Modeling Problems in Constraint Programming

Jean-Charles REGIN
ILOG, Sophia Antipolis
regin@ilog.fr

Plan

- Principles of Constraint Programming
- A rostering problem
- Modeling in CP: Principles
- A difficult problem
- A Network Design problem
- Modeling Over-constrained problems
- Discussion
- Conclusion

3 problems

- 3 problems will be detailed:
  - A rostering problem (G. Pesant). This is a real world problem. The problem is easy to solve in CP because all the needed constraints are available. The presentation will be constructive, that is the problem is modeled and described at the same time
  - A part of a real world problem. Mainly a didactic problem which is difficult to solve in CP.
  - A real world problem will be presented: a network design. First the whole problem will be described and then a CP solution will be proposed

Plan

- **Principles of Constraint Programming**
- A rostering problem
- Modeling in CP: Principles
- A difficult problem
- A Network Design problem
- Modeling Over-constrained problems
- Discussion
- Conclusion

Constraint programming

- Identify sub-problems that are easy (called constraints)

Constraint programming

- Identify sub-problems that are easy (called constraints)
- 1) Use specific algorithm for solving these sub-problems and for performing domain-reduction
- 2) Instantiate a variable. Go to 1) and backtrack if necessary

Constraint programming

- Identify sub-problems that are easy (called constraints)
- 1) Use specific algorithm for solving these sub-problems and for performing domain-reduction
- 2) Instantiate a variable. Go to 1) and backtrack if necessary
- **Local point of view on sub-problems. “Global” point of view by propagation of domain reductions**

Constraint Programming

- 3 notions:
  - constraint network: variables, domains, constraints
  + filtering (domain reduction)
  - propagation
  - search procedure (assignments + backtrack)

Problem = conjunction of sub-problems

- In CP a problem can be viewed as a conjunction of sub-problems that we are able to solve
- A sub-problem can be trivial: $x < y$ or complex: search for a feasible flow
- A sub-problem = a constraint

Constraints

- Predefined constraints: arithmetic (x < y, x = y +z, |x-y| > k, alldiff, cardinality, sequence …
- Constraints given in extension by the list of allowed (or forbidden) combinations of values
- user-defined constraints: any algorithm can be encapsulated
- Logical combination of constraints using OR, AND, NOT, XOR operators. Sometimes called meta-constraints

Filtering

- We are able to solve a sub-problem: a method is available
- CP uses this method to remove values from domain that do not belong to a solution of this sub-problem: **filtering**
- E.g: x < y and D(x)=[10,20], D(y)=[5,15]
  => D(x)=[10,14], D(y)=[11,15]

Filtering

- A filtering algorithm is associated with each constraint (sub-problem).
- Can be simple (x < y) or complex (alldiff)

Arc consistency

- All the values which do not belong to any solution of the constraint are deleted.
- Example: Alldiff({x,y,z}) with
  D(x)=D(y)={0,1}, D(z)={0,1,2}
  the two variables x and y take the values 0 and 1, thus z cannot take these values.
  FA by AC => 0 and 1 are removed from D(z)

Propagation

- Domain Reduction due to one constraint can lead to new domain reduction of other variables
- When a domain is modified all the constraints involving this variable are studied and so on ...

Why Propagation?

- A problem = conjunction of easy sub-problems.
- Sub-problems: local point of view. Propagation tries to obtain a global point of view from independent local point of view
- The conjunction is stronger that the union of independent resolutions

Why Propagation?

- A problem = conjunction of easy sub-problems.
- Sub-problems: local point of view. Propagation tries to obtain a global point of view from independent local point of view
- The conjunction is stronger that the union of independent resolution
- **To help the propagation to have a global point of view: use global constraints !**
- **Global constraint = conjunction of constraints**

Search

- Backtrack algorithm with strategies:
  try to successively assign variables with values. If a dead-end occurs then backtrack and try another value for the variable
- Strategy: define which variable and which value will be chosen.
- After each domain reduction (i.e assignment) filtering and propagation are triggered

Plan

- Principles of Constraint Programming
- **A rostering problem**
- Modeling in CP: Principles
- A difficult problem
- A Network Design problem
- Modeling Over-constrained problems
- Discussion
- Conclusion

Rostering (G. Pesant)

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
D							
E							
N							

M. Green   M. Red
Mrs. Blue   M. Yellow

Rostering

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
D							
E							
N							

M. Green

M. Red

Mrs. Blue

M. Yellow

Each works at most one shift per day

Rostering

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
D							
E							
N							

M. Green
M. Red
Mrs. Blue
M. Yellow

$x_{ij} \in \{g, b, r, y\}$

$x_{iD} \neq x_{iE},\ x_{iD} \neq x_{iN},\ x_{iE} \neq x_{iN}$ $\quad \text{Mon} \leq i \leq \text{Sun}$

Rostering

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
D							
E							
N							

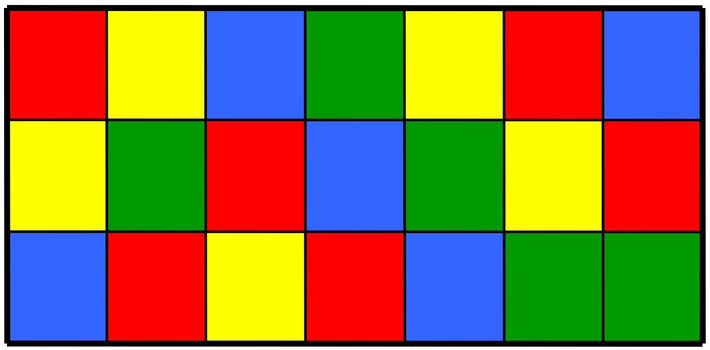
M. Green
Mrs. Blue
M. Red
M. Yellow

```
enum Days = {mon,tue,wed,thu,fri,sat,sun}
enum Shifts = {D,E,N}
enum Workers = {green,white,red,yellow}
var Workers onDuty[Days,Shifts]
forall( i in Days )
        forall( j,k in Shifts: j < k )
                onDuty[i,j] ≠ onDuty[i,k]
```

Rostering

Mon Tue Wed Thu Fri Sat Sun

D
E
N



M. Green M. Red
Mrs. Blue M. Yellow

```
enum Days = {mon,tue,wed,thu,fri,sat,sun}
enum Shifts = {D,E,N}
enum Workers = {green,white,red,yellow}
var Workers onDuty[Days,Shifts]
forall( i in Days )
        forall( j,k in Shifts: j < k )
                onDuty[i,j] ≠ onDuty[i,k]
```

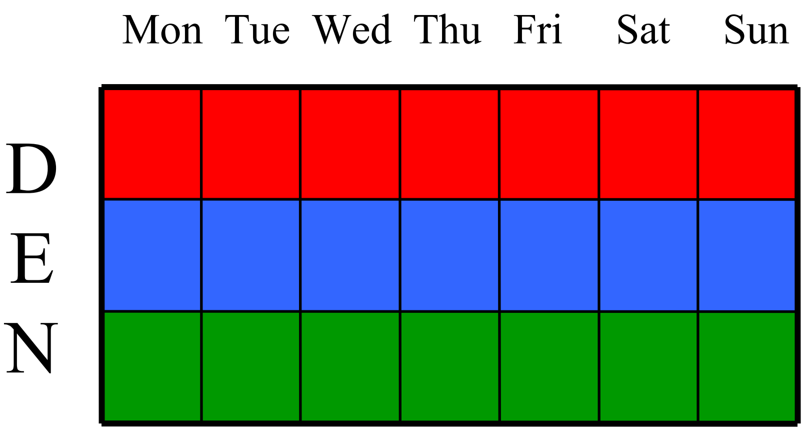
Mutual exclusion

- A set of variables must take on distinct values.
- forall(i in Days)
  forall(j,k in Shifts: j < k)
  onDuty[i,j] ≠ onDuty[i,k]

Mutual exclusion

- A set of variables must take on distinct values.
- forall(i in Days)
  forall(j,k in Shifts: j < k)
  onDuty[i,j] ≠ onDuty[i,k]
- Can be replaced by
  forall(i in Days)
  alldifferent(onDuty[i])

