Modeling and Solving Constraint Problems

Emmanuel Hebrard

Avant propos

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

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

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!

Outline

1. **Language**
2. **Variables**
3. **Constraints**
4. **Modeling**
   - Ex: Golomb Ruler

Outline

1. **Language**
2. Variables
3. Constraints
4. Modeling

Constraint Optimization Problem

Constraint Optimization Problem

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

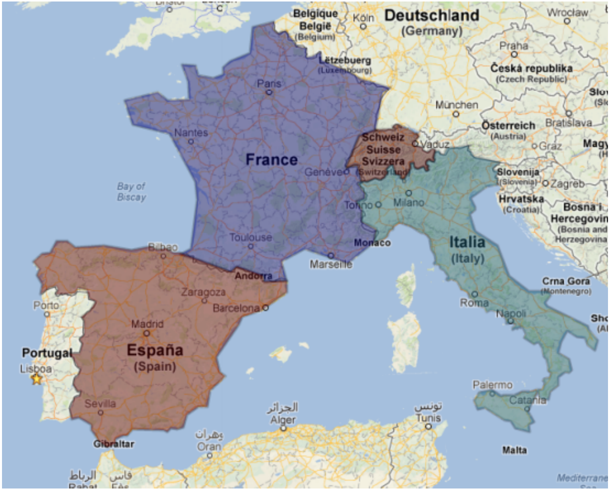
Constraint Optimization Problem

- 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. $x = (\sqrt{y} \bmod 15)$)

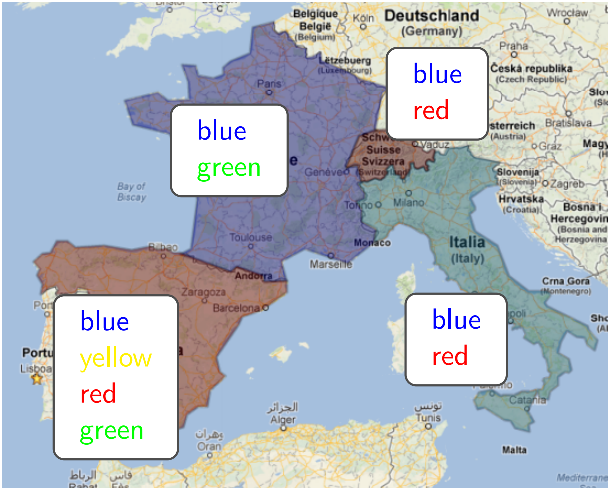
Constraint Optimization Problem

- 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. $x = (\sqrt{y} \bmod 15)$)
- Objective: distinguished variable to minimize/maximize

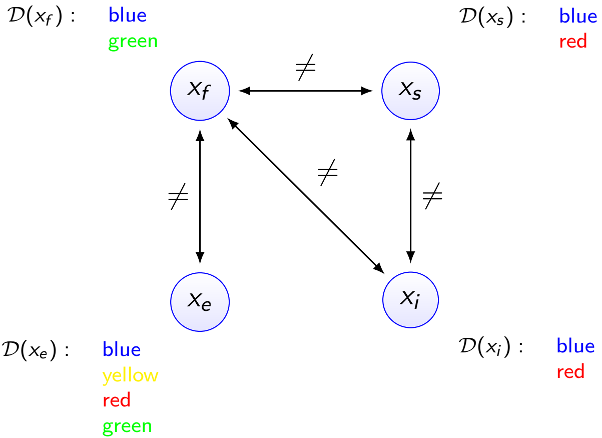
Map Coloring



Map Coloring



Map Coloring



Map Coloring (Numberjack)

```
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()
```

