Modeling and Solving Constraint Problems

Emmanuel Hebrard
Avant propos
Introduction to constraint programming (no pre-requisite)
Avant propos

- Introduction to constraint programming (no pre-requisite)
  - Or almost none
Avant propos

- Introduction to constraint programming (no pre-requisite)
  - Or almost none
  - Constraint programming = combinatorial branch & bound plus a lot of jargon
Avant propos

- Introduction to constraint programming (no pre-requisite)
  - Or almost none
  - Constraint programming = combinatorial branch & bound plus a lot of jargon

- Language-level modeling: stating and solving a problem with an off-the-shelf toolkit
  - Notions of model and solver
Introduction to constraint programming (no pre-requisite)

- Or almost none

- Constraint programming = combinatorial branch & bound plus a lot of jargon

Language-level modeling: stating and solving a problem with an off-the-shelf toolkit

- Notions of model and solver

- I will not talk about user-defined propagator
Avant propos

- Introduction to constraint programming (no pre-requisite)
  - Or almost none
  - Constraint programming = combinatorial branch & bound plus a lot of jargon

- Language-level modeling: stating and solving a problem with an off-the-shelf toolkit
  - Notions of model and solver
  - I will not talk about user-defined propagator
  - I will not talk about search strategies (though there are things to do at the language level)
Avant propos

- Introduction to constraint programming (no pre-requisite)
  - Or almost none
  - Constraint programming = combinatorial branch & bound plus a lot of jargon

- Language-level modeling: stating and solving a problem with an off-the-shelf toolkit
  - Notions of model and solver
  - I will not talk about user-defined propagator
  - I will not talk about search strategies (though there are things to do at the language level)

- The minimum about solving methods to allow for clever modeling
Avant propos

- Introduction to constraint programming (no pre-requisite)
  - Or almost none
  - Constraint programming = combinatorial branch & bound plus a lot of jargon

- Language-level modeling: stating and solving a problem with an off-the-shelf toolkit
  - Notions of model and solver
  - I will not talk about user-defined propagator
  - I will not talk about search strategies (though there are things to do at the language level)

- The minimum about solving methods to allow for clever modeling
  - It turns out, it is already a lot!
1 Language

2 Variables

3 Constraints

4 Modeling
   - Ex: Golomb Ruler
1 Language
2 Variables
3 Constraints
4 Modeling
Constraint Optimization Problem
Constraint Optimization Problem

- **Variables:** with finite discrete domains (e.g. $x \in \{2, 3, 5, 7, 11, 13\}$, $y \in [0, 100000]$)

Constraints: any relation between variables (e.g. $x = \sqrt{y} \mod 15$)

Objective: distinguished variable to minimize/maximize
Constraint Optimization Problem

- **Variables:** with finite discrete domains (e.g. $x \in \{2, 3, 5, 7, 11, 13\}$, $y \in [0, 100000]$)

- **Constraints:** any relation between variables (e.g. $x = (\sqrt{y} \mod 15)$)
Constraint Optimization Problem

- **Variables**: with finite discrete domains (e.g. $x \in \{2, 3, 5, 7, 11, 13\}, y \in [0, 100000]$)

- **Constraints**: any relation between variables (e.g. $x = (\sqrt{y} \mod 15)$)

- **Objective**: distinguished variable to minimize/maximize
Map Coloring
Map Coloring

blue
green

blue
red

blue
yellow
red
green
Map Coloring

\[ \mathcal{D}(x_f) : \text{blue, green} \]
\[ \mathcal{D}(x_s) : \text{blue, red} \]
\[ \mathcal{D}(x_e) : \text{blue, yellow, red, green} \]
\[ \mathcal{D}(x_i) : \text{blue, red} \]
from Numberjack import *

france = Variable(["blue","green"], "france")
switzerland = Variable(["blue","red"], "switzerland")
spain = Variable(["blue","yellow","red","green"], "spain")
italy = Variable(["blue","red"], "italy")

model = Model(
    france != switzerland,
    france != italy,
    france != spain,
    italy != switzerland
)

solver = model.load('Mistral2')

if solver.solve():
    for var in [france, switzerland, spain, italy]:
        print var.name(), 'in', var.get_value()
static final String[] colorname = {"red", "blue", "green", "yellow"};
static final Map<String, Integer> colorindex = new HashMap<String, Integer>();

public static void main(String[] args) {
    for(int i=0; i<colorname.length; ++i) colorindex.put(colorname[i], i);

    Model model = new Model("Map coloring example");

    IntVar france = model.intVar("france", new int[]{colorindex.get("blue"), colorindex.get("green")});
    IntVar switzerland = model.intVar("switzerland", new int[]{colorindex.get("blue"), colorindex.get("red")});
    IntVar spain = model.intVar("spain", new int[]{colorindex.get("blue"), colorindex.get("yellow"), colorindex.get("red"), colorindex.get("green")});
    IntVar italy = model.intVar("italy", new int[]{colorindex.get("blue"), colorindex.get("red")});

    model.arithm(france, "!=" , switzerland).post();
    model.arithm(france, "!=" , italy).post();
    model.arithm(france, "!=" , spain).post();
    model.arithm(italy, "!=" , switzerland).post();

    if(model.getSolver().solve()){
        for(IntVar x : new IntVar[]{france, switzerland, spain, italy})
            System.out.printf("%s in %s\n", x.getName(), color_name[x.getValue()]);
    }
}
Constraint Toolkits

Declare variables and their domains e.g.,
```
france = Variable(['blue','green'], 'france')
```

Declare constraints e.g.,
```
france != switzerland
```

▶ Among the constraints defined in the language/toolkit (or user-defined!)
▶ Linear constraints, arithmetic and logic operators (=, =, ≤, >, ∨, ∧, ⇒, %, ×, +, /, . . .)
▶ Some keyworded relations AllDifferent, Element, etc.
▶ Any Expression tree of the above
Declare **variables** and their **domains** e.g.,
france = Variable([‘blue’,’green’], ’france’)
Constraint Toolkits

- Declare **variables** and their **domains** e.g.,
  \[
  \text{france} = \text{Variable}([\text{'blue'}, \text{'green'}], \text{'france'})
  \]

- Declare **constraints** e.g.,
  \[
  \text{france} \neq \text{switzerland}
  \]
  - Among the constraints defined in the language/toolkit
Declare **variables** and their **domains** e.g.,
france = Variable([‘blue’,’green’], ’france’)