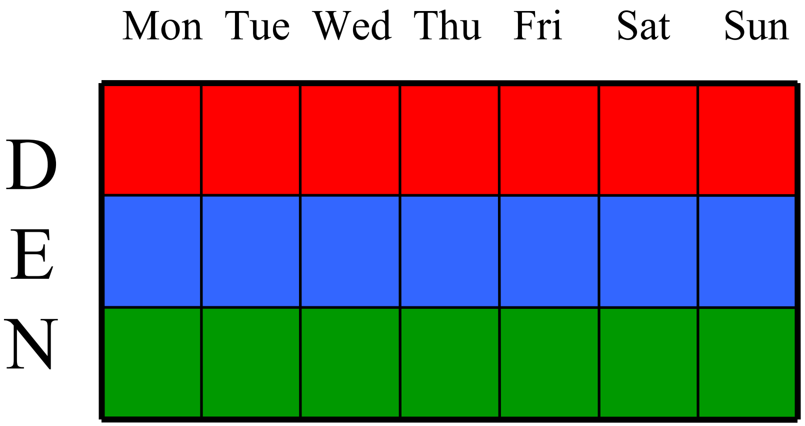
Cardinality



M. Green M. Red
Mrs. Blue M. Yellow

This is not a good solution

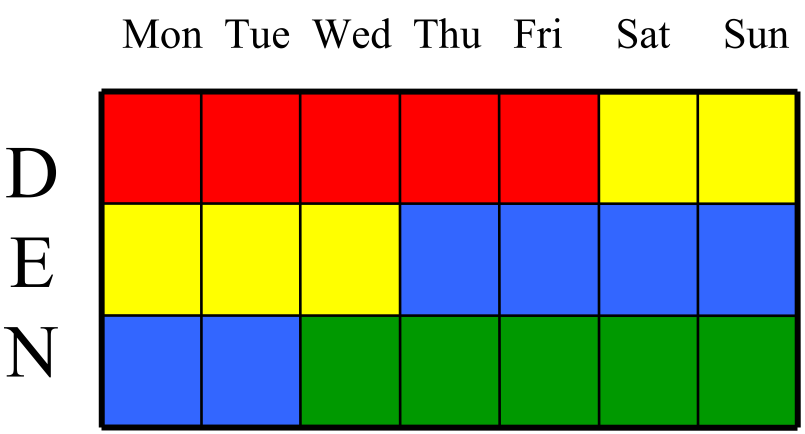
Cardinality



M. Green
M. Red
Mrs. Blue
M. Yellow

```
var 0..7 nbShifts[Workers]
distribute(nbShifts,Workers,onDuty)
forall( k in Workers )
        nbShifts[k] ≥ 5
```

Cardinality



M. Green
M. Red
Mrs. Blue
M. Yellow

var 0..7 nbShifts[Workers]
distribute(nbShifts,Workers,onDuty)
forall(k in Workers)
nbShifts[k] ≥ 5

Dual Model

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
M. Green							
Mrs. Blue							
M. Red							
M. Yellow							

enum Jobs = {D,E,N,-}
var Jobs job[Days,Workers]

implicitly, each works at most one shift per day.
But every job has to be performed and by only one worker

Dual Model

M. Green
Mrs. Blue
M. Red
M. Yellow

Mon	Tue	Wed	Thu	Fri	Sat	Sun
D						
E						
N						
-						

implicitly, each works at most one shift per day.
But every job is performed by only one worker
forall(i in Days)
    distribute([1,1,1,1],Jobs,job[i])

Dual Model: weights on jobs

M. Green
Mrs. Blue
M. Red
M. Yellow

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
	D						
	E						
	N						
	-						

Jobs have weights: D=1.; E=0.8; N=0.5; -=0

float load[Jobs] = {1.0, 0.8, 0.5, 0.0}

$job[i,k] \in \{D,N\} \leftrightarrow load[job[i,k]] \in \{1.0, 0.5\}$

Dual Model: weights on jobs

M. Green
Mrs. Blue
M. Red
M. Yellow

Mon	Tue	Wed	Thu	Fri	Sat	Sun
D						
E						
N						
-						

Jobs have weights: D=1.; E=0.8; N=0.5; -=0

```
float load[Jobs] = {1.0, 0.8, 0.5, 0.0}
forall( k in Workers )
        sum( i in Days ) load[job[i,k]] ≥ 3.0
```

Dual Model: weights on jobs

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
M. Green	D	-	D	-	D	-	D
Mrs. Blue	-	N	N	N	N	N	N
M. Red	N	D	-	D	E	D	-
M. Yellow	E	E	E	E	-	E	E

Jobs have weights: D=1.; E=0.8; N=0.5; -=0

```
float load[Jobs] = {1.0, 0.8, 0.5, 0.0}
forall( k in Workers )
        sum( i in Days ) load[job[i,k]] ≥ 3.0
```

Length of Runs

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
M. Green	D	-	D	-	D	-	D
Mrs. Blue	-	N	N	N	N	N	N
M. Red	N	D	-	D	E	D	-
M. Yellow	E	E	E	E	-	E	E

This is not nice, isn’t it?

Length of Runs

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
M. Green	D	-	D	-	D	-	D
Mrs. Blue	-	N	N	N	N	N	N
M. Red	N	D	-	D	E	D	-
M. Yellow	E	E	E	E	-	E	E

New constraint: length of runs defined by a range, i.e. between a **min** and a **max** value

Length of Runs

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
M. Green	D	-	D	-	D	-	D
Mrs. Blue	-	N	N	N	N	N	N
M. Red	N	D	-	D	E	D	-
M. Yellow	E	E	E	E	-	E	E

int min[Jobs] = {2,1,1,1}
int max[Jobs] = {4,4,4,7}
forall(k in Workers)
    stretch(min,max,job[□,k])

Length of Runs

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
M. Green	D	D	-	N	E	D	D
Mrs. Blue	N	N	N	-	N	N	N
M. Red	-	-	D	D	D	-	-
M. Yellow	E	E	E	E	-	E	E

```
int min[Jobs] = {2,1,1,1}
int max[Jobs] = {4,4,4,7}
forall( k in Workers )
        stretch(min,max,job[□,k])
```

Pattern Constraint

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
M. Green	D	D	-	N	E	D	D
Mrs. Blue	N	N	N	-	N	N	N
M. Red	-	-	D	D	D	-	-
M. Yellow	E	E	E	E	-	E	E

No change of shift type without a rest period

Forward rotation (D... E... N... D...)

Pattern Constraint

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
M. Green	D	D	-	N	E	D	D
Mrs. Blue	N	N	N	-	N	N	N
M. Red	-	-	D	D	D	-	-
M. Yellow	E	E	E	E	-	E	E

No change of shift type without a rest period

Forward rotation (D... E... N... D...)

forall(k in Workers)

regular(A,job[□,k])

Pattern Constraint

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
M. Green	D	D	-	E	E	E	E
Mrs. Blue	E	E	E	-	N	N	N
M. Red	N	N	N	N	-	D	D
M. Yellow	-	-	D	D	D	-	-

No change of shift type without a rest period

Forward rotation (D... E... N... D...)

forall(k in Workers)

regular(A,job[□,k])