Map Coloring (Choco)

```
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"), colori
	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

Constraint Toolkits

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

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

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!*)

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 ($=, \neq, \leq, >, \vee, \wedge, \implies, \%, \times, +, /, \ldots$)

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 $(=, \neq, \leq, >, \vee, \wedge, \implies, \%, \times, +, /, \ldots)$
  - Some keyworded relations AllDifferent, Element, etc.

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 ($=, \neq, \leq, >, \vee, \wedge, \implies, \%, \times, +, /, \ldots$)
  - Some keyworded relations AllDifferent, Element, etc.
  - Any Expression tree of the above

Outline

Choice of representation

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 it the i-th visited city?

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

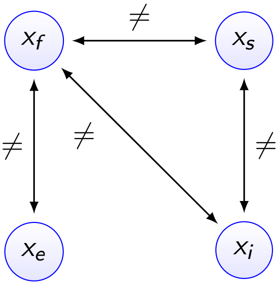
Choice of representation

- 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

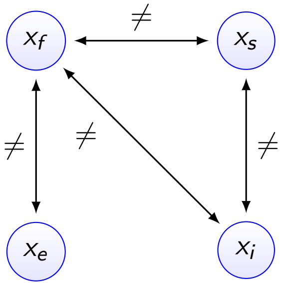
Choice of representation

- 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

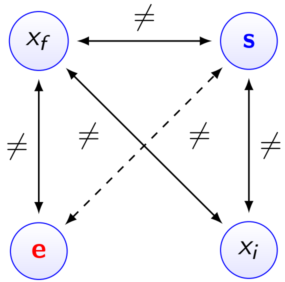


Choice of representation



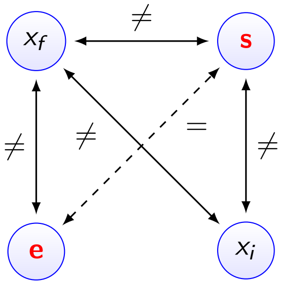
- Zykov recurrence [Zykov 49]: take a non-edge e, s. In the optimal coloring:

Choice of representation



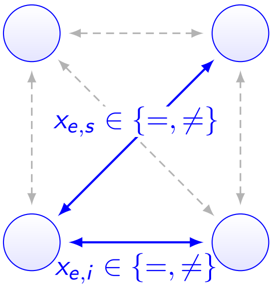
- Zykov recurrence [Zykov 49]: take a non-edge e, s. In the optimal coloring:
  - either e and s take a different color, so adding the edge would not hurt

Choice of representation



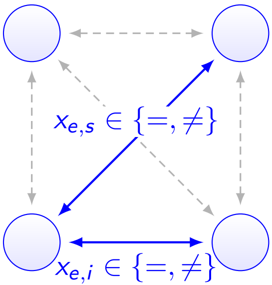
- Zykov recurrence [Zykov 49]: take a non-edge e, s. In the optimal coloring:
  - either e and s take a different color, so adding the edge would not hurt
  - or e and s take the same color, so merging them (adding an equality constraint) would not hurt

Choice of representation



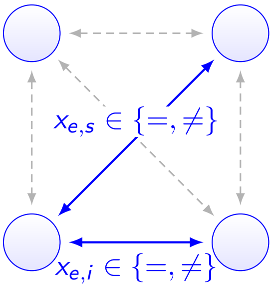
- Zykov recurrence [Zykov 49]: take a non-edge e, s. In the optimal coloring:
  - either e and s take a different color, so adding the edge would not hurt
  - or e and s take the same color, so merging them (adding an equality constraint) would not hurt
- Instead of assigning colors to nodes, we can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

Combining constraints (logically)

Combining constraints (logically)

- Most logic operators
  - can be used as a relation ($x \neq y$)

Combining constraints (logically)

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

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)

$$(x \neq y) \implies y \leq 12 \quad \text{encoded as} \quad \begin{array}{l} (x \neq y) \iff z \\ z \implies (y \leq 12) \end{array}$$

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

Combining constraints (functionally)

Combining constraints (functionally)

- There are also function operators that must be combined similarly
  - For instance $(|x - y| * z) \leq (z + 12)$

$$(|x - y| * z) \leq (z + 12) \quad \text{encoded as} \quad \begin{aligned} &(x - y) = a_1 \\ &|a_1| = a_2 \\ &a_2 * z = a_3 \\ &z + 12 = a_4 \\ &a_3 \leq a_4 \end{aligned}$$

Constraints - Root of the expression tree

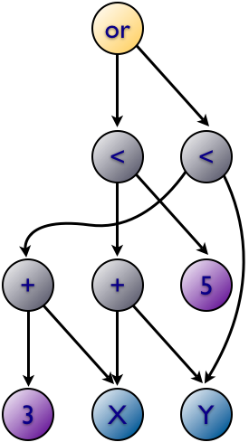
```
C1 = (X+Y < 5) | (X+3 < Y)
C2 = AllDiff([x,y,z])
C3 = Sum([a,b,c,d]) >= e
```

Predicates & functions - Internal nodes

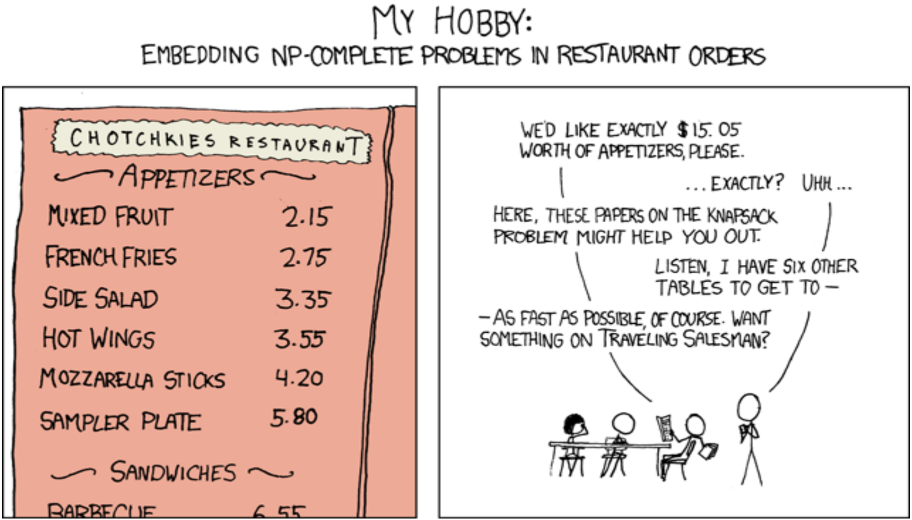
```
P = X+Y          # arythmetic value
Q = X+3 <= Y     # truth (logic) value
```

Variables - Leaves of the expression tree