Declare **constraints** e.g., france != switzerland

- Among the constraints defined in the language/toolkit *(or user-defined!)*
Declare **variables** and their **domains** e.g.,
france = Variable(['blue', 'green'], 'france')

Declare **constraints** e.g., france != switzerland

- Among the constraints defined in the language/toolkit *(or user-defined!)*
- Linear constraints, arithmetic and logic operators (\(=, \neq, \leq, >, \lor, \land, \implies, \%, \times, +, /, \ldots\))
Declare **variables** and their **domains** e.g.,
france = Variable(['blue','green'], 'france')

Declare **constraints** e.g., france != switzerland

- Among the constraints defined in the language/toolkit *(or user-defined!)*
- Linear constraints, arithmetic and logic operators (=, ≠, ≤, ≥, ∨, ∧, ⇒, %, ×, +, /,...)
- Some keyworded relations **AllDifferent**, **Element**, etc.
Constraint Toolkits

- Declare **variables** and their **domains** e.g.,
  \[\text{france} = \text{Variable([‘blue’, ‘green’], ’france’)}\]

- Declare **constraints** e.g., \[\text{france} \neq \text{switzerland}\]
  
  - Among the constraints defined in the language/toolkit (or user-defined!)
  
  - Linear constraints, arithmetic and logic operators (\(=, \neq, \leq, >, \lor, \land, \implies, \%, \times, +, /, \ldots\))
  
  - Some keyworded relations **AllDifferent**, **Element**, etc.
  
  - Any **Expression tree** of the above
1 Language

2 Variables

3 Constraints

4 Modeling
Choice of representation

The same problem might be mapped to many models. The most important and fundamental choice is the choice of variable viewpoint. For example, in TSP:

- $x_{ij} \leftrightarrow$ do we use arc $(i, j)$?
- $x_i \leftrightarrow$ what is the $i$-th visited city?

Constraints follow from the choice of variable viewpoint. Sometimes the best choice is clear, but not always. Consider the graph coloring example.

Variables
The choice of representation

The same problem might be mapped to many models
Choice of representation

- The same problem might be mapped to many models

- The most important and fundamental choice is the choice of variable viewpoint [Barbara Smith]
  - TSP: \( x_{ij} \leftrightarrow \) do we use arc \((i, j)\)? or \( x_i \leftrightarrow \) what is the \(i\)-th visited city?

Constraints follow from the choice of variable viewpoint. Sometimes the best choice is clear, but not always. Consider the graph coloring example.
Choice of representation

- The same problem might be mapped to many models

- The most important and fundamental choice is the choice of variable viewpoint [Barbara Smith]
  
  ▶ TSP: $x_{ij} \leftrightarrow$ do we use arc $(i, j)$? or $x_i \leftrightarrow$ what it the $i$-th visited city?
  ▶ Constraints follow from the choice of variable viewpoint
The same problem might be mapped to many models.

The most important and fundamental choice is the choice of variable viewpoint [Barbara Smith]

- TSP: $x_{ij} \leftrightarrow$ do we use arc $(i, j)$? or $x_i \leftrightarrow$ what it the $i$-th visited city?
- Constraints follow from the choice of variable viewpoint

Sometimes the best choice is clear, but not always.
Choice of representation

- The same problem might be mapped to many models

- The most important and fundamental choice is the choice of variable viewpoint [Barbara Smith]
  - TSP: $x_{ij} \leftrightarrow$ do we use arc $(i, j)$? or $x_i \leftrightarrow$ what it the $i$-th visited city?
  - Constraints follow from the choice of variable viewpoint

- Sometimes the best choice is clear, but not always

- Consider the graph coloring example
Choice of representation

Zykov recurrence [Zykov 49]: take a non-edge $e$, $s$. In the optimal coloring:

- Either $e$ and $s$ take a different color, so adding the edge would not hurt.
- Or $e$ and $s$ take the same color, so merging them (adding an equality constraint) would not hurt.

Instead of assigning colors to nodes, we can assign $\{\neq, \equiv\}$ to non-edges.

No color symmetry anymore!

But stating the constraints is difficult.
Choice of representation

- Zykov recurrence [Zykov 49]: take a non-edge $e, s$. In the optimal coloring:

Variables
Choice of representation

- Zykov recurrence [Zykov 49]: take a non-edge \( e, s \). In the optimal coloring:
  - either \( e \) and \( s \) take a different color, so adding the edge would not hurt

Variables
Choice of representation

- Zykov recurrence [Zykov 49]: take a non-edge $e, s$. In the optimal coloring:
  - either $e$ and $s$ take a different color, so adding the edge would not hurt
  - or $e$ and $s$ take the same color, so merging them (adding an equality constraint) would not hurt
Choice of representation

- Zykov recurrence [Zykov 49]: take a non-edge $e, s$. In the optimal coloring:
  - either $e$ and $s$ take a different color, so adding the edge would not hurt
  - or $e$ and $s$ take the same color, so merging them (adding an equality constraint) would not hurt

- Instead of assigning colors to nodes, we can assign $\{=, \neq\}$ to non-edges
Choice of representation

- Zykov recurrence [Zykov 49]: take a non-edge $e, s$. In the optimal coloring:
  - either $e$ and $s$ take a different color, so adding the edge would not hurt
  - or $e$ and $s$ take the same color, so merging them (adding an equality constraint) would not hurt

- Instead of assigning colors to nodes, we can assign $\{=, \neq\}$ to non-edges

- No color symmetry anymore!
Choice of representation

- Zykov recurrence [Zykov 49]: take a non-edge $e, s$. In the optimal coloring:
  - either $e$ and $s$ take a different color, so adding the edge would not hurt
  - or $e$ and $s$ take the same color, so merging them (adding an equality constraint) would not hurt

- Instead of assigning colors to nodes, we can assign $\{=, \neq\}$ to non-edges
- No color symmetry anymore!
- But stating the constraints is difficult
The best variable viewpoint is the one that...
The best variable viewpoint is the one that...

...induces the smallest search tree
The best variable viewpoint is the one that...

- induces the smallest search tree
- induces the "best" set of constraints
The best variable viewpoint is the one that...