Real life rostering

∘	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S
23796	-	-	-	-	D	D	N	N	-	-	D	-	-	-	-	D	-	D	D	-	D	D	-	-	-	-	-	-
603042	D	D	D	D	E	-	-	-	D	D	D	D	-	D	D	D	D	D	E	-	-	-	D	D	D	D	-	D
12310	D	D	-	-	-	-	-	-	-	-	-	-	-	D	D	D	-	-	-	-	-	-	-	-	-	-	-	D
511811	D	D	D	-	D	D	-	-	D	D	-	-	-	D	D	D	D	-	D	D	-	-	D	D	-	-	-	D
60324	-	-	D	D	D	-	D	D	-	D	D	D	-	-	-	-	-	D	D	-	D	D	D	-	D	D	D	-
603095	E	-	-	E	E	E	-	-	-	-	-	-	E	E	E	-	-	E	E	E	-	-	-	-	E	-	-	E
603230	-	D	D	D	D	-	D	D	D	D	-	D	D	-	-	D	D	D	D	-	D	D	D	-	D	D	D	-
510723	D	D	D	-	-	D	-	-	D	D	D	-	-	D	D	D	D	-	-	D	-	-	D	D	D	-	-	D
511104	-	R	R	R	R	R	-	-	R	R	R	R	R	-	-	-	-	E	E	-	E	E	-	-	E	E	E	-
34108	-	D	D	D	D	-	D	D	D	D	-	-	-	-	-	R	R	R	R	R	D	D	-	-	D	-	-	-
11866	-	D	-	D	D	D	E	E	-	-	D	-	-	-	-	D	-	D	D	D	E	E	-	D	-	-	-	-
35022	-	R	R	R	R	R	D	D	-	-	-	-	-	-	-	-	-	-	D	-	D	D	D	-	D	D	D	-
512287	E	E	E	-	D	D	E	E	-	-	-	-	-	E	E	E	E	-	D	-	E	E	-	-	E	-	-	E
56507	D	D	-	D	D	D	-	-	D	-	-	-	-	D	D	D	-	D	D	D	-	-	D	-	-	-	-	D
512281	-	E	-	D	D	-	D	D	E	-	-	-	-	-	-	E	-	D	D	-	D	D	E	-	-	-	-	-
511066	-	D	D	-	-	-	D	D	-	-	-	-	D	-	-	-	-	-	-	-	D	D	-	-	D	D	D	-
600955	D	D	-	D	D	-	-	-	-	-	-	-	-	D	D	D	-	D	D	-	-	-	-	-	-	-	-	D
602576	D	D	-	D	D	D	-	-	-	-	-	-	-	D	D	D	-	D	D	D	-	-	-	-	-	-	-	D
600315	-	-	T	T	-	-	T	T	-	T	-	T	T	-	-	-	T	-	-	T	T	T	-	-	T	T	T	-
511865	-	-	-	-	-	-	T	T	-	T	T	T	T	-	-	-	-	-	-	-	-	-	R	R	R	R	R	T

Real life rostering

◦	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S
603287	-	-	-	-	-	-	E	E	-	-	E	E	E	-	-	-	-	-	-	-	E	E	-	-	E	E	E	-
603138	-	-	E	E	E	E	E	E	-	E	E	E	E	-	-	-	E	E	E	E	E	E	-	E	E	E	E	-
510595	-	D	D	D	-	-	D	D	R	R	R	R	R	-	-	R	R	R	R	R	D	D	-	-	D	D	D	-
53033	-	R	R	R	R	R	D	D	D	D	-	-	-	D	D	D	D	N	N	N	-	-	D	D	D	-	-	D
602712	D	D	D	N	N	N	-	-	D	D	D	-	-	D	D	D	D	N	N	N	-	-	D	D	D	-	-	D
601933	D	D	-	D	D	D	-	-	D	D	D	D	-	D	D	D	D	D	N	N	-	-	D	D	D	-	-	D
603134	D	D	D	N	N	N	-	-	D	D	D	-	-	D	D	D	D	N	N	N	-	-	D	D	D	-	-	D
511938	-	-	-	-	D	D	D	D	-	-	D	D	D	-	-	-	-	D	D	-	D	D	D	-	D	D	-	-
601659	N	N	N	-	N	N	-	-	N	N	-	-	-	N	N	N	-	N	N	N	-	-	N	N	-	-	-	N
62273	N	N	-	-	N	N	-	-	N	-	N	-	-	N	N	N	-	N	N	N	-	-	N	-	-	-	-	N
601630	-	D	D	D	D	-	D	D	-	-	-	D	D	-	-	D	D	D	-	-	D	D	-	-	D	D	D	-
601983	N	N	-	N	N	-	-	-	-	-	-	-	-	N	N	N	N	N	-	-	-	-	-	-	-	-	-	N
511545	-	N	N	-	-	-	D	D	-	-	-	N	N	-	-	-	-	-	-	D	N	N	-	-	N	N	N	-
603157	D	-	D	D	D	E	-	-	D	D	D	-	D	D	D	-	D	D	D	-	-	-	D	D	D	E	E	E
603361	-	D	D	D	E	-	D	D	D	E	-	D	D	-	-	D	D	D	E	-	D	D	D	-	D	D	D	-
602759	-	-	-	-	-	-	D	D	-	D	D	D	-	-	-	-	-	-	-	-	D	D	-	-	D	D	D	-
73999	D	D	D	-	D	-	-	-	-	-	-	-	-	-	-	R	R	R	R	R	-	-	R	R	R	R	R	-
601949	-	D	-	-	-	-	D	D	-	-	-	-	D	-	-	-	-	-	-	-	D	D	-	-	D	D	-	-
511668	D	E	-	-	-	-	-	D	E	-	-	-	-	-	D	E	-	-	-	-	-	D	E	-	-	-	-	-
7096	-	R	R	R	R	R	-	-	R	R	R	R	R	-	D	D	-	D	D	-	D	D	D	-	D	D	D	-
602373	-	D	D	D	D	D	-	-	D	D	D	D	D	-	-	D	D	D	D	D	-	-	D	D	D	D	D	-

Plan

- Principles of Constraint Programming
- A rostering problem
- **Modeling in CP: Principles**
- A difficult problem
- A Network Design problem
- Modeling Over-constrained problems
- Discussion
- Conclusion

Modeling: principles

- What a good model is?
- Symmetries
- Implicit constraints
- Global constraints
- Relevant and redundant constraints
- Back propagation
- Dominance rules

Good Model?

- A good model is a model that leads to an efficient resolution of a given problem

Good Model?

- A good model is a model that leads to an efficient resolution of a given problem
- Deals with several notions:

Symmetries
Implicit constraints
Global constraints
Relevant and redundant constraints
Back propagation
Dominance rules

Symmetries

- Tutorial on this topic at CP’04
- The complexity of a problem can often be reduced by detecting intrinsic symmetries
- When two or more variables have identical characteristics, it is pointless to differentiate them artificially:
  - The initial domains of these variables are identical
  - These variables are subject to the same constraints
  - The variables can be permuted without changing the statement of the problem
- Usually symmetries are removed by introducing an order between variables

Implicit constraints

- See work of B. Smith
- An **implicit** constraint makes explicit a property that satisfies any solution implicitly.
- D(x1)=D(x2)=D(x3)=D(x4)={a,b,c,d}
- Constraints: b,c and d have to be taken at least 1

Implicit constraints

- See work of B. Smith
- An **implicit** constraint makes explicit a property that satisfies any solution implicitly.
- D(x1)=D(x2)=D(x3)=D(x4)={a,b,c,d}
- Constraints: b,c and d have to be taken at least 1
- Filtering algorithm: if b is not assigned and if there is only one variable x that contains b in its domain then x=b

Implicit constraints

- See work of B. Smith
- An **implicit** constraint makes explicit a property that satisfies any solution implicitly.
- D(x1)=D(x2)=D(x3)=D(x4)={a,b,c,d}
- Constraints: b,c and d have to be taken at least 1
- Filtering algorithm: if b is not assigned and if there is only one variable x that contains b in its domain then x=b
- Problem: if x1=a and x2=a then nothing is deduced

Implicit constraints

- See work of B. Smith
- An **implicit** constraint makes explicit a property that satisfies any solution implicitly.
- D(x1)=D(x2)=D(x3)=D(x4)={a,b,c,d}
- Constraints: b,c and d have to be taken at least 1
- Filtering algorithm: if b is not assigned and if there is only one variable x that contains b in its domain then x=b
- Problem: if x1=a and x2=a then nothing is deduced
- Implicit constraints: a can be taken at most 1
  b,c,d can be taken at most 2
- **From the simultaneous presence of some constraints implicit constraints can be deduced**

Global constraints

- A global constraint is a conjunction of constraints. This conjunction often takes into account implicit constraint deduced from the simultaneous presence of the other constraints
- This is the case for the previous example with the global cardinality constraint
- Use the strongest filtering algorithm as you can at the beginning
- It is rare to be able to solve a problem with weak FA and not to be able to solve it with strong FA

Global constraint: Alldiff results

- Color the graph with cliques:
  c0 = {0, 1, 2, 3, 4}
  c1 = {0, 5, 6, 7, 8}
  c2 = {1, 5, 9, 10, 11}
  c3 = {2, 6, 9, 12, 13}
  c4 = {3, 7, 10, 12, 14}
  c5 = {4, 8, 11, 13, 14}
- clique size:27 Global: #fails: 0 cpu time: 1.212 s
  Local: #fails: 1 cpu time: 0.171 s
  clique size:31 Global: #fails: 4 cpu time: 2.263 s
  Local: #fails: 65 cpu time: 0.37 s
  clique size:51 Global: #fails: 501 cpu time: 25.947 s
  Local: #fails: 24512 cpu time: 66.485 s
  clique size:61 Global: #fails: 5 cpu time: 58.223 s
  Local: ?????????????