```
X = Variable(0,10)
X = Variable([1,3,5,7])
```



XKCD Knapsack



XKCD Knapsack

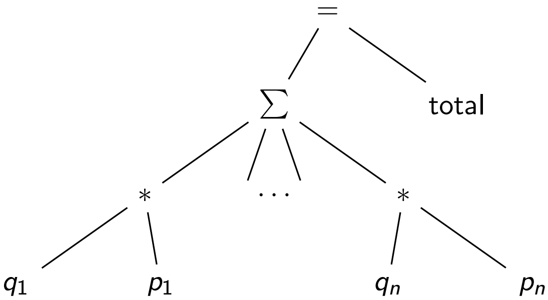
```
from Numberjack import *

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([quantities[i] * price[i] for i in range(num_appetizers)]) == total
```

XKCD Knapsack

```
Sum([quantities[i] * price[i] for i in range(num_appetizers)]) == 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

Global constraints

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

AllDifferent

$$AllDifferent(x_1, \ldots, x_n) \iff \forall 1 \leq i < j \leq n \; x_i \neq x_j$$

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

AllDifferent

$$AllDifferent(x_1, \dots, x_n) \iff \forall 1 \leq i < j \leq n \; x_i \neq x_j$$

$\bar{x} = 3, 5, 1, 2, 7$ 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

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

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

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

Element

$$\mathit{Element}(x_0, \ldots, x_{n-1}, y, z) \iff x_y = z$$

Global constraints

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

AllDifferent

$$AllDifferent(x_1, \ldots, x_n) \iff \forall 1 \leq i < j \leq n \; x_i \neq x_j$$

