Chapter 16: More Global Constraints (Car Sequencing)

Helmut Simonis

Cork Constraint Computation Centre
Computer Science Department
University College Cork
Ireland

ECLiPSe ELearning
This work is licensed under the Creative Commons Attribution-Noncommercial-Share Alike 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/3.0/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.
Outline

1. Problem
2. Program
3. Search
4. Improved Search Strategy
What we want to introduce

- Car sequencing problem
- gcc global cardinality constraint
- sequence constraint
- Search does not always have to be based on original problem variables
- Can be useful to consider additional variables which allow more clever search
Outline

1. Problem
2. Program
3. Search
4. Improved Search Strategy
We have to schedule a number of cars for production on an assembly line. Each car is of a certain type, and we know how many cars of each type we have to produce. Car types differ in the options they require, i.e. sun-roof, air conditioning. For each option, we have capacity limits on the assembly line, expressed as \( k \) cars out of \( n \) consecutive cars on the line may have some option. Find an assignment which produces the correct number of cars of each type, while satisfying the capacity constraints.
Example (DSV88)

- 100 cars
- 18 types
- 5 options
  - Option 1: 1 out of 2
  - Option 2: 2 out of 3
  - Option 3: 1 out of 3
  - Option 4: 2 out of 5
  - Option 5: 1 out of 5
## Car Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Cars Required</th>
<th>Option 1</th>
<th>Option 2</th>
<th>Option 3</th>
<th>Option 4</th>
<th>Option 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>11</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>9</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Solution
Modelling Alternatives

- Assign start time (sequence number) to each car
  - 100 variables, each with 100 values
  - Handling of car types implicit
  - Symmetry breaking for cars of same type (inequalities)?
  - Capacity constraints?

- Assign car type to each slot on assembly line
  - 100 variables, 18 values
  - How to control number of cars of each type?
  - How to express capacity constraints?
Modelling Alternatives

- Assign start time (sequence number) to each car
  - 100 variables, each with 100 values
  - Handling of car types implicit
  - Symmetry breaking for cars of same type (inequalities)?
  - Capacity constraints?

- Assign car type to each slot on assembly line
  - 100 variables, 18 values
  - How to control number of cars of each type?
  - How to express capacity constraints?
100 variables ranging over car types
- \textit{gcc} constraint to control number of items with same type
- $5 \times 100$ 0/1 variables indicating use of option for each slot
- \textit{element} constraints to map car types to options used
- \textit{sequence} constraints to enforce limits on each option
Reminder: \texttt{gcc(Pattern, Variables)}

- \texttt{gcc} \textit{global cardinality constraint}
- Pattern is list of terms \texttt{gcc(Low, High, Value)}
- The overall number of variables taking value \texttt{Value} is between \texttt{Low} and \texttt{High}
- Generalization of \texttt{alldifferent}
- Domain consistent version in ECLiPSe
Example

\[\begin{align*}
\text{X1} &: [2,4], \quad \text{X2} &: [1,3,4], \quad \text{X3} &: [1,2,3,4], \\
\text{X4} &: [3,4,5], \quad \text{X5} &: [3,4,5], \\
gcc([gcc(1,1,1), gcc(2,3,2), gcc(1,3,3), \\
gcc(0,4,4), gcc(1,3,5)], \\
[\text{X1}, \text{X2}, \text{X3}, \text{X4}, \text{X5}]), \\
\text{X1} &= ?, \quad \text{X2} = ?, \quad \text{X3} = ?, \quad \text{X4} = ?, \quad \text{X5} = ?
\end{align*}\]
gcc Reasoning

\[ X_1 : [2, 4], X_2 : [1, 3, 4], X_3 : [1, 2, 3, 4], \]
\[ X_4 : [3, 4, 5], X_5 : [3, 4, 5], \]
\[ gcc([gcc(1, 1, 1), gcc(2, 3, 2), gcc(1, 3, 3),
    gcc(0, 4, 4), gcc(1, 3, 5)],
   [X_1, X_2, X_3, X_4, X_5]), \]
\[ X_1 = ?, X_2 = ?, X_3 = ?, X_4 = ?, X_5 = ? \]
gcc Reasoning

\[ X_1 ::= [2, 4], \quad X_2 ::= [1, 3, 4], \quad X_3 ::= [1, 2, 3, 4], \]
\[ X_4 ::= [3, 4, 5], \quad X_5 ::= [3, 4, 5], \]
\[ gcc([gcc(1, 1, 1), gcc(2, 3, 2), gcc(1, 3, 3), \]
\[ \quad gcc(0, 4, 4), gcc(1, 3, 5)], \]
\[ [X_1, X_2, X_3, X_4, X_5]), \]

\[ X_1 = ?, \quad X_2 = ?, \quad X_3 = ?, \quad X_4 = ?, \quad X_5 = ? \]
**Problem**