- ...induces the smallest search tree
- ...induces the "best" set of constraints

What is a good constraint set?
Outline

1. Language
2. Variables
3. Constraints
   - Expression tree
   - Global constraints
   - Constraint solving
4. Modeling
Combining constraints (logically)
Most logic operators

- can be used as a relation \((x \neq y)\)

Two different constraints:

\[ x \neq y \quad \text{and} \quad (x \neq y) \iff z \quad \text{(reification)} \]

\[(x \neq y) \implies y \leq 12 \]

Which you can write \((x \neq y) \implies y \leq 12\) (and let the system insert extra variables)
Most logic operators

- can be used as a relation \((x \neq y)\)...
- or as a predicate \(((x \neq y) \implies y \leq 12)\)
Combining constraints (logically)

- Most logic operators
  - can be used as a relation \((x \neq y)\)...
  - or as a predicate \(((x \neq y) \implies y \leq 12)\)

- Two different constraints: \(x \neq y\) and \((x \neq y) \iff z\) (reification)
Combining constraints (logically)

• Most logic operators
  ▶ can be used as a relation \((x \neq y)\)...
  ▶ or as a predicate \(((x \neq y) \implies y \leq 12)\)

• Two different constraints: \(x \neq y\) and \((x \neq y) \iff z\) (reification)

\[(x \neq y) \implies y \leq 12\] encoded as \((x \neq y) \iff z\)
\[z \implies (y \leq 12)\]

• Which you can write \((x \neq y) \implies y \leq 12\) (and let the system insert extra variables)
Combining constraints (functionally)
There are also function operators that must be combined similarly.

For instance \((|x - y| \times z) \leq (z + 12)\)

\[ (|x - y| \times z) \leq (z + 12) \quad \text{encoded as} \quad (x - y) = a_1 \]
\[ |a_1| = a_2 \]
\[ a_2 \times z = a_3 \]
\[ z + 12 = a_4 \]
\[ a_3 \leq a_4 \]
### Constraints - Root of the expression tree

- **C1** = \((X+Y < 5) \lor (X+3 < Y)\)
- **C2** = \(\text{AllDiff}([x,y,z])\)
- **C3** = \(\text{Sum}([a,b,c,d]) \geq e\)

### Predicates & functions - Internal nodes

- **P** = \(X+Y\) \# arithmetic value
- **Q** = \(X+3 \leq Y\) \# truth (logic) value

### Variables - Leaves of the expression tree

- **X** = \(\text{Variable}(0,10)\)
- **X** = \(\text{Variable}([1,3,5,7])\)
**My Hobby:**

Embedding NP-Complete Problems in Restaurant Orders

**Chotchkie's Restaurant**

<table>
<thead>
<tr>
<th>Appetizers</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mixed Fruit</td>
<td>2.15</td>
</tr>
<tr>
<td>French Fries</td>
<td>2.75</td>
</tr>
<tr>
<td>Side Salad</td>
<td>3.35</td>
</tr>
<tr>
<td>Hot Wings</td>
<td>3.55</td>
</tr>
<tr>
<td>Mozzarella Sticks</td>
<td>4.20</td>
</tr>
<tr>
<td>Sampler Plate</td>
<td>5.80</td>
</tr>
</tbody>
</table>

**Sandwiches**

<table>
<thead>
<tr>
<th>Barbecue</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6.55</td>
</tr>
</tbody>
</table>

We'd like exactly $15.05 worth of appetizers, please.

...Exactly? Uhh...

Here, these papers on the knapsack problem might help you out:

Listen, I have six other tables to get to—

As fast as possible, of course. Want something on traveling salesman?
from Numberjack import *

price = [215, 275, 335, 355, 420, 580]
appetizers = ['Mixed Fruit', 'French Fries', 'Side Salad',
            'Hot Wings', 'Mozzarella Sticks', 'Sample Plate']
total = 1505
num_appetizers = len(appetizers)

quantities = [Variable(0, 1505/price[i], '#'+appetizers[i])
              for i in range(num_appetizers)]

model = Model(
    Sum([quantities[i] * price[i] for i in range(num_appetizers)]) == total)

solver = model.load('Mistral2')
solver.startNewSearch()

while solver.getNextSolution() == SAT:
    print "\nSOLUTION:\n", "\n".join("%s x %s ($%.2lf)") % (quantities[i], \
            appetizers[i], price[i] / 100.0) for i in xrange(num_appetizers)"
XKCD Knapsack

\[
\sum [\text{quantities}[i] \times \text{price}[i] \text{ for } i \text{ in range(num_appetizers)]} = \text{total}
\]
XKCD Knapsack

\[\text{Sum([quantities[i] \times price[i] for i in range(num_appetizers)])} == \text{total}\]
Solution

Solution 1:

7 × Mixed Fruit ($2.15)
0 × French Fries ($2.75)
0 × Side Salad ($3.35)
0 × Hot Wings ($3.55)
0 × Mozzarella Sticks ($4.20)
0 × Sample Plate ($5.80)

Solution 2:

1 × Mixed Fruit ($2.15)
0 × French Fries ($2.75)
0 × Side Salad ($3.35)
2 × Hot Wings ($3.55)
0 × Mozzarella Sticks ($4.20)
1 × Sample Plate ($5.80)
Global constraints

- CP languages contain a number of keywords for specific relations on variables

- AllDifferent

\[ (x_1, \ldots, x_n) \Leftrightarrow \forall 1 \leq i < j \leq n \ x_i \neq x_j \]

- Element

\[ (x_0, \ldots, x_{n-1}, y, z) \Leftrightarrow x_y = z \]

- Constraints
CP languages contain a number of keywords for specific relations on variables

**AllDifferent**

\[\text{AllDifferent}(x_1, \ldots, x_n) \iff \forall 1 \leq i < j \leq n \ x_i \neq x_j\]
Global constraints

- CP languages contain a number of keywords for specific relations on variables

**AllDifferent**

\[
\text{AllDifferent}(x_1, \ldots, x_n) \iff \forall 1 \leq i < j \leq n \; x_i \neq x_j
\]

\(\bar{x} = 3, 5, 1, 2, 7\) satisfies AllDifferent

\(\bar{x} = 3, 5, 1, 2, 5\) does not satisfy AllDifferent
Global constraints

- CP languages contain a number of keywords for specific relations on variables

**AllDifferent**

\[ \text{AllDifferent}(x_1, \ldots, x_n) \iff \forall 1 \leq i < j \leq n \ x_i \neq x_j \]