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

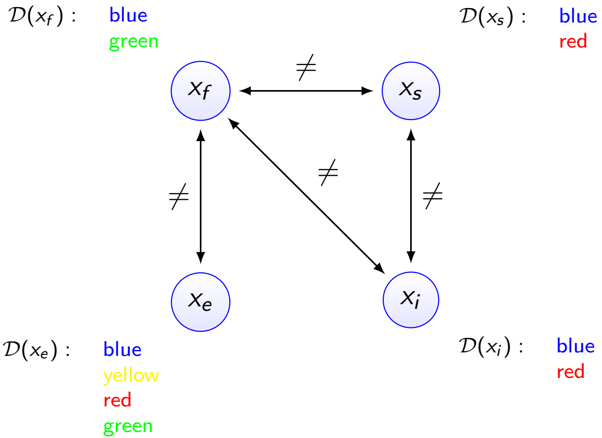
$\bar{x} = 3, 5, 1, 2, 5$ does not satisfy AllDifferent

Element

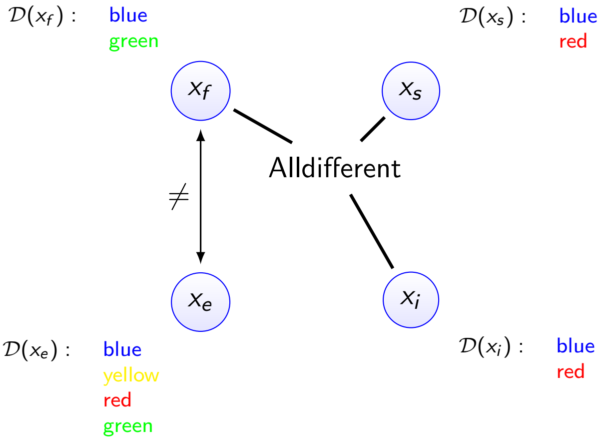
$$Element(x_0, \ldots, x_{n-1}, y, z) \iff x_y = z$$

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

$\bar{x} = 3, 5, 1, 2, 5, y = 2, z = 5$ does not satisfy Element


$\mathcal{D}(x_f)$: blue green
$\mathcal{D}(x_s)$: blue red
x_f
x_s
x_e
x_i
$\neq$
$\neq$
$\neq$
$\neq$
$\mathcal{D}(x_e)$: blue yellow red green
$\mathcal{D}(x_i)$: blue red

Map Coloring



Constraint solver

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 $x \neq v$

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

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 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

Example: binary constraint

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 $v \in \mathcal{D}(x_f)$ and a value $w \in \mathcal{D}(x_e)$ with $v \neq w$

Example: binary constraint

- What inference can the inequality $x_f \neq x_e$ make?
- A support: a value $v \in \mathcal{D}(x_f)$ 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 <u>any</u> value
- $x_f \in \{\mathbf{b}, \mathbf{r}\}, x_e \in \{\mathbf{b}, \mathbf{r}, \mathbf{g}\}$: 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 <u>any</u> value
- $x_f \in \{\mathbf{b}, \mathbf{r}\}, x_e \in \{\mathbf{b}, \mathbf{r}, \mathbf{g}\}$: there is no correct domain reduction
- If $\mathcal{D}(x_f) = \{v\}$ then x_e cannot take the value v
- $x_f \in \{\mathbf{b}\}, x_e \in \{\mathbf{b}, \mathbf{r}, \mathbf{g}\} \implies x_f \in \{\mathbf{b}\}, x_e \in \{\mathbf{r}, \mathbf{g}\}$

Search Tree

$$x_f \in \{\mathbf{b}, \mathbf{g}\} \; - \; x_s \in \{\mathbf{b}, \mathbf{r}\}$$

$$x_e \in \{\mathbf{b}, \mathbf{r}, \mathbf{g}, \mathbf{y}\} \qquad x_i \in \{\mathbf{b}, \mathbf{r}\}$$

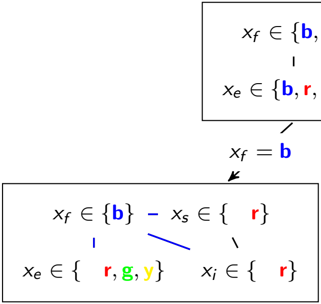
Search Tree

$$x_f \in \{\mathbf{b}, \mathbf{g}\} \; - \; x_s \in \{\mathbf{b}, \mathbf{r}\}$$
$$x_e \in \{\mathbf{b}, \mathbf{r}, \mathbf{g}, \mathbf{y}\} \qquad x_i \in \{\mathbf{b}, \mathbf{r}\}$$