**Program**

**Search**

**Improved Search Strategy**

---

**gcc Reasoning**

\[
\begin{align*}
X_1 & : [2, 4], \quad X_2 : [1, 3, 4], \quad X_3 : [1, 2, 3, 4], \\
X_4 & : [3, 4, 5], \quad X_5 : [3, 4, 5], \\
gcc([gcc(1, 1, 1), gcc(2, 3, 2), gcc(1, 3, 3), \\
gcc(0, 4, 4), gcc(1, 3, 5)], \\
[X_1, X_2, X_3, X_4, X_5]),
\end{align*}
\]

\[X_1 = ?, \quad X_2 = ?, \quad X_3 = ?, \quad X_4 = ?, \quad X_5 = ?\]
gcc Reasoning

\[ X_1 :: [2, 4], \ X_2 :: [1, 3, 4], \ X_3 :: [1, 2, 3, 4], \]
\[ X_4 :: [3, 4, 5], \ X_5 :: [3, 4, 5], \]
\[ gcc([gcc(1, 1, 1), gcc(2, 3, 2), gcc(1, 3, 3), \]
\[ gcc(0, 4, 4), gcc(1, 3, 5)], \]
\[ [X_1, X_2, X_3, X_4, X_5]), \]
\[ X_1 = 2, \ X_2 = ?, \ X_3 = 2, \ X_4 = ?, \ X_5 = ? \]
X₁ : [2, 4], X₂ : [1, 3, 4], X₃ : [1, 2, 3, 4],
X₄ : [3, 4, 5], X₅ : [3, 4, 5],
gcc([gcc(1, 1, 1), gcc(2, 3, 2), gcc(1, 3, 3),
gcc(0, 4, 4), gcc(1, 3, 5)],
[X₁, X₂, X₃, X₄, X₅]),

X₁ = 2, X₂ = ?, X₃ = 2, X₄ = ?, X₅ = ?
\( \text{gcc} \) Next Step

\[
X_1 : [2,4], \quad X_2 : [1,3,4], \quad X_3 : [\bot,2,3,4], \\
X_4 : [3,4,5], \quad X_5 : [3,4,5], \\
gcc([gcc(1,1,1), gcc(2,3,2), gcc(1,3,3), \\
gcc(0,4,4), gcc(1,3,5)], \\
[X_1, X_2, X_3, X_4, X_5]),
\]

\( X_1 = 2, \ X_2 = ?, \ X_3 = 2, \ X_4 = ?, \ X_5 = ? \)
X1 :: [2,4], X2 :: [1,3,4], X3 :: [1,2,3,4],
X4 :: [3,4,5], X5 :: [3,4,5],
gcc([gcc(1,1,1), gcc(2,3,2), gcc(1,3,3),
gcc(0,4,4), gcc(1,3,5)],
[X1, X2, X3, X4, X5]),

X1 = 2, X2 = ?, X3 = 2, X4 = ?, X5 = ?
\texttt{gcc} \textbf{Next Step}

\begin{align*}
X_1 &:: [2,4], \quad X_2 :: [1,3,4], \quad X_3 :: [\bot,2,3,4], \\
X_4 &:: [3,4,5], \quad X_5 :: [3,4,5], \\
gcc([gcc(1,1,1), gcc(2,3,2), gcc(1,3,3), \\
    gcc(0,4,4), gcc(1,3,5)], \\
    [X_1, X_2, X_3, X_4, X_5]), \\
X_1 = 2, \quad X_2 = 1, \quad X_3 = 2, \quad X_4 = \text{?}, \quad X_5 = \text{?}
\end{align*}
\begin{align*}
X_1 & : [2, 4], \quad X_2 : [1, 3, 4], \quad X_3 : [1, 2, 3, 4], \\
X_4 & : [3, 4, 5], \quad X_5 : [3, 4, 5], \\
gcc([gcc(1, 1, 1), gcc(2, 3, 2), gcc(1, 3, 3), \\
gcc(0, 4, 4), gcc(1, 3, 5)], \\
[X_1, X_2, X_3, X_4, X_5]), \\
X_1 &= 2, \quad X_2 = 1, \quad X_3 = 2, \quad X_4 = ?, \quad X_5 = ?
\end{align*}
Continued