\( \bar{x} = 3, 5, 1, 2, 7 \) satisfies AllDifferent

\( \bar{x} = 3, 5, 1, 2, 5 \) does not satisfy AllDifferent

**Element**

\[ \text{Element}(x_0, \ldots, x_{n-1}, y, z) \iff x_y = z \]
Global constraints

- CP languages contain a number of keywords for specific relations on variables

**AllDifferent**

\[ \text{AllDifferent}(x_1, \ldots, x_n) \iff \forall 1 \leq i < j \leq n \ x_i \neq x_j \]

\( \bar{x} = 3, 5, 1, 2, 7 \) satisfies AllDifferent

\( \bar{x} = 3, 5, 1, 2, 5 \) does not satisfy AllDifferent

**Element**

\[ \text{Element}(x_0, \ldots, x_{n-1}, y, z) \iff x_y = z \]

\( \bar{x} = 3, 5, 1, 2, 5, y = 1, z = 5 \) satisfies Element

\( \bar{x} = 3, 5, 1, 2, 5, y = 2, z = 5 \) does not satisfy Element
Map Coloring

\[ \mathcal{D}(x_f) : \begin{array}{c} \text{blue} \\ \text{green} \end{array} \quad \mathcal{D}(x_s) : \begin{array}{c} \text{blue} \\ \text{red} \end{array} \]

\[ \mathcal{D}(x_e) : \begin{array}{c} \text{blue} \\ \text{yellow} \\ \text{red} \\ \text{green} \end{array} \quad \mathcal{D}(x_i) : \begin{array}{c} \text{blue} \\ \text{red} \end{array} \]
Map Coloring

\[ \mathcal{D}(x_f) : \quad \text{blue} \quad \text{green} \]
\[ \mathcal{D}(x_s) : \quad \text{blue} \quad \text{red} \]
\[ \mathcal{D}(x_e) : \quad \text{blue} \quad \text{yellow} \quad \text{red} \quad \text{green} \]
\[ \mathcal{D}(x_i) : \quad \text{blue} \quad \text{red} \]

Alldifferent
Constraint solver

Search

Develop a search tree (depth first).

Select a variable $x$, a value $v$ in its domain and branch on $x = v$ or $x \neq v$.

Inference

At every node of the tree, the domains of the variables are reduced.

Every constraint makes local deductions.

Consistent iff every value of every variable is in a support.

Domain reductions from a constraint might trigger reduction by another constraint.

Constraint propagation.
**Search**

Develop a search tree (depth first).

- Select a variable $x$, a value $v$ in its domain and branch on $x = v$ or $x \neq v$
Constraint solver

**Search**

Develop a search tree (depth first).

- Select a variable $x$, a value $v$ in its domain and branch on $x = v$ or $x \neq v$

**Inference**

At every node of the tree, the **domains** of the variables are reduced

- Every constraint makes local deductions
Search

Develop a search tree (depth first).

- Select a variable $x$, a value $v$ in its domain and branch on $x = v$ or $x \neq v$

Inference

At every node of the tree, the domains of the variables are reduced

- Every constraint makes local deductions
  
  Consistent iff every value of every variable is in a support

- Domain reductions from a constraint might trigger reduction by another constraint
Search

Develop a search tree (depth first).

- Select a variable $x$, a value $v$ in its domain and branch on $x = v$ or $x \neq v$

Inference

At every node of the tree, the domains of the variables are reduced

- Every constraint makes local deductions

  **Consistent** iff every value of every variable is in a support

- Domain reductions from a constraint might trigger reduction by another constraint

  **constraint propagation**
Example: binary constraint

What inference can the inequality $x_f \neq x_e$ make?

A support: a value $v \in D(x_f)$ and a value $w \in D(x_e)$ with $v \neq w$.

Propagation of $x_f \neq x_e$:

As long as the domain $D(x_f)$ has two distinct values, then $x_e$ could take any value $x_f \in \{b, r\}$, $x_e \in \{b, r, g\}$: there is no correct domain reduction.

If $D(x_f) = \{v\}$ then $x_e$ cannot take the value $v$.

$x_f \in \{b\}$, $x_e \in \{b, r, g\}$ $\Rightarrow$ $x_f \in \{b\}$, $x_e \in \{r, g\}$.
Example: binary constraint

What inference can the inequality $x_f \neq x_e$ make?
Example: binary constraint

- What inference can the inequality $x_f \neq x_e$ make?
- A support: a value $v \in D(x_f)$ and a value $w \in D(x_e)$ with $v \neq w$
Example: binary constraint

- What inference can the inequality \( x_f \neq x_e \) make?
- A support: a value \( v \in D(x_f) \) and a value \( w \in D(x_e) \) with \( v \neq w \)

**Propagation of** \( x_f \neq x_e \)

- As long as the domain \( D(x_f) \) has two distinct values, then \( x_e \) could take any value.
- \( x_f \in \{b, r\}, x_e \in \{b, r, g\} \): there is no correct domain reduction.
Example: binary constraint

- What inference can the inequality $x_f \neq x_e$ make?
- A support: a value $v \in \mathcal{D}(x_f)$ and a value $w \in \mathcal{D}(x_e)$ with $v \neq w$

### Propagation of $x_f \neq x_e$

- As long as the domain $\mathcal{D}(x_f)$ has two distinct values, then $x_e$ could take any value
- $x_f \in \{b, r\}, x_e \in \{b, r, g\}$: there is no correct domain reduction
- If $\mathcal{D}(x_f) = \{v\}$ then $x_e$ cannot take the value $v$
- $x_f \in \{b\}, x_e \in \{b, r, g\} \implies x_f \in \{b\}, x_e \in \{r, g\}$
Search Tree

$$
\begin{align*}
xf & \in \{b, g\} \quad x_s \in \{b, r\} \\
\quad & \quad \\
xe & \in \{b, r, g, y\} \quad xi \in \{b, r\}
\end{align*}
$$
Search Tree

\[ x_f \in \{b, g\} - x_s \in \{b, r\} \]
\[ x_e \in \{b, r, g, y\} \quad x_i \in \{b, r\} \]

\[ x_f = b \]
Search Tree

\[ x_f \in \{b, g\} \quad x_s \in \{b, r\} \quad x_i \in \{b, r\} \]

\[ x_e \in \{b, r, g, y\} \]

\[ x_f = b \]

\[ x_f \in \{b\} \quad x_s \in \{r\} \quad x_i \in \{r\} \]

\[ x_e \in \{r, g, y\} \]
$x_f \in \{b, g\} - x_s \in \{b, r\}$

$\downarrow$

$e \in \{b, r, g, y\}$  

$s \in \{b, r\}$

$\downarrow$

$i \in \{b, r\}$

$x_f = b$

$\downarrow$

$e \in \{r, g, y\}$

$s \in \{r\}$

$\downarrow$

$i \in \{\}$
Search Tree

\[ x_f \in \{b, g\} \quad x_s \in \{b, r\} \]
\[ x_e \in \{b, r, g, y\} \quad x_i \in \{b, r\} \]

\[ x_f = b \]
\[ x_f \neq b \]

\[ x_f \in \{b\} \quad x_s \in \{r\} \]
\[ x_e \in \{r, g, y\} \quad x_i \in \{\} \]

\[ x_f \in \{g\} \quad x_s \in \{b, r\} \]
\[ x_e \in \{b, r, y\} \quad x_i \in \{b, r\} \]