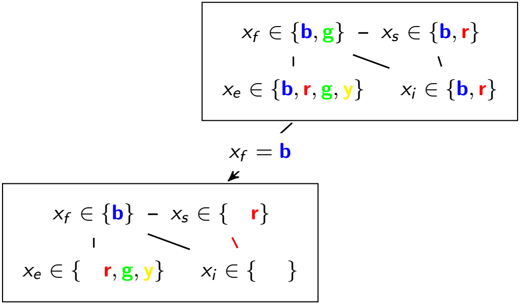
$$x_f = \mathbf{b}$$

$$x_f \in \{\mathbf{b}\} \; - \; x_s \in \{\mathbf{b}, \mathbf{r}\}$$
$$x_e \in \{\mathbf{b}, \mathbf{r}, \mathbf{g}, \mathbf{y}\} \qquad x_i \in \{\mathbf{b}, \mathbf{r}\}$$

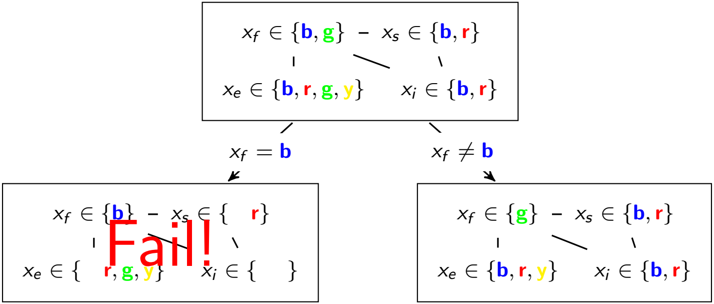
Search Tree


$x_f \in \{b, g\}$ – $x_s \in \{b, r\}$
$x_e \in \{b, r, g, y\}$
$x_i \in \{b, r\}$
$x_f = b$
$x_f \in \{b\}$ – $x_s \in \{ \; r\}$
$x_e \in \{ \; r, g, y\}$
$x_i \in \{ \; r\}$

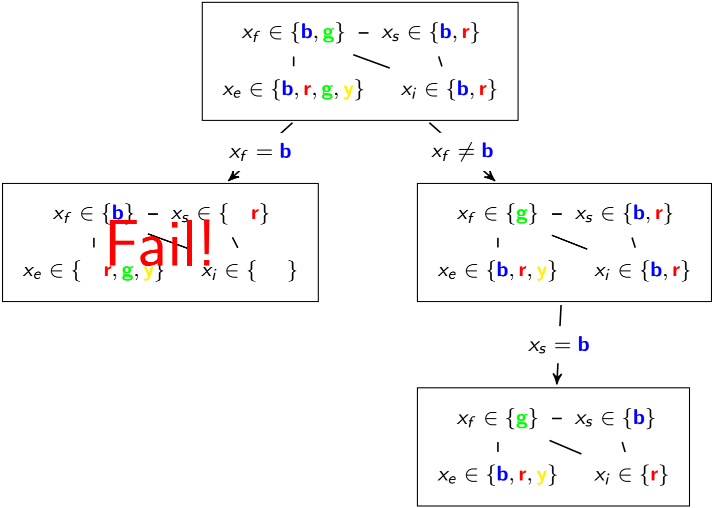
Search Tree


$x_f \in \{b, g\}$ – $x_s \in \{b, r\}$
$x_e \in \{b, r, g, y\}$
$x_i \in \{b, r\}$
$x_f = b$
$x_f \in \{b\}$ – $x_s \in \{ \quad r\}$
$x_e \in \{ \quad r, g, y\}$
$x_i \in \{ \quad \}$

Search Tree


$x_f \in \{b, g\}$ – $x_s \in \{b, r\}$
$x_e \in \{b, r, g, y\}$
$x_i \in \{b, r\}$
$x_f = b$
$x_f \neq b$
$x_f \in \{b\}$ – $x_s \in \{\ \ r\}$
$x_e \in \{\ \ r, g, y\}$
$x_i \in \{\ \ \}$
Fail!
$x_f \in \{g\}$ – $x_s \in \{b, r\}$
$x_e \in \{b, r, y\}$
$x_i \in \{b, r\}$