X1 :: [2,4], X2 :: [1,3,4], X3 :: [4,2,3,4],
X4 :: [3,4,5], X5 :: [3,4,5],
gcc([gcc(1,1,1), gcc(2,3,2), gcc(1,3,3),
gcc(0,4,4), gcc(1,3,5)],
[X1, X2, X3, X4, X5]),

X1 = 2, X2 = 1, X3 = 2, X4 = ?, X5 = ?
\texttt{gcc} Continued

\begin{align*}
X1 & : \ [2,4], \ X2 : \ [1,3,4], \ X3 : \ [\bot,2,3,4], \\
X4 & : \ [3,4,5], \ X5 : \ [3,4,5], \\
gcc([gcc(1,1,1),gcc(2,3,2),gcc(1,3,3), \\
gcc(0,4,4),gcc(1,3,5)], \\
[X1,X2,X3,X4,X5]),
\end{align*}

\begin{align*}
X1 = 2, \ X2 = 1, \ X3 = 2, \ X4 = ?, \ X5 = ?
\end{align*}
\( X_1 :: [2, 4], \ X_2 :: [1, 3, 4], \ X_3 :: [1, 2, 3, 4], \)
\( X_4 :: [3, 4, 5], \ X_5 :: [3, 4, 5], \)
\[
\text{gcc} ([\text{gcc}(1, 1, 1), \text{gcc}(2, 3, 2), \text{gcc}(1, 3, 3), \\
\text{gcc}(0, 4, 4), \text{gcc}(1, 3, 5)], \\
[X_1, X_2, X_3, X_4, X_5])
\]

\( X_1 = 2, \ X_2 = 1, \ X_3 = 2, \ X_4 = ?, \ X_5 = ? \)
Problem
Program
Search
Improved Search Strategy

Made Domain Consistent

gcc

X1 :: [2, 4], X2 :: [1, 3, 4], X3 :: [1, 2, 3, 4],
X4 :: [3, 4, 5], X5 :: [3, 4, 5],
gcc([gcc(1, 1, 1), gcc(2, 3, 2), gcc(1, 3, 3),
gcc(0, 4, 4), gcc(1, 3, 5)],
[X1, X2, X3, X4, X5]),

X1 = 2, X2 = 1, X3 = 2, X4 ∈ {3, 5}, X5 ∈ {3, 5}
How does the constraint solver do that?

Explained in optional material at end

- Domain Consistent gcc
Reminder: $\text{element}(X, \text{List}, Y)$

- \text{List} is a list of integers
- The $X^{th}$ element of \text{List} is $Y$
- The index starts from 1
- Typical uses:
  - Projection
  - Cost
Element Examples

Prime is 1 iff $X \in 1..10$ is a prime number

$X :: 1..10,$
element($X, [1,1,1,0,1,0,1,0,0,0], Prime),

Cost is the cost corresponding to the assignment of $Y$

$Y :: 1..10,$
element($Y, [5,3,34,0,3,1,12,12,1,3], Cost)
sequence_total (Min, Max, Low, High, K, Vars)

- Variables \( Vars \) have 0/1 domain
- Between \( Min \) and \( Max \) variables have value 1
- For every sub-sequence of length \( K \), between \( Low \) and \( High \) variables have value 1
sequence_total Example

\[
\begin{align*}
\text{sequence_total}(2, 3, 1, 2, 3, \ [X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8, X_9, X_{10}]) &= 0..1, \\
X_1 &= 0, \ X_4 = 0, \ X_7 = 0, \ X_{10} = 0
\end{align*}
\]
Example, cont’d

\[ x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10} \]
Example, cont’d

\[ x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10} \]
Example, cont’d

\[
\begin{array}{c}
1..2 \\
\overbrace{x_1, x_2, x_3, x_4} \quad \overbrace{x_5, x_6, x_7} \quad \overbrace{x_8, x_9, x_{10}}
\end{array}
\]
Example, cont’d

\[\begin{aligned}
&\{x_1, x_2, x_3, x_4\} \\
&\{x_5, x_6, x_7\} \\
&\{x_8, x_9, x_{10}\}
\end{aligned}\]
Example, cont’d

\[ x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10} \]