Relevant Constraints

- At first glance it seems that adding a constraint which removes some symmetries, or which is an implicit or a global constraint improves the current model. This is **FALSE**
- Because:
  - The new filtering algorithm can delete no value, because everything is already deduced by the combination of constraints
  - The new filtering algorithm can remove some values and impacts the variable-value strategy (more backtracks can be needed to reach the first solution)

Relevant constraints

- A constraint is **relevant w.r.t. a model** if the introduction of this constraint:
  - Is needed by the definition of the problem
  - Or if it permits to remove some symmetries, or it is an implied or a global constraint, and the introduction of this constraint improves the search for the solution in term of performance
- A constraint is **redundant w.r.t. a model** if the constraint is not relevant w.r.t. the model.

Back propagation

- Consider an optimization problem with an objective variable obj.
- The back propagation is the consequences of the modifications of the variable obj
- Example:
  $\sum x = obj$.
  Back propagation = modification of the x variable when obj is modified

Back propagation

- Try to improve the back propagation, because when a solution with a cost c is found the constraint obj < c is added and a new solution is sought.
- It is important to use constraints involving cost variable. For instance : gcc with cost

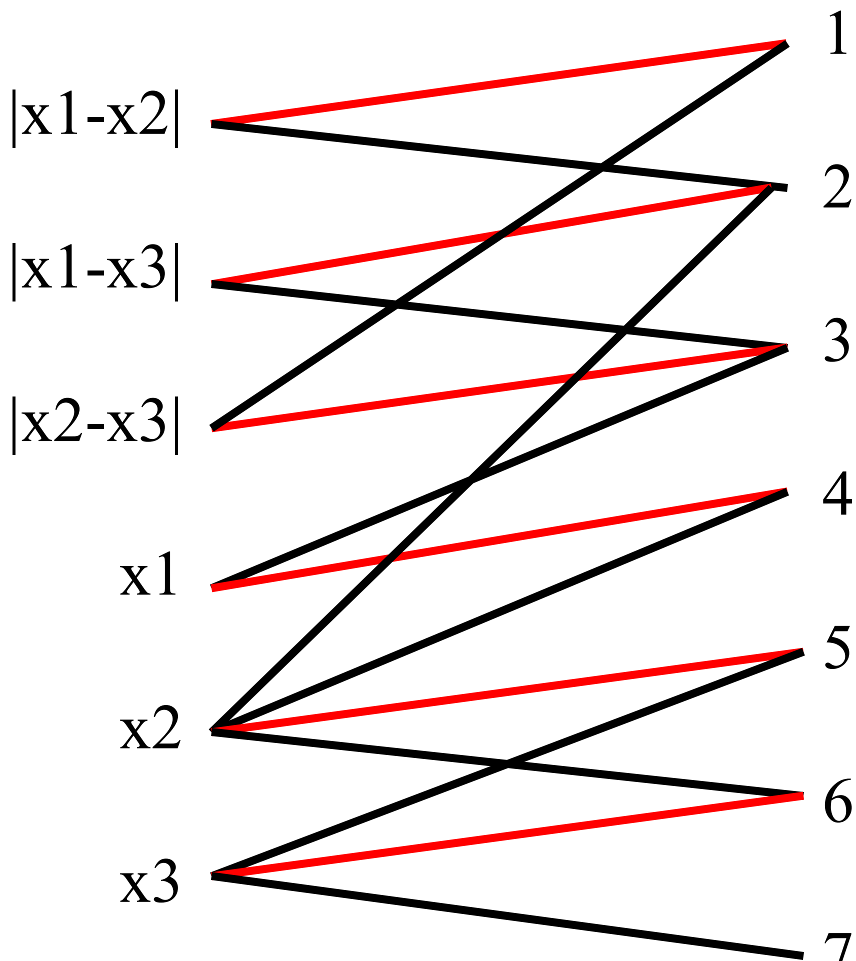
Dominance rules

- A dominance rule is a rule that eliminates some solutions that are not optimal, or some optimal solutions but not all
- This is a kind of symmetry breaking in regards to the optimality
- An example is given in the resolution of the next problem

A bad model?

- Golomb ruler (see CSP lib):
  "A Golomb ruler may be defined as a set of n integers $0=x1 < x2 < \ldots < xn$ s.t. the $n(n-1)/2$ differences $(xj - xi)$ are distinct. Goal minimize xn."
- with CP difficult for $n > 13$.
- x1,…,xn = variables; (xi-xj)= variables. Alldiff involving all the variables.

Alldiff



Not a good solution
Bad incorporation of constraint |xi – xj| in alldiff

Plan

- Principles of Constraint Programming
- A rostering problem
- Modeling in CP: Principles
- **A difficult problem**
- A Network Design problem
- Modeling Over-constrained problems
- Discussion
- Conclusion

Even Round Robin

1		+ 2	- 3	+ 4	- 5	+ 6	- 7	+ 8
2		- 1	+ 4	- 6	+ 8	- 3	+ 5	- 7
3		- 8	+ 1	+ 5	- 7	+ 2	- 4	+ 6
4		+ 7	- 2	- 1	+ 6	- 8	+ 3	- 5
5		- 6	+ 8	- 3	+ 1	+ 7	- 2	+ 4
6		+ 5	- 7	+ 2	- 4	- 1	+ 8	- 3
7		- 4	+ 6	- 8	+ 3	- 5	+ 1	+ 2
8		+ 3	- 5	+ 7	- 2	+ 4	- 6	- 1

The schedule is given
You have to find the place where the games are played

+ home game
- away game

Even Round Robin

1		+ 2	- 3	+ 4	- 5	+ 6	- 7	+ 8
2		- 1	+ 4	- 6	+ 8	- 3	+ 5	- 7
3		- 8	+ 1	+ 5	- 7	+ 2	- 4	+ 6
4		+ 7	- 2	- 1	+ 6	- 8	+ 3	- 5
5		- 6	+ 8	- 3	+ 1	+ 7	- 2	+ 4
6		+ 5	- 7	+ 2	- 4	- 1	+ 8	- 3
7		- 4	+ 6	- 8	+ 3	- 5	+ 1	+ 2
8		+ 3	- 5	+ 7	- 2	+ 4	- 6	- 1

A **break** for a team is two consecutive home games or two consecutive away games

Home break

Away break

Goal: minimize the number of breaks

Model: Variables

1		+ 2	- 3	+ 4	- 5	+ 6	- 7	+ 8
2		- 1	+ 4	- 6	+ 8	- 3	+ 5	- 7
3		- 8	+ 1	+ 5	- 7	+ 2	- 4	+ 6
4		+ 7	- 2	- 1	+ 6	- 8	+ 3	- 5
5		- 6	+ 8	- 3	+ 1	+ 7	- 2	+ 4
6		+ 5	- 7	+ 2	- 4	- 1	+ 8	- 3
7		- 4	+ 6	- 8	+ 3	- 5	+ 1	+ 2
8		+ 3	- 5	+ 7	- 2	+ 4	- 6	- 1

place variables:
for each team i and for each period j:
a 0-1 variable Pij is defined

break variables:
for each team and for each pair of consecutive period:
a 0-1 variable Bij is defined.
Bij=1 means that the team i has a break for the games played at period J and j+1

Model: Objective

1		+ 2	- 3	+ 4	- 5	+ 6	- 7	+ 8
2		- 1	+ 4	- 6	+ 8	- 3	+ 5	- 7
3		- 8	+ 1	+ 5	- 7	+ 2	- 4	+ 6
4		+ 7	- 2	- 1	+ 6	- 8	+ 3	- 5
5		- 6	+ 8	- 3	+ 1	+ 7	- 2	+ 4
6		+ 5	- 7	+ 2	- 4	- 1	+ 8	- 3
7		- 4	+ 6	- 8	+ 3	- 5	+ 1	+ 2
8		+ 3	- 5	+ 7	- 2	+ 4	- 6	- 1

Objective Variable:

#B is the variable that counts the total number of breaks for the schedule

Model: Constraints

1		+ 2	- 3	+ 4	- 5	+ 6	- 7	+ 8
2		- 1	+ 4	- 6	+ 8	- 3	+ 5	- 7
3		- 8	+ 1	+ 5	- 7	+ 2	- 4	+ 6
4		+ 7	- 2	- 1	+ 6	- 8	+ 3	- 5
5		- 6	+ 8	- 3	+ 1	+ 7	- 2	+ 4
6		+ 5	- 7	+ 2	- 4	- 1	+ 8	- 3
7		- 4	+ 6	- 8	+ 3	- 5	+ 1	+ 2
8		+ 3	- 5	+ 7	- 2	+ 4	- 6	- 1

place-opponent constraint:
i plays k at home at period j
is equivalent to
k plays i away at period j

break constraint:
if a team i plays for two consecutive periods j and j+1 at home or away then Bij=1 and conversly.