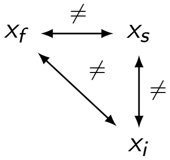
Search Tree



$x_f \in \{b, g\}$ – $x_s \in \{b, r\}$
$x_e \in \{b, r, g, y\}$
$x_i \in \{b, r\}$
$x_f = b$
$x_f \neq b$
$x_f \in \{b\}$ – $x_s \in \{ \quad r\}$
$x_e \in \{ \quad r, g, y\}$
$x_i \in \{ \quad \}$
Fail!
$x_f \in \{g\}$ – $x_s \in \{b, r\}$
$x_e \in \{b, r, y\}$
$x_i \in \{b, r\}$
$x_s = b$
$x_f \in \{g\}$ – $x_s \in \{b\}$
$x_e \in \{b, r, y\}$
$x_i \in \{r\}$

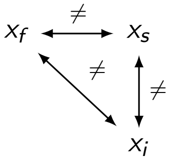
Example: global constraint

Example: global constraint



$$
\begin{aligned}
x_f &\in \{\mathbf{b}, \mathbf{g}\} \\
x_s &\in \{\mathbf{b}, \mathbf{r}\} \\
x_i &\in \{\mathbf{b}, \mathbf{r}\}
\end{aligned}
$$

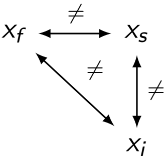
Example: global constraint



$$x_f \in \{\mathbf{b}, \mathbf{g}\}$$
$$x_s \in \{\mathbf{b}, \mathbf{r}\}$$
$$x_i \in \{\mathbf{b}, \mathbf{r}\}$$

- Every inequality is consistent

Example: global constraint

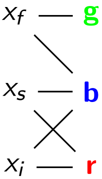


$$
\begin{aligned}
x_f &\in \{\mathbf{b}, \mathbf{g}\} \\
x_s &\in \{\mathbf{b}, \mathbf{r}\} \\
x_i &\in \{\mathbf{b}, \mathbf{r}\}
\end{aligned}
$$

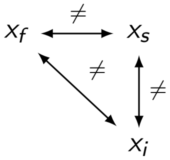
- Every inequality is consistent
- AllDifferent is not consistent!

Propagation of AllDifferent($\bar{x}$)

- A support is a perfect matching in the graph



Example: global constraint

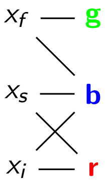


$$x_f \in \{\mathbf{b}, \mathbf{g}\}$$
$$x_s \in \{\mathbf{b}, \mathbf{r}\}$$
$$x_i \in \{\mathbf{b}, \mathbf{r}\}$$

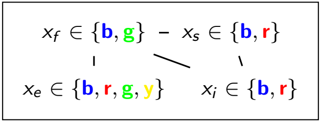
- Every inequality is consistent
- AllDifferent is not consistent!

Propagation of AllDifferent($\bar{x}$)

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



Search Tree (AllDifferent)

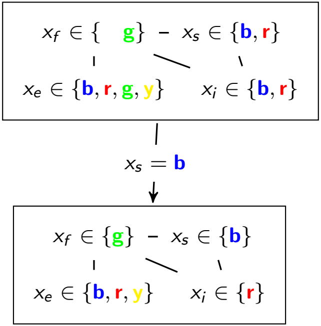


Search Tree (AllDifferent)

$x_f \in \{\ \ \mathbf{g}\}$ – $x_s \in \{\mathbf{b}, \mathbf{r}\}$

$x_e \in \{\mathbf{b}, \mathbf{r}, \mathbf{g}, \mathbf{y}\}$ $x_i \in \{\mathbf{b}, \mathbf{r}\}$

Search Tree (AllDifferent)



Propagation algorithm

- Every constraint has a propagation algorithm

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 $\mathcal{D}(x)$

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 $\mathcal{D}(x)$
- 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 $\mathcal{D}(x)$
- The bigger (more global) the stronger! (and the slower...)