- 3..6
- 1..2
- 2..3
Example, cont’d

\[
\begin{align*}
X_1, X_2, X_3, X_4, & \quad X_5, X_6, X_7, \\
& \quad X_8, X_9, X_{10}, \\
1..2, & \quad 1..2, & \quad 1..2, \\
2..3, & \quad 3..6
\end{align*}
\]
Example, cont’d

\[\begin{array}{c}
0, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10} \\
\end{array}\]

\[\begin{array}{c}
1..2 \\
2..3 \\
1..2 \\
3..6 \\
1..2 \\
\end{array}\]

Helmut Simonis  More Global Constraints
Mathematical Equivalent

\[ \text{Vars} = [x_1, x_2, \ldots x_N] \]

\[ \text{Min} \leq \sum_{1 \leq i \leq N} x_i \leq \text{Max} \]

\[ 1 \leq s \leq N - k + 1 : \quad \text{Low} \leq \sum_{s \leq j \leq s + k - 1} x_j \leq \text{High} \]
Pruning very different when using finite domain inequalities
Currently no domain consistent implementation of `sequence_total`
Weaker version `sequence` (no global counters) domain consistent
Currently using decomposition:

\[ \text{sequence}_\text{total} = \text{sequence} + \text{gcc} + \text{more} \]
Outline

1. Problem
2. Program
3. Search
4. Improved Search Strategy
Main Program

:-module(car).
:-export(top/0).
:-lib(ic).
:-lib(ic_global_gac).

top:-
     problem(Problem),
     model(Problem,L),
     writeln(L).
Structure Definitions

:-local struct(problem(cars,
                        models,
                        required,
                        using_options,
                        value_order)).

:-local struct(option(k,
                       n,
                       index_set,
                       total_use)).
Model (Part 1)

model(problem{cars:NrCars, models:NrModels, required:Required, using_options:List, value_order:Ordered},L):-

length(L,NrCars),
L :: 1..NrModels,
(foreach(Cnt,Required),
count(J,1,_),
foreach(gcc(Cnt,Cnt,J),Card) do true ),
gcc(Card,L),
...
Model (Part 2)

... 

(foreach (option {k: K, 
n: N, 
index_set: IndexSet, 
total_use: Total}, List), 

param (L, NrCars) do 

(foreach (X, L), 

foreach (B, Binary), 

param (IndexSet) do 

  element (X, IndexSet, B) 

), 

sequence_total (Total, Total, 0, K, N, Binary) 

), 

search (L, 0, input_order, ordered (Ordered), 


Data

```prolog
problem(100,18,
    [5,3,7,1,10,2,11,5,4,6,12,1,1,5,9,5,12,1],
    [option(1,2, [1,2,3,5,6,7,8,14],
        [1,1,1,0,1,1,1,0,0,0,0,1,0,0,0,0],48),
    option(2,3, [1,2,3,4,5,9,10,11,15],
        [1,1,1,1,1,0,0,0,1,1,1,0,0,1,0,0,0],57),
    option(1,3, [3,4,8,11,12,13,18],
        [0,0,1,1,0,0,0,1,0,1,1,1,0,0,0,1],28),
    option(2,5, [2,4,7,10,13,17],
        [0,1,0,1,0,0,1,0,0,1,0,0,0,1,0,0],34),
    option(1,5, [1,6,9,12,16],
        [1,0,0,0,0,1,0,0,1,0,0,1,0,0,1,0],17)])
```

Helmut Simonis

More Global Constraints 47
Data Generation

- Data not really stored as facts
- Generated from text data files in different format
- Benchmark set from CSPLIB
  (http://www.csplib.org)
Outline

1. Problem
2. Program
3. Search
   - DSV88 Example
   - More Difficult Example
4. Improved Search Strategy
Classical Example
Classical Example
Classical Example
Classical Example