$\#B=\sum_i \sum_j B_{ij}$ $(i=1..n, j=1..n-2)$

First test

#teams	6	8	10	12
#bk	16	3,899	352,701	?
time (s)	0	0.7	73	?

First test

#teams	6	8	10	12
#bk	16	3,899	352,701	?
time (s)	0	0.7	73	?

The goal is 20

We have to work!

Symmetry?

1		+ 2	- 3	+ 4	- 5	+ 6	- 7	+ 8
2		- 1	+ 4	- 6	+ 8	- 3	+ 5	- 7
3		- 8	+ 1	+ 5	- 7	+ 2	- 4	+ 6
4		+ 7	- 2	- 1	+ 6	- 8	+ 3	- 5
5		- 6	+ 8	- 3	+ 1	+ 7	- 2	+ 4
6		+ 5	- 7	+ 2	- 4	- 1	+ 8	- 3
7		- 4	+ 6	- 8	+ 3	- 5	+ 1	+ 2
8		+ 3	- 5	+ 7	- 2	+ 4	- 6	- 1

Problem:
Difficult to identify one

Study of the problem

- There is no schedule with less than n-2 breaks
- Proof: consider 2 teams a, b
  Assume a has no break
  Assume b has no break: this means that a and b "alternate" (a is + - + - + - … and b is - + - + - + …) because a plays b at a moment.
  Now consider any other team c, then c has necessarily a break because c cannot alternate simultaneously with a and with b

N-2 as lower bound

- If the minimal value is close to n-2 then it is more interesting to try successively the values from n-2 w.r.t. an increasing order than finding a first solution and then trying to reduce the objective value

Relevant constraints

- For each two consecutive periods the number of away break (- -) is equal to the number of home breaks (+ +)
- Proof: for each period the number of + is equal to the number of -. We cannot have an odd number of non breaks.
- Corollary: #B is even

+	-
+	+
-	+
-	-

Second test

#teams	6	8	10	12
#bk	16	3,899	352,701	?
time (s)	0	0.7	73	?

#teams	6	8	10	12
#bk	5	970	101,844	?
time (s)	0	0.2	20.8	?

Relevant constraints

- Suppose that for a team we have
  + . - . . . .
  and exactly one break is required
  then we can deduce: + . - + - + -
- Property: #Bi(j,k): number of break for team i between period j and k

  j a period, k a period with k = j + q
  Pij=Pik ⇔ #Bi(j,k) has the parity of q

Third test

#teams	8	10	12	14
#bk	970	101,844	?	?
time (s)	0.2	20.8	?	?

#teams	8	10	12	14
#bk	226	11,542	135,129	?
time (s)	0.1	4.0	55.3	?

Relevant constraint

- As proved at the beginning: there are at most two teams with no break

Fourth test

#teams	8	10	12	14
#bk	226	11,542	135,129	?
time (s)	0.1	4.0	55.3	?

#teams	8	10	12	14
#bk	41	846	2,435	1,716,513
time (s)	0.1	0.4	1.37	904.4

Variable-value strategy

- Strategy:
  1) The #Bi variables with domain-min
  2) The place variables for the first period
  3) the break variables by trying first value 1
  4) the place variables

Fifth test

#teams	8	10	12	14
#bk	41	846	2,435	1,716,513
time (s)	0.1	0.4	1.37	904.4

#teams	8	10	12	14
#bk	41	846	2,209	711,408
time (s)	0.1	0.4	1.18	397.1

Dominance rules

i	+ j	x
j	- i	y

break

i	+ j	+ x
j	- i	+ y

break

break

i	+ j	+ x
j	- i	- y

i	+ j	- x
j	- i	+ y

break

i	+ j	- x
j	- i	- y

Dominance rules

DR: If i < j then break on i is forbidden for the two first period

break	i	+ j	+ x
	j	- i	+ y

break	i	+ j	+ x
break	j	- i	- y

	i	+ j	- x
	j	- i	+ y

	i	+ j	- x
break	j	- i	- y

Dominance rules

DR: If i < j then break on i is forbidden for the two first period

This is possible. If there is a break then if we swap the location the number of break is never increased

~~break~~	i	- j	+ x
break	j	+ i	+ y

	i	+ j	- x
	j	- i	+ y

~~break~~	i	- j	+ x
~~break~~	j	+ i	- y

	i	+ j	- x
break	j	- i	- y

Dominance rules

- The dominance rule can be defined for the first two column and for the last two columns
- It is also possible to define dominance rules for the middle, but this is quite complex.

Final result

- 16 teams in 5s
- 18 teams in 20s
- 20 teams in 200s

Plan

- Principles of Constraint Programming
- A rostering problem
- Modeling in CP: Principles
- A difficult problem
- **A Network Design problem**
- Modeling Over-constrained problems
- Discussion
- Conclusion

The ROCOCO Project

- France Telecom R&D ISE
  - Problem and benchmark definition
  - Algorithm validation
- Research laboratories: INRIA Numopt, LRI Orsay, PRiSM Versailles, Evry, …
  - Lower bounds: Lagrangean relaxation, column generation, cuts
  - Optimization techniques: genetic algorithms
- ILOG
  - Optimization techniques: constraint programming, mixed integer programming, column generation

The Problem (1)

- Routing of Communications
  - Mono-routing: each demand from a point p to a point q must follow a unique path
- Dimensioning of Links
  - The capacity of each link must exceed the sums of the demands going through the link
- Additional Constraints
  - Depend on the customer for whom the network is designed

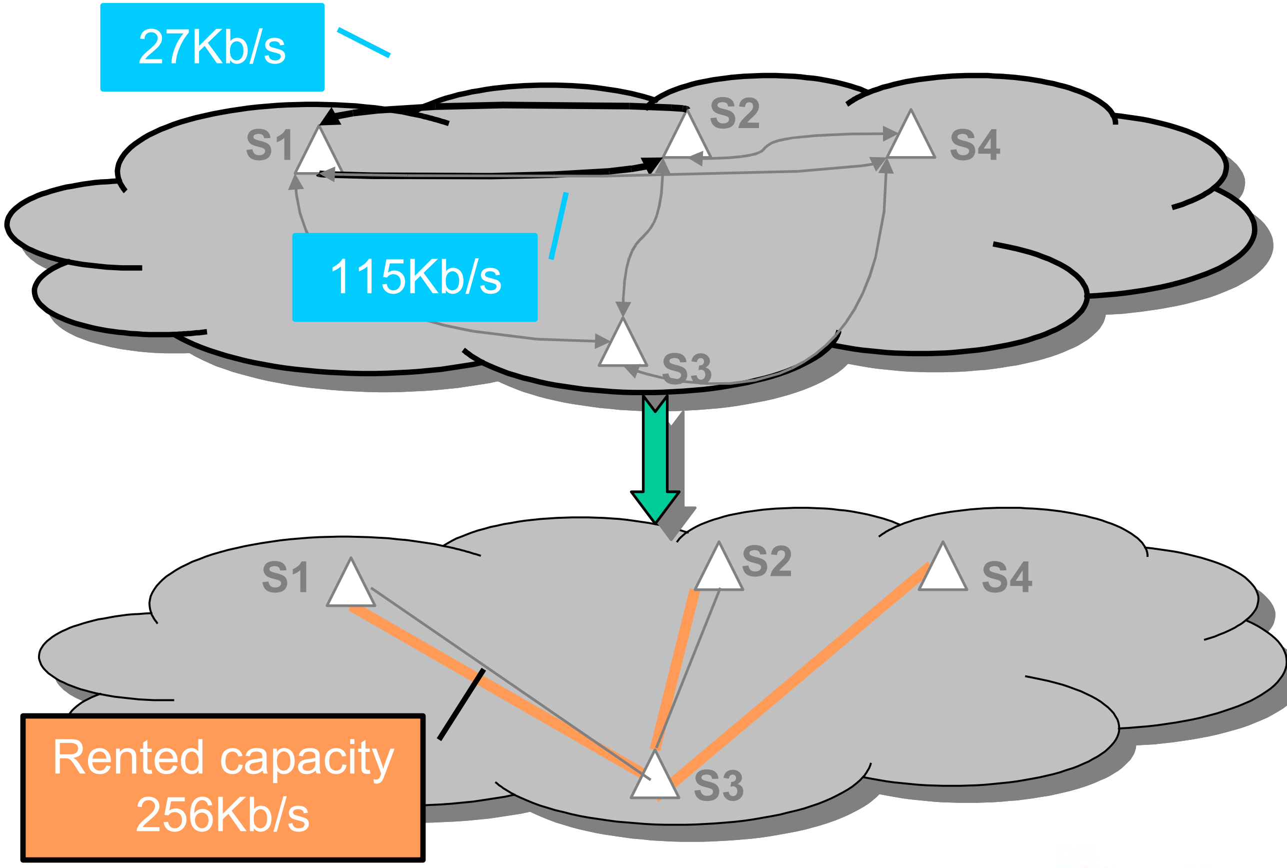
The Problem (2)