Outline

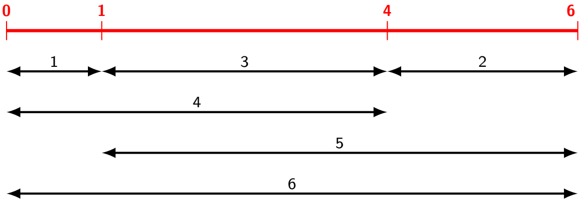
The art of modeling

Techniques to strenghthen 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



A First Model (Numberjack)

```
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)

```
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);
```

Branch & Bound

- An objective variable

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

Branch & Bound

- An objective variable

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

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

Branch & Bound

- An objective variable

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

- The upper bound is updated when a new solution is found
- The lower bound is maintained via constraint propagation

```
model.max(objective, marks).post();
```

Branch & Bound

- An objective variable

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

- The upper bound is updated when a new solution is found
- The lower bound is maintained via constraint propagation

```
model.max(objective, marks).post();
```

- Different models may entail different lower bounds for the same objective function

Global Constraints (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 = [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]
```

Global Constraints (Choco)

```
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);
```

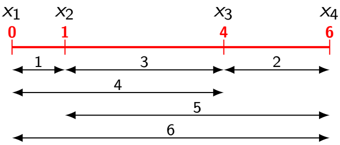
Symmetry breaking

Symmetry breaking

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

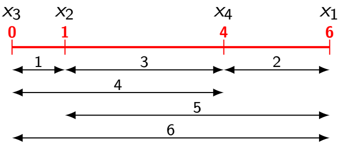
Symmetry breaking

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



- Variable symmetries: `marks`, `distance`

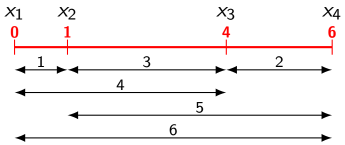
- Solution symmetries $\Rightarrow$ symmetric (suboptimal) branches in the search tree



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

Symmetry breaking

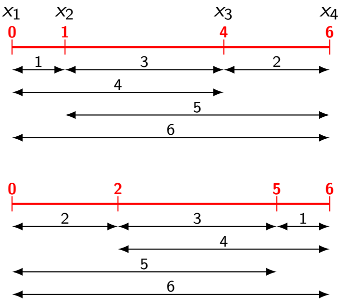
- Solution symmetries $\Rightarrow$ symmetric (suboptimal) branches in the search tree



- Variable symmetries: `marks`, `distance`
- We can swap the marks or the distances of a solution (but not both)
- Force an arbitrary ordering
  - ★ `marks[1]` $<$ `marks[2]` $< \ldots <$ `marks[`m`]`

Symmetry breaking

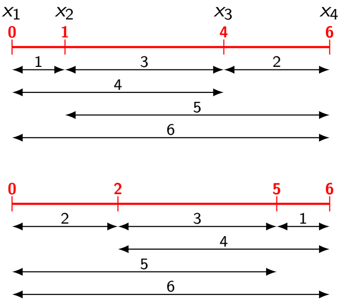
- Solution symmetries ⇒ symmetric (suboptimal) branches in the search tree



- 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]` $< \ldots <$ `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
  - ★ `marks[1]` $<$ `marks[2]` $< \ldots <$ `marks[`m`]`
- Distances are still symmetric by reflection
  - ★ `distance[0,1]` $<$ `distance[`$m-2, m-1$`]`

Symmetry breaking (Numberjack)

```
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]
```