1 1
1
2
4
3
7
5

DSV88 Example
More Difficult Example

Improved Search Strategy

Back to Start  Skip Animation

Helmut Simonis  More Global Constraints
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example

1 2 3 5 7 8 9 11 12 13 15 16
3 2 11 6 5 13 2 3 7 4 1

Back to Start  Skip Animation

Helmut Simonis  More Global Constraints
Classical Example
Classical Example

Improved Search Strategy

DSV88 Example
More Difficult Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example

Improved Search Strategy

DSV88 Example
More Difficult Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example

Improved Search Strategy

DSV88 Example
More Difficult Example

Problem
Program
Search

Helmut Simonis
More Global Constraints
Classical Example
Classical Example

Improved Search Strategy

DSV88 Example

More Difficult Example

Helmut Simonis

More Global Constraints
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example

DSV88 Example
More Difficult Example

Helmut Simonis
More Global Constraints
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Classical Example
Another Example (PR97)

- 100 cars
- 22 types
- 5 options
  - Option 1: 1 out of 2
  - Option 2: 2 out of 3
  - Option 3: 1 out of 3
  - Option 4: 2 out of 5
  - Option 5: 1 out of 5
## Second Example: Car Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Cars Required</th>
<th>Option 1</th>
<th>Option 2</th>
<th>Option 3</th>
<th>Option 4</th>
<th>Option 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>15</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>8</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>22</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Search (Stopped After 1000 Nodes)
Observation

- This does not look good
- Typical thrashing behaviour
- We made a wrong choice at some point
- ... but did not detect it
- Many additional choices are made before failure is detected
- We have to explore the complete tree under the wrong choice
- This is far too expensive
Outline

1. Problem
2. Program
3. Search
4. Improved Search Strategy
Do not label car slot variables
Decide instead if slot should use an option or not
This restricts the car models which can be placed in this slot
Start with the most restricted option
When all options are assigned, the car type is fixed
Potential problem: We now have 500 instead of 100 decision variables
Naive searchspace $2^{500} = 3.2 \times 10^{150}$ instead of $2^{100} = 1.7 \times 10^{134}$
Second Modification

- Instead of assigning values left to right
- Start assigning in middle of board
- And alternate around middle until you reach edges
- Idea: Slots at edges are less constrained, i.e. easier to assign
- Save those slots until the end
- We already encountered this idea for the N-Queens problem
Modified Search

51
Modified Search
Modified Search
Modified Search

Helmut Simonis
More Global Constraints
Modified Search
Modified Search
Modified Search
**Modified Search**

- 51
- 53
- 55
- 57
- 59
- 61
- 63
- 65

Back to Start  Skip Animation

Helmut Simonis  More Global Constraints
Modified Search
Modified Search
Modified Search
Modified Search
Modified Search
Modified Search
Modified Search

Improved Search Strategy

Helmut Simonis

More Global Constraints
Modified Search
Modified Search
Modified Search
Modified Search
Modified Search
Improved Search Strategy

Modified Search
Modified Search
Modified Search
Modified Search
Modified Search
Modified Search

Improved Search Strategy
Improved Search Strategy

Modified Search
Modified Search
Modified Search
Improved Search Strategy

Modified Search
Modified Search
Modified Search
Modified Search
Modified Search
Modified Search
Modified Search
Modified Search
Modified Search

Improved Search Strategy
Modified Search
Modified Search
Modified Search
Modified Search
Modified Search
Modified Search
Improved Search Strategy

Modified Search

[Diagram of the Modified Search strategy]
Modified Search
Modified Search
Modified Search
Improved Search Strategy

Modified Search

Back to Start  
Skip Animation

Helmut Simonis  
More Global Constraints
Modified Search
Modified Search
Modified Search
Improved Search Strategy

Modified Search
Modified Search
Modified Search
Modified Search
Modified Search
Modified Search
Modified Search
Modified Search
Modified Search
Modified Search
Modified Search
Improved Search Strategy

Modified Search
Modified Search
Improved Search Strategy

Modified Search
Modified Search
Modified Search
Modified Search
Modified Search
Modified Search
Modified Search
Modified Search
Modified Search
Modified Search
Modified Search
Modified Search
Modified Search
Modified Search
Modified Search
Improved Search Strategy