Data:

- Customer traffic demands
- Possible links, capacities and costs

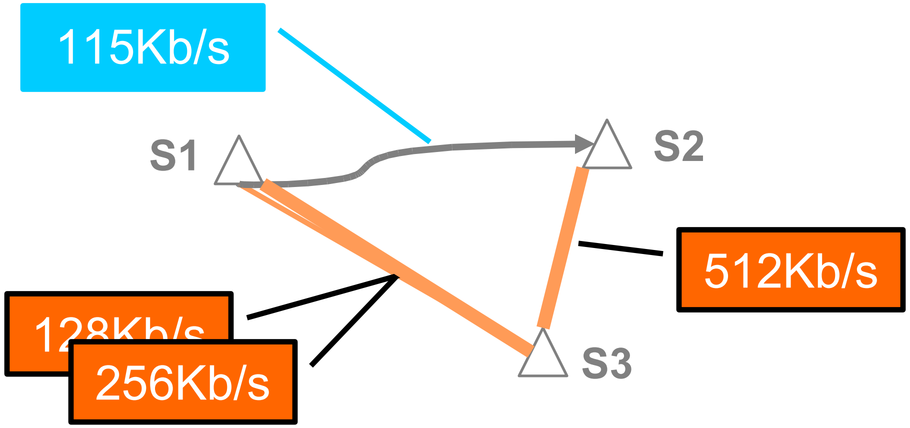
Result:

- **Minimal cost network able to simultaneously respond to all the demands**
- **Route for each demand**



The Problem (3)

- Cost minimization principle
  - Traffic demands share link capacities



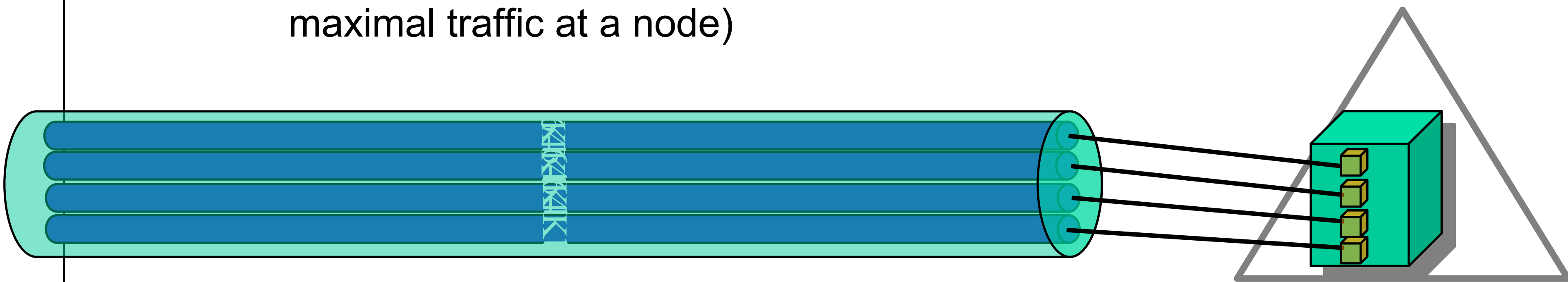
The Problem (4)

Demands share links

- $\sum \text{demands}_{i \to j} \leq \text{capacity}_{i \to j}$
- Technological constraints

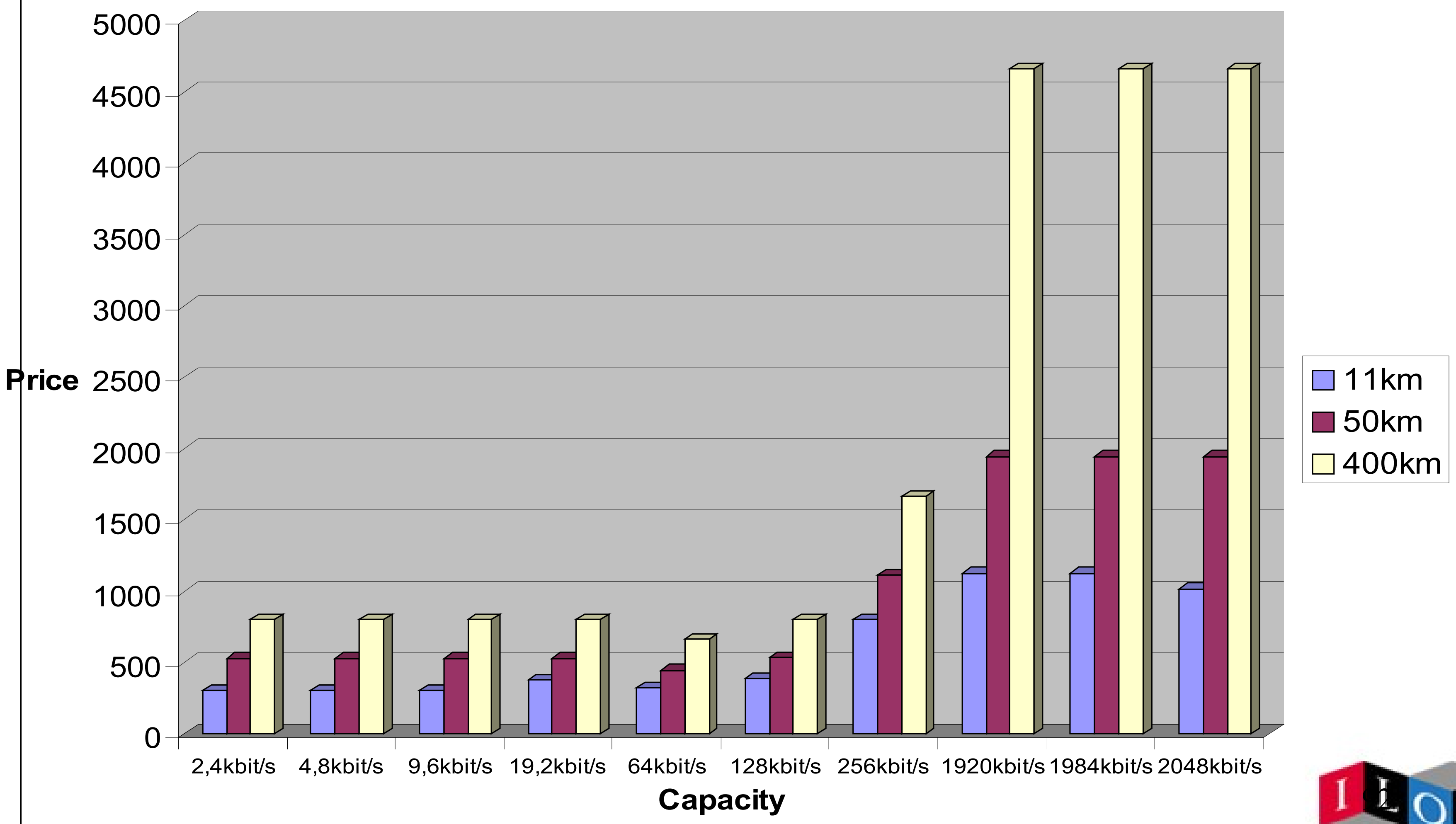
The Problem (5)