Constraints
Search Tree

\[ x_f \in \{b, g\} \quad x_s \in \{b, r\} \]
\[ x_e \in \{b, r, g, y\} \quad x_i \in \{b, r\} \]
\[ x_f = b \]
\[ x_f \neq b \]

Fail!

\[ x_f \in \{b\} \quad x_s \in \{r\} \]
\[ x_e \in \{r, g, y\} \quad x_i \in \{\} \]

\[ x_f \in \{g\} \quad x_s \in \{b, r\} \]
\[ x_e \in \{b, r, y\} \quad x_i \in \{b, r\} \]
\[ x_s = b \]
\[ x_f \in \{g\} \quad x_s \in \{b\} \]
\[ x_e \in \{b, r, y\} \quad x_i \in \{r\} \]
Example: global constraint

Constraints
Example: global constraint

$$x_f \neq x_s \neq x_i$$

$$x_f \in \{b, g\}$$
$$x_s \in \{b, r\}$$
$$x_i \in \{b, r\}$$
Example: global constraint

\[ x_f \neq x_s \neq x_i \]

\[ x_f \in \{b, g\} \]
\[ x_s \in \{b, r\} \]
\[ x_i \in \{b, r\} \]

- Every inequality is consistent

Constraints
Example: global constraint

\[ x_f \neq x_s \neq x_i \]

- Every inequality is consistent
- AllDifferent is not consistent!

Propagation of $\text{AllDifferent}(\overline{x})$

- A support is a perfect matching in the graph

\[ x_f \rightarrow g \]
\[ x_s \rightarrow b \]
\[ x_i \rightarrow r \]
Example: global constraint

\[ x_f \neq x_s \neq x_i \]

\( x_f \in \{b, g\} \)
\( x_s \in \{b, r\} \)
\( x_i \in \{b, r\} \)

- Every inequality is consistent
- AllDifferent is not consistent!

Propagation of \textbf{AllDifferent}(\(\overline{x}\))

- A support is a perfect matching in the graph
- The edge \((x_f, b)\) does not belong to any perfect matching
- \textbf{AllDifferent}\((x_f, x_s, x_i)\) is consistent for \(x_f \in \{g\} \ x_s \in \{b, r\} \ x_i \in \{b, r\}\)
Search Tree (AllDifferent)

\[ x_f \in \{b, g\} \quad \text{and} \quad x_s \in \{b, r\} \]

\[ x_e \in \{b, r, g, y\} \quad x_i \in \{b, r\} \]
Search Tree (AllDifferent)

\[ x_f \in \{ g \} - x_s \in \{ b, r \} \]

\[ x_e \in \{ b, r, g, y \} \quad x_i \in \{ b, r \} \]
Search Tree (AllDifferent)

\[
\begin{align*}
    x_f & \in \{g\} - x_s \in \{b, r\} \\
    x_e & \in \{b, r, g, y\} \\
    x_i & \in \{b, r\} \\
\end{align*}
\]

Constraints
Propagation algorithm

- Every constraint has a propagation algorithm

Arc consistency
Every possible deduction w.r.t a single constraint on its variable’s domain

- For every value $v$ of every variable $x$
  - Does there exist a support for $x = v$ (a solution of the constraint involving $x = v$)?
  - Otherwise, remove $v$ from $D(x)$

The bigger (more global) the stronger! (and the slower...)
Propagation algorithm

- Every constraint has a propagation algorithm

- How do we know what inference we can expect from a propagation algorithm?
Propagation algorithm

- Every constraint has a propagation algorithm

- How do we know what inference we can expect from a propagation algorithm?

Arc consistency

Every possible deduction w.r.t a single constraint on its variable’s domain
Propagation algorithm

- Every constraint has a propagation algorithm

- How do we know what inference we can expect from a propagation algorithm?

Arc consistency

Every possible deduction w.r.t a single constraint on its variable’s domain

- For every value $v$ of every variable $x$
Propagation algorithm

- Every constraint has a **propagation algorithm**

- How do we know what inference we can expect from a propagation algorithm?

---

**Arc consistency**

Every **possible deduction w.r.t a single constraint on its variable’s domain**

- For every value $v$ of every variable $x$
  
  - Does there exist a **support** for $x = v$ (a solution of the constraint involving $x = v$)
  
  - Otherwise, **remove** $v$ from $D(x)$
Propagation algorithm

- Every constraint has a propagation algorithm

- How do we know what inference we can expect from a propagation algorithm?

Arc consistency

Every possible deduction w.r.t a single constraint on its variable’s domain

- For every value $v$ of every variable $x$
  - Does there exist a support for $x = v$ (a solution of the constraint involving $x = v$)
  - Otherwise, remove $v$ from $D(x)$

- The bigger (more global) the stronger!
Propagation algorithm

- Every constraint has a propagation algorithm

- How do we know what inference we can expect from a propagation algorithm?