Modified Search
Improved Search Strategy

Modified Search
Modified Search
Modified Search
Modified Search
Improved Search Strategy

Modified Search
Modified Search
Modified Search
Modified Search
Modified Search
Modified Search
Modified Search
Modified Search
Improved Search Strategy

Modified Search
Modified Search
Modified Search
Modified Search

Improved Search Strategy
Modified Search
Modified Search

Improved Search Strategy

Problem
Program
Search

Back to Start  Skip Animation
Modified Search
Improved Search Strategy

Modified Search
Modified Search
Modified Search
Modified Search
Improved Search Strategy

Modified Search

[Diagram of modified search strategy]
Modified Search
Modified Search
Modified Search
Modified Search

Problem
Program
Search
Improved Search Strategy

Helmut Simonis

More Global Constraints
Modified Search
Modified Search
Modified Search
Modified Search
Modified Search
Improved Search Strategy

Modified Search
Modified Search
Modified Search

Improved Search Strategy
Observations

- Important to start in middle
- Making hard choices first
- Concentrate on difficult to satisfy sub-problem
- Number of choices is much smaller than number of variables
- Some assignments lead to a lot of propagation
Conclusions

- Introduced global constraint sequence
- Reuse gcc and element
- Search on auxiliary variables can work well
- Raw search space measures are unreliable
- Modelling idea
  - Decide what to make in a given time slot
  - ... and not when to schedule some given activity
Making \texttt{gcc} Domain Consistent
X1 :: [2, 4], X2 :: [1, 3, 4], X3 :: [1, 2, 3, 4],
X4 :: [3, 4, 5], X5 :: [3, 4, 5],
gcc([gcc(1, 1, 1), gcc(2, 3, 2), gcc(1, 3, 3),
    gcc(0, 4, 4), gcc(1, 3, 5)],
   [X1, X2, X3, X4, X5]),
Method: Max Flow Model

- Express constraint as max-flow problem
- Any flow solution corresponds to a valid assignment
- Only work with one flow solution
- Obtain all others by considering
  - residual graph and
  - strongly connected components
- Classical method, faster methods exist
- Use of max flow based propagators for many constraints
Start with Value Graph

\[ X_1 \rightarrow 1 \]
\[ X_2 \rightarrow 2 \]
\[ X_3 \rightarrow 3 \]
\[ X_4 \rightarrow 4 \]
\[ X_5 \rightarrow 5 \]
Making gcc Domain Consistent

Convert to Flow Problem

\[ X_1 \rightarrow X_2 \rightarrow X_3 \rightarrow X_4 \rightarrow X_5 \rightarrow t \]

\[ s \rightarrow X_1 \rightarrow X_2 \rightarrow X_3 \rightarrow X_4 \rightarrow X_5 \rightarrow t \]

Helmut Simonis

More Global Constraints
Find Maximal Flow
Making gcc Domain Consistent

Mark Value Edges in Flow

\[ X_1 \quad X_2 \quad X_3 \quad X_4 \quad X_5 \]

\[ 1 \quad 2 \quad 3 \quad 4 \quad 5 \]

Helmut Simonis
Residual Graph
Find Strongly Connected Components
Mark Edges

\[
\begin{array}{c}
s \quad X_1 \quad 1 \\
X_2 \quad 2 \\
X_3 \quad 3 \\
X_4 \quad 4 \\
X_5 \quad 5 \\
t
\end{array}
\]
Remove Unmarked Edges

Making gcc Domain Consistent

Helmut Simonis
More Global Constraints
Constraint is Domain Consistent

$X_1, X_2, X_3, X_4, X_5$
Mehmet Dincbas, Helmut Simonis, and Pascal Van Hentenryck.
Solving the car-sequencing problem in constraint logic programming.

Jean-Charles Regin and Jean-Francois Puget.
A filtering algorithm for global sequencing constraints.
Christine Solnon, Van Dat Cung, Alain Nguyen, and Christian Artigues.  

Willem Jan van Hoeve, Gilles Pesant, Louis-Martin Rousseau, and Ashish Sabharwal.  
Revisiting the sequence constraint.  
Michael J. Maher, Nina Narodytska, Claude-Guy Quimper, and Toby Walsh.
Flow-based propagators for the sequence and related global constraints.