- Side constraints
  - Quality of service
  - Reuse of existing equipment (limit on the number of ports, maximal traffic at a node)



  - Commercial and legal constraints
  - Possible future network evolution
  - Network management (e.g., traffic concentration)

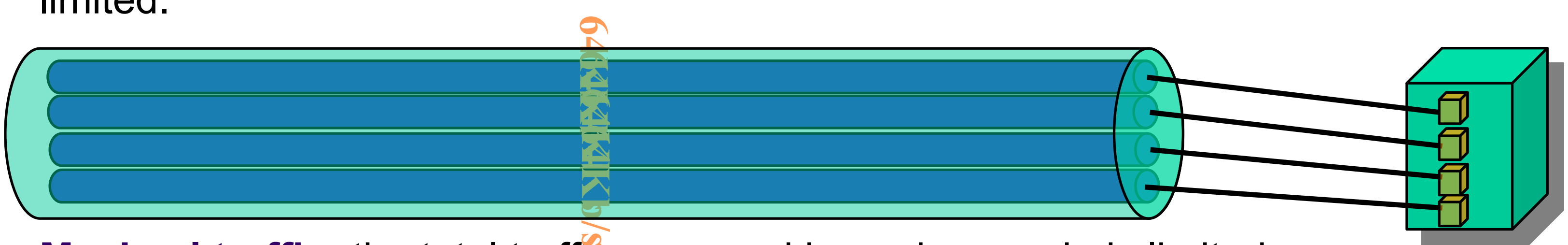
Data



Price
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
2,4kbit/s
4,8kbit/s
9,6kbit/s
19,2kbit/s
64kbit/s
128kbit/s
256kbit/s
1920kbit/s
1984kbit/s
2048kbit/s
Capacity
11km
50km
400km

Optional Constraints

- **Security:** some commodities to be secured cannot go through unsecured nodes and links
- **No line multiplication:** at most one line per arc.
- **Symmetric routing:** demands from node p to node q and demands from node q to node p are routed on symmetric paths.
- **Number of bounds (hops):** the number of arcs of the path used to route a given demand is limited.
- **Number of ports:** the number of links entering into or leaving from a node is limited.



- **Maximal traffic:** the total traffic managed by a given node is limited.

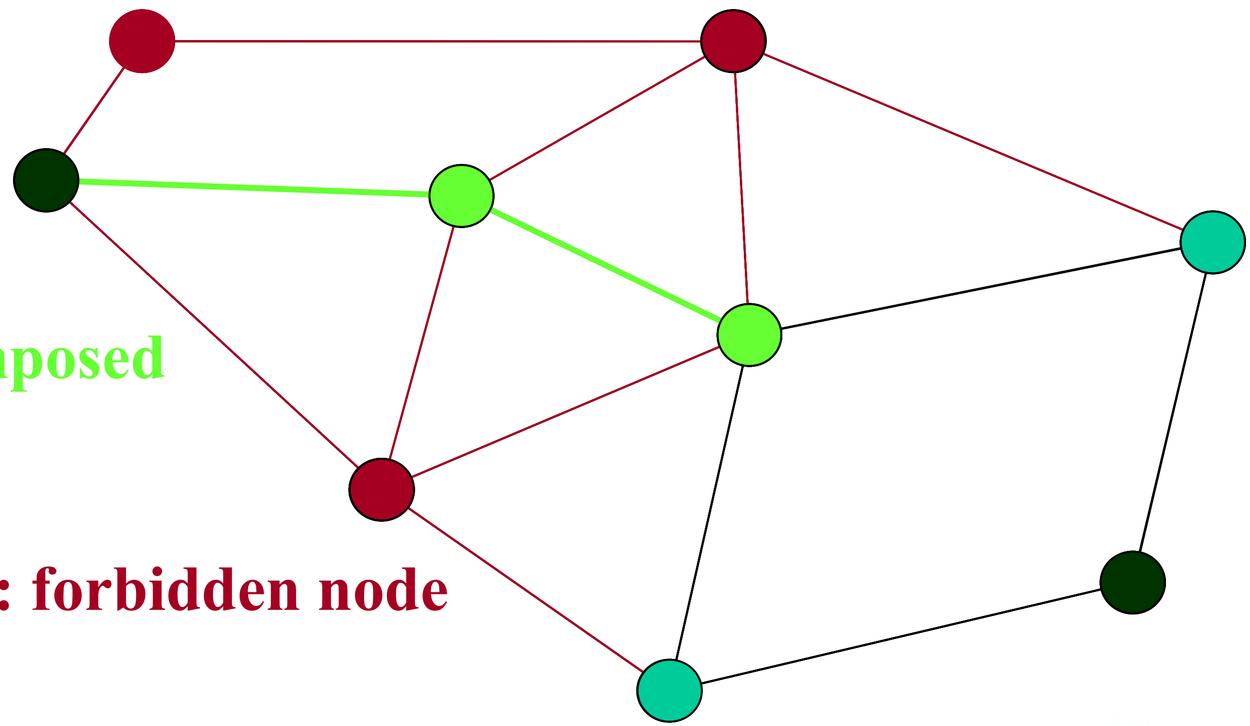
Numerical Characteristics

Mixed Integer Programming

Constraint Programming

- Routing variables: paths (D set variables)
  - A set of arcs joining the origin to the destination of the demand
  - Basic functions : impose or forbid an arc (or a node)
- Dimensioning variables: chosen capacity levels (M enumerated variables)
- Specific constraints and constraint propagation algorithms

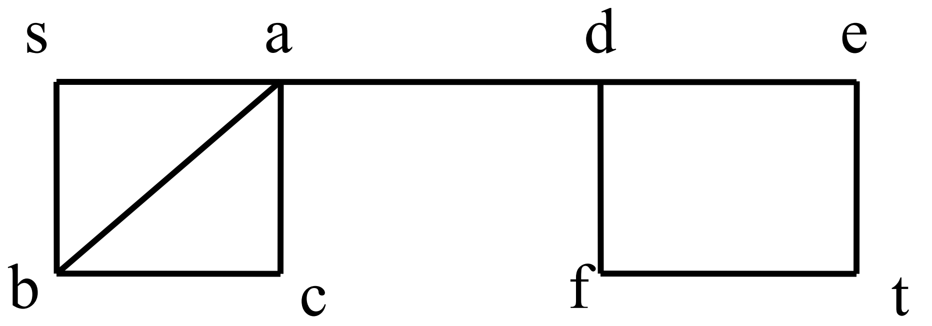
Constraint Programming



Path representation in CP

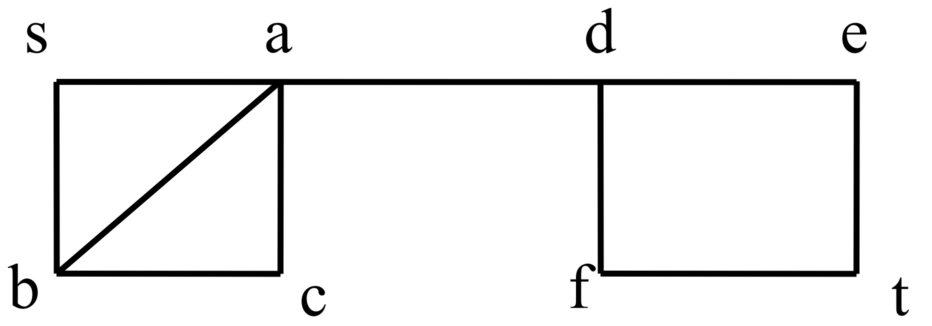
- “Classical” model:
  Graph represented by the nodes:
  One variable per node
  Value = possible neighboor
- Path from s to t: alldiff on nodes.