Symmetry breaking (Choco)

```
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 constraint

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

Implied Constraints

Implied constraint

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

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

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

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 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)

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:

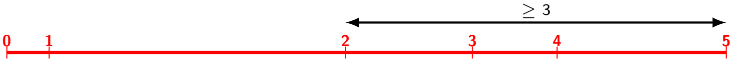
- $x \neq y, y \neq z, x \neq z \implies \text{AllDifferent}(x, y, z)$
- $x \neq y, x \leq y \implies x < y$

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

- $x \neq y$ 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 $x \in \{1, \dots, 9\}, y \in \{2, \dots, 10\}$

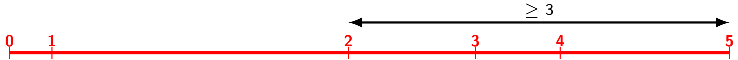
Implied Constraints: Golomb Ruler

Implied Constraints: Golomb Ruler



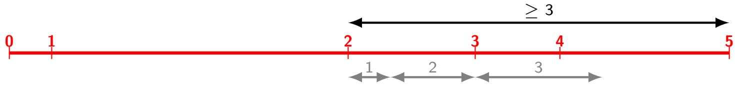
- `distance[i,j]` $\geq$ sum of $j - i$ distances

Implied Constraints: Golomb Ruler



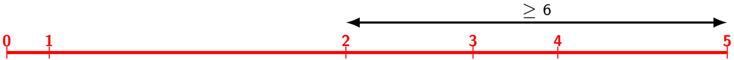
- `distance[i,j]` $\geq$ sum of $j - i$ distances
- The distances are all different

Implied Constraints: Golomb Ruler



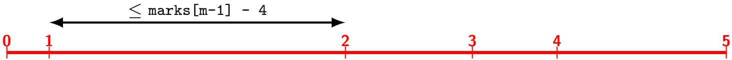
- `distance[i,j]` $\geq$ sum of $j - i$ distances
- The distances are all different

Implied Constraints: Golomb Ruler



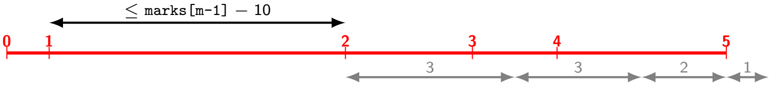
- `distance[i,j]` $\geq$ sum of $j - i$ distances
- The distances are all different `distance[i,j]` $\geq (j - i) * (j - i + 1)/2$

Implied Constraints: Golomb Ruler



- `distance[i,j]` $\geq$ sum of $j - i$ distances
- The distances are all different `distance[i,j]` $\geq (j - i) * (j - i + 1)/2$
- Same reasoning from the end (`marks[`$m - 1$`]`)
  - `distance[i,j]` $\leq$ `marks[m]` $-$ sum of $m - 1 - j + i$ distances

Implied Constraints: Golomb Ruler



- distance[i,j] $\geq$ sum of $j - i$ distances
- The distances are all different distance[i,j] $\geq (j - i) * (j - i + 1)/2$
- Same reasoning from the end (marks[$m - 1$])
  - distance[i,j] $\leq$ marks[m]$-$ sum of $m - 1 - j + i$ distances
  - distance[i,j] $\leq$ marks[m] $- (m - 1 - j + i) * (m - j + i)/2$

Implied Constraints: Golomb Ruler

- Implied constraints
  - `distance[i,j]` $\geq (j - i) * (j - i + 1)/2$
  - `distance[i,j]` $\leq$ `marks[m]` $- (m - 1 - j + i) * (m - j + i)/2$

Implied Constraints: Golomb Ruler

- Implied constraints
  - `distance[i,j]` $\geq (j - i) * (j - i + 1)/2$
  - `distance[i,j]` $\leq$ `marks[m]` $- (m - 1 - j + i) * (m - j + i)/2$

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

Implied Constraints: Golomb Ruler

- Implied constraints
  - `distance[i,j]` $\geq (j-i)*(j-i+1)/2$
  - `distance[i,j]` $\leq$ `marks[m]` $-(m-1-j+i)*(m-j+i)/2$

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

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

Implied Constraints: Golomb Ruler

- Implied constraints
  - `distance[i,j]` $\geq (j-i)*(j-i+1)/2$
  - `distance[i,j]` $\leq$ `marks[m]` $-(m-1-j+i)*(m-j+i)/2$

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

- We need to combine the reasoning of two constraints (`AllDifferent(distance)` and `distance[i,j]` $= \sum_{k=i}^{j-1}$ `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
  - distance[i,j] $\geq (j - i) * (j - i + 1)/2$
  - distance[i,j] $\leq$ marks[m] $- (m - 1 - j + i) * (m - j + i)/2$

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

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

Implied Constraints (Numberjack)

```
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 (Choco)

```
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]);
```

Conclusions

Good modeling practices

Good modeling practices

- What are the variables, what are the values?

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**

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**

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!