- $\beta$: most significant index from which the two vectors’ tails necessarily violate the constraint.
How GACLex Works

- 0, 1 removed from $y_\alpha$.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>{2}</td>
<td>{1,3,4}</td>
<td>{1,2,3,4,5}</td>
<td>{1,2}</td>
<td>{3,4,5}</td>
</tr>
<tr>
<td>y</td>
<td>{0,1,2}</td>
<td>{1}</td>
<td>{0,1,2,3,4}</td>
<td>{0,1}</td>
<td>{0,1,2}</td>
</tr>
</tbody>
</table>

- $\alpha$: index such that all variables at more significant indices are ground and equal.
- $\beta$: most significant index from which the two vectors’ tails necessarily violate the constraint.
How GACLex Works

- 0, 1 removed from $y_\alpha$.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>{2}</td>
<td>{1,3,4}</td>
<td>{1,2,3,4,5}</td>
<td>{1,2}</td>
<td>{3,4,5}</td>
</tr>
<tr>
<td>y</td>
<td>{0,1,2}</td>
<td>{1}</td>
<td>{0,1,2,3,4}</td>
<td>{0,1}</td>
<td>{0,1,2}</td>
</tr>
</tbody>
</table>

- $\alpha$: index such that all variables at more significant indices are ground and equal.
- $\beta$: most significant index from which the two vectors’ tails necessarily violate the constraint.
How GACLex Works

- Update $\alpha$.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>{2}</td>
<td>{1,3,4}</td>
<td>{1,2,3,4,5}</td>
<td>{1,2}</td>
<td>{3,4,5}</td>
</tr>
<tr>
<td>$y$</td>
<td>{0,1,2}</td>
<td>{1}</td>
<td>{0,1,2,3,4}</td>
<td>{0,1}</td>
<td>{0,1,2}</td>
</tr>
</tbody>
</table>

- $\alpha$: index such that all variables at more significant indices are ground and equal.
- $\beta$: most significant index from which the two vectors’ tails necessarily violate the constraint.
How GACLex Works

- Update $\alpha$.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>{2}</td>
<td>{1,3,4}</td>
<td>{1,2,3,4,5}</td>
<td>{1,2}</td>
<td>{3,4,5}</td>
</tr>
<tr>
<td>$y$</td>
<td>{0,1,2}</td>
<td>{1}</td>
<td>{0,1,2,3,4}</td>
<td>{0,1}</td>
<td>{0,1,2}</td>
</tr>
</tbody>
</table>

- $\alpha$: index such that all variables at more significant indices are ground and equal.
- $\beta$: most significant index from which the two vectors’ tails necessarily violate the constraint.
How GACLex Works

- 3, 4 removed from $x_\alpha$.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>{2}</td>
<td>{1,3,4}</td>
<td>{1,2,3,4,5}</td>
<td>{1,2}</td>
<td>{3,4,5}</td>
</tr>
<tr>
<td>y</td>
<td>{0,1,2}</td>
<td>{1}</td>
<td>{0,1,2,3,4}</td>
<td>{0,1}</td>
<td>{0,1,2}</td>
</tr>
</tbody>
</table>

- $\alpha$: index such that all variables at more significant indices are ground and equal.
- $\beta$: most significant index from which the two vectors’ tails necessarily violate the constraint.
### How GACLex Works

- 3, 4 removed from $x_\alpha$.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>{2}</td>
<td>{1,3,4}</td>
<td>{1,2,3,4,5}</td>
<td>{1,2}</td>
<td>{3,4,5}</td>
</tr>
<tr>
<td>y</td>
<td>{0,1,2}</td>
<td>{1}</td>
<td>{0,1,2,3,4}</td>
<td>{0,1}</td>
<td>{0,1,2}</td>
</tr>
</tbody>
</table>

- $\alpha$: index such that all variables at more significant indices are ground and equal.
- $\beta$: most significant index from which the two vectors’ tails necessarily violate the constraint.
How GACLex Works

- Update $\alpha$.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>${2}$</td>
<td>${1,3,4}$</td>
<td>${1,2,3,4,5}$</td>
<td>${1,2}$</td>
<td>${3,4,5}$</td>
</tr>
<tr>
<td>$y$</td>
<td>${0,1,2}$</td>
<td>${1}$</td>
<td>${0,1,2,3,4}$</td>
<td>${0,1}$</td>
<td>${0,1,2}$</td>
</tr>
</tbody>
</table>

- $\alpha$: index such that all variables at more significant indices are ground and equal.
- $\beta$: most significant index from which the two vectors’ tails necessarily violate the constraint.
How GACLex Works

- Update \( \alpha \).

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>{2}</td>
<td>{1,3,4}</td>
<td>{1,2,3,4,5}</td>
<td>{1,2}</td>
<td>{3,4,5}</td>
</tr>
<tr>
<td>( y )</td>
<td>{0,1,2}</td>
<td>{1}</td>
<td>{0,1,2,3,4}</td>
<td>{0,1}</td>
<td>{0,1,2}</td>
</tr>
</tbody>
</table>

- \( \alpha \): index such that all variables at more significant indices are ground and equal.
- \( \beta \): most significant index from which the two vectors’ tails necessarily violate the constraint.
How GACLex Works

- 4, 5 removed from $x_\alpha$, 0, 1 removed from $y_\alpha$.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>{2}</td>
<td>{1,3,4}</td>
<td>{1,2,3,4,5}</td>
<td>{1,2}</td>
<td>{3,4,5}</td>
</tr>
<tr>
<td>y</td>
<td>{0,1,2}</td>
<td>{1}</td>
<td>{0,1,2,3,4}</td>
<td>{0,1}</td>
<td>{0,1,2}</td>
</tr>
</tbody>
</table>