Arc consistency

Every possible deduction w.r.t a single constraint on its variable’s domain

- For every value $v$ of every variable $x$
  - Does there exist a support for $x = v$ (a solution of the constraint involving $x = v$)
  - Otherwise, remove $v$ from $D(x)$

- The bigger (more global) the stronger! (and the slower...)
1. Language

2. Variables

3. Constraints

4. Modeling
   - Ex: Golomb Ruler
The art of modeling

Techniques to strengthen propagation

- Common sub-expressions
- Global constraints
- Implied constraints
- Symmetry breaking
- Dominance
Problem definition

- Place $m$ marks on a ruler
- Distance between each pair of marks is different
- Goal is to minimise the size of the ruler
- Proposed by Sidon [1932] then independently by Golomb and Babcock
import sys
from Numberjack import *

m = int(sys.argv[1]) if len(sys.argv)>1 else 6
n = 2 ** (m - 1)

marks = VarArray(m, n, 'm')
distance = [Abs(marks[i] - marks[j]) for i in range(1, m) for j in range(i)]

model = Model(
    Minimise(Max(marks)),  # objective function
    [m1 != m2 for m1,m2 in pair_of(marks)],
    [d1 != d2 for d1,d2 in pair_of(distance)]
)

solver = model.load('Mistral2', marks)
if solver.solve():
    print marks, [d.get_value() for d in distance]
A First Model (Choco)

```java
Model model = new Model();

IntVar[] marks = model.intVarArray("m", m, 0, n);
IntVar[] distance = model.intVarArray("d", m * (m - 1) / 2, 1, n);

int k = 0;
for(int i=0; i<m; ++i) {
    for(int j=i+1; j<m; ++j) {
        model.distance(marks[i], marks[j], ",", distance[k++]).post();
        model.arithm(marks[i], ",!=", marks[j]).post();
    }
}

for(int i=0; i<distance.length; ++i)
    for(int j=i+1; j<distance.length; ++j)
        model.arithm(distance[i], ",!=", distance[j]).post();

IntVar objective = model.intVar("obj", 0, n);
model.max(objective, marks).post();
model.setObjective(Model.MINIMIZE, objective);
```
An **objective** variable

```java
model.setObjective(Model.MINIMIZE, objective);
```
An objective variable

```java
model.setObjective(Model.MINIMIZE, objective);
```

The upper bound is updated when a new solution is found
An objective variable

```java
model.setObjective(Model.MINIMIZE, objective);
```

- The upper bound is updated when a new solution is found

- The lower bound is maintained via constraint propagation

```java
model.max(objective, marks).post();
```
Branch & Bound

- An **objective** variable
  
  ```java
  model.setObjective(Model.MINIMIZE, objective);
  ```

- The **upper bound** is updated when a new solution is found

- The **lower bound** is maintained via constraint propagation
  
  ```java
  model.max(objective, marks).post();
  ```

- Different models may entail different lower bounds for the same objective function
import sys
from Numberjack import *

m = int(sys.argv[1]) if len(sys.argv)>1 else 6
n = 2 ** (m - 1)

marks = VarArray(m, n, 'm')
distance = [Abs(marks[i] - marks[j]) for i in range(m-1) for j in range(i+1,m)]

model = Model(
    Minimise(Max(marks)), # objective function
    AllDiff(marks),
    AllDiff(distance)
)

solver = model.load('Mistral2', marks)
if solver.solve():
    print marks, [d.get_value() for d in distance]
Model model = new Model();

IntVar[] marks = model.intVarArray("m", m, 0, n);
IntVar[] distance = model.intVarArray("d", m * (m - 1) / 2, 1, n);

int k = 0;
for(int i=0; i<m; ++i)
    for(int j=i+1; j<m; ++j)
        model.distance(marks[i], marks[j], "," , distance[k++]).post();

model.allDifferent(marks).post();
model.allDifferent(distance).post();

IntVar objective = model.intVar("obj", 0, n);
model.max(objective, marks).post();

model.setObjective(Model.MINIMIZE, objective);
Symmetry breaking

Solution symmetries

⇒ symmetric (suboptimal) branches in the search tree

Variable symmetries:

- marks
- distance

We can swap the marks or the distances of a solution (but not both)

Force an arbitrary ordering

⋆ marks[1] < marks[2] < ... < marks[m]

Distances are still symmetric by reflection

⋆ distance[0,1] < distance[m−2, m−1]
Symmetry breaking

- Solution symmetries $\Rightarrow$ symmetric (suboptimal) branches in the search tree
Solution symmetries ⇒ symmetric (suboptimal) branches in the search tree

Variable symmetries: marks, distance
Symmetry breaking

- Solution symmetries $\Rightarrow$ symmetric (suboptimal) branches in the search tree

- Variable symmetries: marks, distance
  - We can swap the marks or the distances of a solution (but not both)
Symmetry breaking

- Solution symmetries ⇒ symmetric (suboptimal) branches in the search tree

- Variable symmetries: marks, distance

- We can swap the marks or the distances of a solution (but not both)

- Force an arbitrary ordering
  - marks[1] < marks[2] < ... < marks[m]
Symmetry breaking

- Solution symmetries $\Rightarrow$ symmetric (suboptimal) branches in the search tree

- Variable symmetries: marks, distance
  - We can swap the marks or the distances of a solution (but not both)
  - Force an arbitrary ordering
    - $\star$ marks[1] < marks[2] < ... < marks[m]
  - Distances are still symmetric by reflection
Symmetry breaking

- Solution symmetries $\Rightarrow$ symmetric (suboptimal) branches in the search tree

- Variable symmetries: marks, distance
  - We can swap the marks or the distances of a solution (but not both)
  - Force an arbitrary ordering
    $\star$ marks[1] < marks[2] < ... < marks[m]
  - Distances are still symmetric by reflection
    $\star$ distance[0,1] < distance[m − 2, m − 1]
Symmetry breaking (Numberjack)

```python
import sys
from Numberjack import *

m = int(sys.argv[1]) if len(sys.argv)>1 else 6
n = 2 ** (m - 1)

marks = VarArray(m, n, 'm')
distance = [marks[j] - marks[i] for i in range(m-1) for j in range(i+1,m)]

model = Model(
    Minimise(marks[-1]),  # objective function
    [marks[i-1] < marks[i] for i in range(1, m)],
    marks[0] == 0,
    distance[0] < distance[-1],
    AllDiff(distance)
)

solver = model.load('Mistral2', marks)
solver.setHeuristic('MinDomainMinVal');
if solver.solve():
    print marks, [d.get_value() for d in distance]
```
Model model = new Model();

IntVar[] marks = model.intVarArray("m", m, 0, n);
IntVar[] distance = model.intVarArray("d", m * (m - 1) / 2, 1, n);

int k = 0;
for(int i=0; i<m-1; ++i) {
    model.arithm(marks[i], "<", marks[i+1]).post();
    for(int j=i+1; j<m; ++j)
        model.scalar(new IntVar[]{marks[i], marks[j]}, new int[]{1, -1}, "=", distance[k++]).post();
    model.arithm(marks[0], "=", 0).post();
    model.arithm(distance[0], "<", distance[distance.length-1]).post();
}

model.allDifferent(distance).post();

model.setObjective(Model.MINIMIZE, marks[m-1]);
Implied Constraints

Implied constraint
Implied by the model, does not change the set of solutions

- $x \neq y$, $y \neq z$, $x \neq z = \Rightarrow \text{AllDifferent}(x, y, z)$
- $x \neq y$, $x \leq y = \Rightarrow x < y$

Let $x \in \{1, \ldots, 10\}$, $y \in \{1, \ldots, 10\}$
$x \neq y$ is consistent ($x = 10$ has $\langle 10, 9 \rangle$ as support)
$x \leq y$ is consistent ($x = 10$ has $\langle 10, 10 \rangle$ as support)
$x < y$ is inconsistent

consistent with $x \in \{1, \ldots, 9\}$, $y \in \{2, \ldots, 10\}$
Implied constraint

Implied by the model, does not change the set of solutions, ex:

- $x \neq y, y \neq z, x \neq z \implies \text{AllDifferent}(x, y, z)$
- $x \neq y, x \leq y \implies x < y$
**Implied constraint**

Implied by the model, does not change the set of solutions, ex:

1. \( x \neq y, y \neq z, x \neq z \implies \text{AllDifferent}(x, y, z) \)
2. \( x \neq y, x \leq y \implies x < y \)

Let \( x \in \{1, \ldots, 10\}, y \in \{1, \ldots, 10\} \)
**Implied constraint**

Implied by the model, does not change the set of solutions, ex:

- \( x \neq y, y \neq z, x \neq z \implies \text{AllDifferent}(x, y, z) \)
- \( x \neq y, x \leq y \implies x < y \)

Let \( x \in \{1, \ldots, 10\}, y \in \{1, \ldots, 10\} \)

- \( x \neq y \) is consistent (\( x = 10 \) has \( \langle 10, 9 \rangle \) as support)
**Implied constraints**

Implied by the model, does not change the set of solutions, ex:

- \( x \neq y, y \neq z, x \neq z \implies \text{AllDifferent}(x, y, z) \)
- \( x \neq y, x \leq y \implies x < y \)

Let \( x \in \{1, \ldots, 10\}, y \in \{1, \ldots, 10\} \)

- \( x \neq y \) is consistent (\( x = 10 \) has \( \langle 10, 9 \rangle \) as support)
- \( x \leq y \) is consistent (\( x = 10 \) has \( \langle 10, 10 \rangle \) as support)
Implied Constraints

**Implied constraint**

Implied by the model, does not change the set of solutions, ex:

- $x \neq y, y \neq z, x \neq z \implies \text{AllDifferent}(x, y, z)$
- $x \neq y, x \leq y \implies x < y$

Let $x \in \{1, \ldots, 10\}, y \in \{1, \ldots, 10\}$

- $x \neq y$ is consistent ($x = 10$ has $\langle 10, 9 \rangle$ as support)
- $x \leq y$ is consistent ($x = 10$ has $\langle 10, 10 \rangle$ as support)
- $x < y$ is inconsistent
**Implied constraint**

Implied by the model, does not change the set of solutions, ex:

- $x \neq y, y \neq z, x \neq z \implies \text{AllDifferent}(x, y, z)$
- $x \neq y, x \leq y \implies x < y$

Let $x \in \{1, \ldots, 10\}, y \in \{1, \ldots, 10\}$

- $x \neq y$ is consistent ($x = 10$ has $\langle 10, 9 \rangle$ as support)
- $x \leq y$ is consistent ($x = 10$ has $\langle 10, 10 \rangle$ as support)
- $x < y$ is inconsistent
  - consistent with $x \in \{1, \ldots, 9\}, y \in \{2, \ldots, 10\}$
Implied Constraints: Golomb Ruler

The distances are all different

- \( \text{distance}[i,j] \geq (j - i) \times (j - i + 1) / 2 \)

- \( \text{distance}[i,j] \leq \text{marks}[m] - \text{sum of } m - 1 - j + i \text{ distances} \)
Implied Constraints: Golomb Ruler

- distance\[i,j\] ≥ sum of \(j - i\) distances
Implied Constraints: Golomb Ruler

- \( \text{distance}[i,j] \geq \text{sum of } j - i \text{ distances} \)

- The distances are all different
Implied Constraints: Golomb Ruler

- distance[i,j] ≥ sum of j − i distances

- The distances are all different
Implied Constraints: Golomb Ruler

- $\text{distance}[i,j] \geq \text{sum of } j - i \text{ distances}$

- The distances are all different $\text{distance}[i,j] \geq (j - i) * (j - i + 1)/2$
Implied Constraints: Golomb Ruler

- \( \text{distance}[i,j] \geq \text{sum of } j - i \text{ distances} \)

- The distances are all different \( \text{distance}[i,j] \geq (j - i) \times (j - i + 1)/2 \)

- Same reasoning from the end (\( \text{marks}[m-1] \))
  - \( \text{distance}[i,j] \leq \text{marks}[m] - \text{sum of } m - 1 - j + i \text{ distances} \)
Implied Constraints: Golomb Ruler

- distance[i,j] ≥ sum of j − i distances

- The distances are all different distance[i,j] ≥ (j − i) * (j − i + 1)/2

- Same reasoning from the end (marks[m − 1])
  - distance[i,j] ≤ marks[m] − sum of m − 1 − j + i distances
  - distance[i,j] ≤ marks[m] − (m − 1 − j + i) * (m − j + i)/2
Implied Constraints: Golomb Ruler

- Implied constraints
  - \( \text{distance}[i,j] \geq (j - i) * (j - i + 1)/2 \)
  - \( \text{distance}[i,j] \leq \text{marks}[m] - (m - 1 - j + i) * (m - j + i)/2 \)
Implied Constraints: Golomb Ruler

- Implied constraints
  - \( \text{distance}[i,j] \geq (j - i) \times (j - i + 1)/2 \)
  - \( \text{distance}[i,j] \leq \text{marks}[m] - (m - 1 - j + i) \times (m - j + i)/2 \)

- How do we know that these constraints are useful (improving constraint propagation)
Implied Constraints: Golomb Ruler