- $\alpha$: index such that all variables at more significant indices are ground and equal.
- $\beta$: most significant index from which the two vectors’ tails necessarily violate the constraint.
How GACLex Works

- 4, 5 removed from $x_\alpha$, 0, 1 removed from $y_\alpha$.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>{2}</td>
<td>{1,3,4}</td>
<td>{1,2,3,4,5}</td>
<td>{1,2}</td>
<td>{3,4,5}</td>
</tr>
<tr>
<td>$y$</td>
<td>{0,1,2}</td>
<td>{1}</td>
<td>{0,1,2,3,4}</td>
<td>{0,1}</td>
<td>{0,1,2}</td>
</tr>
</tbody>
</table>

- $\alpha$: index such that all variables at more significant indices are ground and equal.
- $\beta$: most significant index from which the two vectors’ tails necessarily violate the constraint.
Complexity

- **Initialisation:** $O(n)$
- **Propagation:**
  - We enforce arc consistency between at most $n$ pairs of variables: $x_\alpha < y_\alpha$ or $x_\alpha \leq y_\alpha$.
  - Cost: $O(d)$.
  - Overall cost: $O(nd)$.
- **Triggers on lower and upper bounds of all variables**
AllDifferent

- AllDifferent([x₁, x₂, ..., xₙ])
- Variables x₁, x₂, ..., xₙ all take different values in any solution
- Lots of propagation algorithms available
  - AC on pairwise not-equal
  - Bounds consistency (only reads and updates bounds)
  - Generalized arc consistency (GAC)
AllDifferent

- Very widely used – very important
- Examples:
  - A student must have lectures at distinct times
  - In a sports schedule, the teams playing on a particular week are all distinct
  - No pair of golfers play together more than once
  - Sudoku, Kakuro generation
  - Generating Golomb rulers (radio telescope scheduling)
  - Anything involving a permutation of a set of elements
Régis's Algorithm

- Enforces GAC
- Therefore, can prune when AC not-equal constraints (on all pairs) cannot:

\[
x_1, \{1,2\} \\
x_2, \{1,3\} \\
x_3, \{2,3\} \\
x_4, \{1,2,3,4\}
\]

All values supported on each arc

Values 1,2,3 of \(x_4\) are not supported
Alldifferent feasibility and pruning

Feasibility? Given domains, create domain/variable bipartite graph
Alldifferent feasibility and pruning

Pruning? Which edges are in no matching?

A matching in a graph is a set of edges without common vertices.

Domain is sharply reduced.
Régin's Algorithm

- Easiest to understand in terms of a flow graph
- Find a maximum flow in the graph
Régis's Algorithm

- Find maximum flow from $s$ to $t$
- All edges carry 0 or 1 units
- Think of water flowing through pipes
Régine's Algorithm

- Find maximum flow from $s$ to $t$
- Ford-Fulkerson algorithm
- Augmenting path in green
Régine's Algorithm

- Find maximum flow from s to t
- Ford-Fulkerson algorithm
- Augmenting path in green
Régin's Algorithm

- Find maximum flow from s to t
- Ford-Fulkerson algorithm
- Augmenting path in green
Régin's Algorithm

- Find maximum flow from s to t
- Ford-Fulkerson algorithm
- Augmenting path in green
Régis's Algorithm

- Find maximum flow from s to t
- Ford-Fulkerson algorithm
- Augmenting path in green
Régis's Algorithm

- Find maximum flow from $s$ to $t$
- Ford-Fulkerson algorithm
- Augmenting path in green
Régin's Algorithm

- Find maximum flow from s to t
- Ford-Fulkerson algorithm
- Augmenting path in green
Régin's Algorithm

- Completed maximum flow from $s$ to $t$
- Covers all variables (constraint is satisfiable)
- One of 24
Régis's Algorithm

- Find strongly-connected components
Régin's Algorithm

- Strongly-connected components (SCCs)
  - Vertices $i$ and $j$ in same SCC iff:
    - Path from $i$ to $j$ and from $j$ to $i$ in digraph
  - Found by Tarjan's algorithm
    - Depth-first search in the graph
Régis's Algorithm

- Find strongly-connected components
Régis's Algorithm

- Cycle within SCC
- Apply cycle to find different maximum flow
- No cycles between SCCs
Régis's Algorithm

- Cycle within SCC
- Apply cycle to find different maximum flow
- No cycles between SCCs
Régis's Algorithm

- No cycles between SCCs
- No maximum flows involving $x_3=2$ or $x_4=2$
Régine's Algorithm

- Remove edges which are:
  - Between SCCs
  - Not in flow
  - Corresponds to theorem by Berge, 1973
Régis's Algorithm

- Remove edges which are:
  - Between SCCs
  - Not in flow
- Delete: 2 from $D_3$ and $D_4$
Régine's Algorithm

- Alternative understanding:

- \( A = \{x_1, x_2\} \) is a Hall set

- \( B = \{1,2\} \) is the set of neighbouring values

- \( |A| = |B| \)

Therefore, values in \( B \) are used up by \( A \)
Régin's Algorithm

- When to propagate this constraint?
  - All variable-value arcs in the graph could be part of the flow
  - Any value removal could affect the flow and/or SCC analysis
  - Propagate for any value removal
  - Expensive! $O(r^2d)$
Conclusions

- Less-than – very simple propagator with an action directly linked to each trigger
- GACLex – more complicated propagator with incremental state: \( \alpha, \beta \)
- allDifferent – one of the most complicated polynomial propagation algorithms