- Implied constraints
  - \( \text{distance}[i,j] \geq (j - i) \cdot (j - i + 1)/2 \)
  - \( \text{distance}[i,j] \leq \text{marks}[m] - (m - 1 - j + i) \cdot (m - j + i)/2 \)

- How do we know that these constraints are useful (improving constraint propagation)

- We need to combine the reasoning of two constraints (AllDifferent(distance) and \( \text{distance}[i,j] = \sum_{k=i}^{j-1} \text{distance}[k,k+1] \))
Implied Constraints: Golomb Ruler

- Implied constraints
  - \( \text{distance}[i,j] \geq (j - i) \times (j - i + 1)/2 \)
  - \( \text{distance}[i,j] \leq \text{marks}[m] - (m - 1 - j + i) \times (m - j + i)/2 \)

- How do we know that these constraints are useful (improving constraint propagation)

- We need to combine the reasoning of two constraints (\text{AllDifferent}(\text{distance}) and \text{distance}[i,j] = \sum_{k=i}^{j-1} \text{distance}[k,k+1])

- Domain reduction is not sufficient to "communicate" between the two constraints
  - The implied constraints reduce the domains at the root node
Implied Constraints: Golomb Ruler

- Implied constraints
  - \( \text{distance}[i,j] \geq (j - i) \times (j - i + 1)/2 \)
  - \( \text{distance}[i,j] \leq \text{marks}[m] - (m - 1 - j + i) \times (m - j + i)/2 \)

- How do we know that these constraints are useful (improving constraint propagation)

- We need to combine the reasoning of two constraints (\texttt{AllDifferent(distance)} and \( \text{distance}[i,j] = \sum_{k=i}^{j-1} \text{distance}[k,k+1] \))

- Domain reduction is not sufficient to “communicate” between the two constraints
  - The implied constraints reduce the domains at the root node

- In doubt, just try!
```python
import sys
from Numberjack import *

m = int(sys.argv[1]) if len(sys.argv)>1 else 6
n = 2 ** (m - 1)

marks = VarArray(m, n, 'm')
dmap = dict([(i,j), marks[j] - marks[i]) for i in range(m-1) for j in range(i+1,m)])
distance = [dmap[(i,j)] for i in range(m-1) for j in range(i+1,m)]

lbs = [(j - i) * (j - i + 1) / 2 for i in range(m-1) for j in range(i+1,m)]
ubs = [marks[-1] - (m - 1 - j + i) * (m - j + i) / 2 for i in range(m-1) for j in range(i+1,m)]

model = Model(
    Minimise(marks[-1]),  # objective function
    [marks[i-1] < marks[i] for i in range(1, m)],
    marks[0] == 0,
    distance[0] < distance[-1],
    AllDiff(distance),
    [d >= l for d,l in zip(distance, lbs)],
    [d <= u for d,u in zip(distance, ubs)],
    [dmap[(i,j)] == dmap[(i,j-1)] + dmap[(j-1,j)] for i in range(m-2) for j in range(i+2,m)]
)

solver = model.load('Mistral2',marks)
if solver.solve():
    print marks, [d.get_value() for d in distance]
```

**Implied Constraints (Numberjack)**
Implied Constraints (Choco)

```java
Model model = new Model();

IntVar[] marks = model.intVarArray("m", m, 0, n);
IntVar[] distance = model.intVarArray("d", m * (m - 1) / 2, 1, n);
IntVar[][] dmap = new IntVar[m][m];

int k = 0;
for (int i = 0; i < m - 1; ++i) {
    model.arithm(marks[i], "<", marks[i + 1]).post();
    for (int j = i + 1; j < m; ++j) {
        dmap[i][j] = distance[k];
        model.arithm(distance[k], "<=",
                      marks[m - 1], "-",
                      ((m - 1 - j + i) * (m - j + i)) / 2).post();
        model.arithm(distance[k], ">=",
                      (j - i) * (j - i + 1) / 2).post();
        model.scalar(new IntVar[]{marks[i], marks[j]},
                      new int[]{1, -1}, ",=",
                      distance[k++]).post();
    }
    model.arithm(marks[0], "=", 0).post();
    model.arithm(distance[0], "<",
                  distance[distance.length - 1]).post();
}

for (int i = 0; i < m - 2; ++i)
    for (int j = i + 2; j < m; ++j)
        model.arithm(dmap[i][j], "=",
                      dmap[i][j - 1], "+",
                      dmap[j - 1][j]).post();

model.allDifferent(distance).post();
model.setObjective(Model.MINIMIZE, marks[m - 1]);
```

Modeling
Conclusions

Good modeling practices
- What are the variables, what are the values?
- Constraints will follow
- Defines the shape of the search tree

Key principle: strengthen constraint propagation

- Global constraints
- Implied constraints
- Symmetry breaking
Conclusions

Good modeling practices
Conclusions

Good modeling practices

- What are the variables, what are the values?
Conclusions

Good modeling practices

- What are the variables, what are the values?
  - Constraints will follow
Good modeling practices

- What are the variables, what are the values?
  - Constraints will follow
  - Defines the shape of the search tree
Conclusions

Good modeling practices

- What are the variables, what are the values?
  - Constraints will follow
  - Defines the shape of the search tree

- Key principle: *strengthen constraint propagation*
  - Global constraints
  - Implied constraints
  - Symmetry breaking
Master class on hybrid optimisation Toulouse
June 4th and 5th

Pierre Bonami (Université d’Aix-Marseille) Mixed-Integer Linear and Nonlinear Programming Methods

Willem Jan van Hoeve (Carnegie Mellon University) Decision diagrams for Discrete Optimization, Constraint programming, and Integer Programming


Paul Shaw (IBM Research) Combinations of local search and constraint programming

Laurent Simon (Université de Bordeaux) Understanding, using and extending SAT solvers
Master class on hybrid optimisation Toulouse
June 4th and 5th

Pierre Bonami  (Université d’Aix-Marseille) **Mixed-Integer Linear and Nonlinear Programming Methods**

Willem Jan van Hoeve  (Carnegie Mellon University) **Decision diagrams for Discrete Optimization, Constraint programming, and Integer Programming**

John Hooker  (Carnegie Mellon University) **Hybrid Mixed-Integer Programming / Constraint Programming Methods**

Paul Shaw  (IBM Research) **Combinations of local search and constraint programming**

Laurent Simon  (Université de Bordeaux) **Understanding, using and extending SAT solvers**

Free registration, students’ accommodation covered!