Path representation in CP



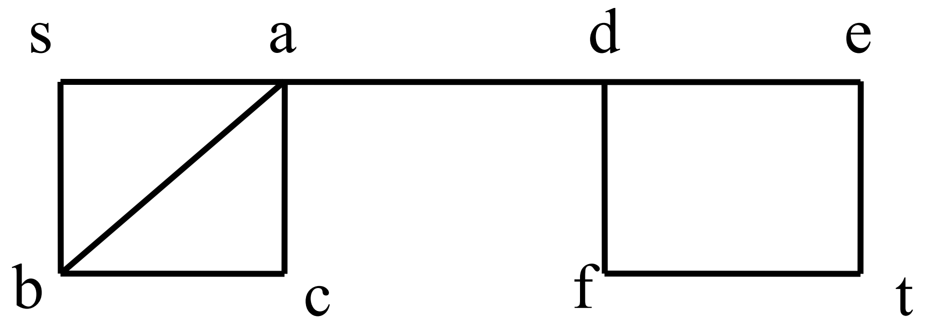
D(s)={a,b}, D(a)={s,b,c,d}, D(b)={s,a,c}, D(c)={a,b}
D(d)={a,e,f}, D(e)={d,t}, D(f)={d,t}, D(t)={s}

Path representation in CP



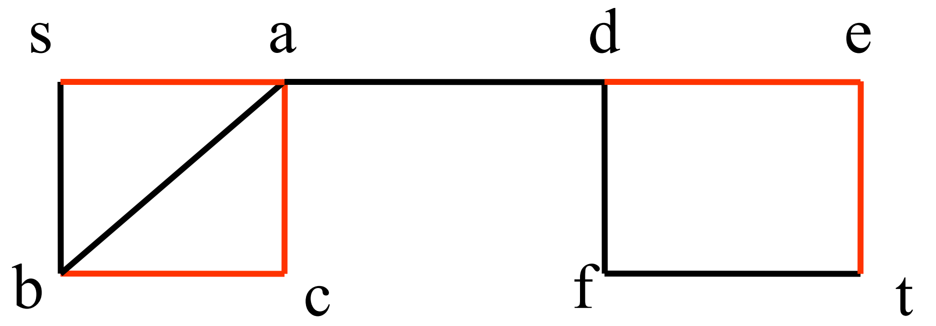
D(s)={a,b}, D(a)={s,b,c,d}, D(b)={s,a,c}, D(c)={a,b}
D(d)={a,e,f}, D(e)={d,t}, D(f)={d,t}, D(t)={s}

Path representation in CP



Problem if some variables do not belong to the path:
What is the value assigned to these variables?

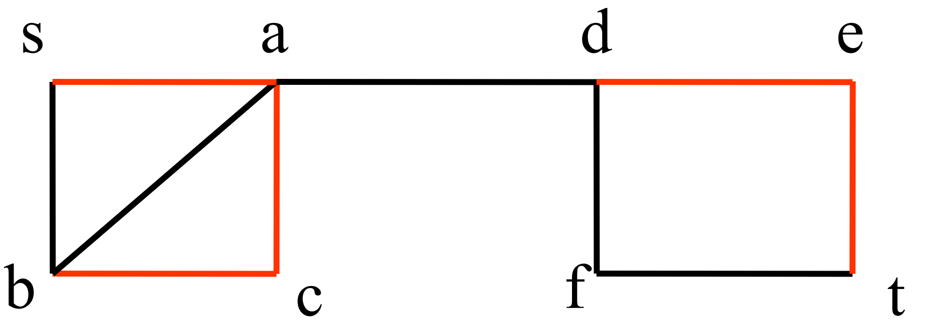
Path representation in CP



A dummy value is added to each domain: BAD IDEA

D(s)={a}, D(a)={c}, D(c)={b}, D(b)={dummyb},
D(d)={e}, D(e)={t}, D(f)={dummyf}, D(t)={s}

Path representation in CP



Loops are allowed (var links to itself): GOOD IDEA

D(s)={a}, D(a)={c}, D(c)={b}, D(b)={b},

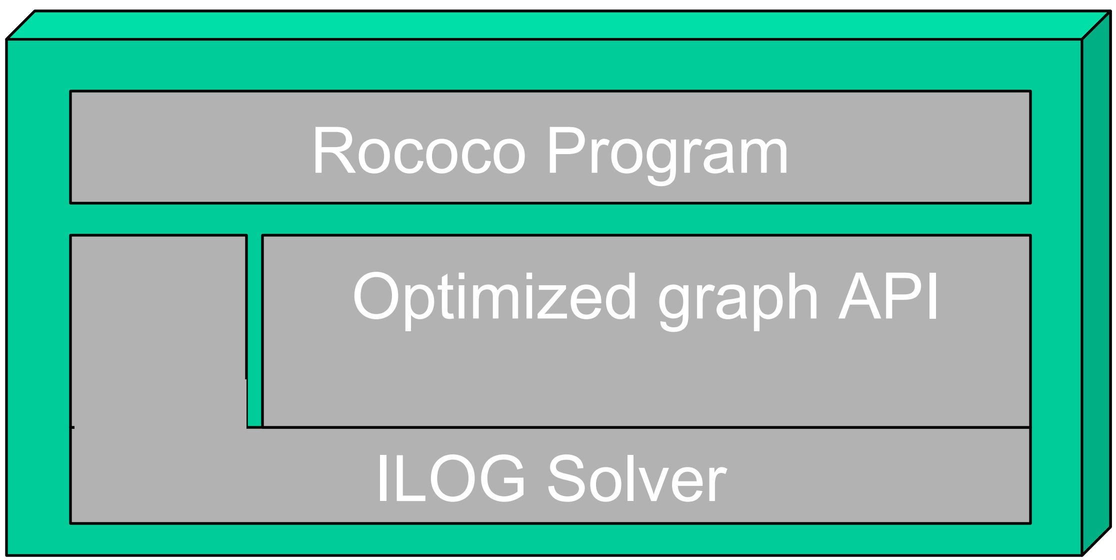
not possible: b has been already taken by c

Path representation in CP

- "Classical" model:
  - One var per node
  - Alldiff constraint: cost for the matching: O(m) per modification

Rococo in CP

- Manipulate only graph abstractions
  - Nodes
  - Valued links
  - Shortest paths …



New model:

- General point of view: we search for a subgraph.
  Two entities:
  - Digraph class
  - DigraphVar class

- A DigraphVar is a subgraph of a digraph w.r.t. properties, for instance: path.
  It is defined from a Digraph

-

New model:

- General point of view: we search for a subgraph.
  Two entities:
  - Digraph class
  - DigraphVar class

- A DigraphVar is a subgraph of a digraph w.r.t. properties, for instance: path.
  It is defined from a Digraph

- API similar to setvar API

Digraph

- **class IlcDigraph** {

```
IlcDigraph(IlcInt nbNodes,IlcIntArray from,IlcIntArray to);
IlcInt getNbNodes()const;
IlcInt getNbArcs()const;
IlcInt getNbOutgoingArcs(const IlcInt node) const;
IlcInt getNbIncomingArcs(const IlcInt node) const;
IlcInt getEmanatingNode(const IlcInt arc)const;
IlcInt getTerminatingNode(const IlcInt arc)const;
IlcInt getFirstOutgoingArc(const IlcInt node)const;
IlcInt getNextOutgoingArc(const IlcInt node, const IlcInt arc)const;
IlcInt getFirstIncomingArc(const IlcInt node)const;
IlcInt getNextIncomingArc(const IlcInt node, const IlcInt arc)const;
```

Digraph Variable

- **Class DigraphVar**{

```
IlcDigraphVar(IlcManager m, IlcDigraph g);
IlcIntSetVar getNodesVar()const;  IlcIntSetVar getArcsVar()const;
IlcIntSetVar getSourcesVar()const;  IlcIntSetVar getSinksVar()const;
IlcBool isBound()const;
IlcBool isAPath()const;
// accessors
IlcBool isArcRequired(IlcInt arc)const;
IlcBool isArcPossible(IlcInt arc)const;
IlcBool isNodeRequired(IlcInt node)const;
IlcBool isNodePossible(IlcInt node)const;
IlcBool isSourceRequired(IlcInt node)const;
IlcBool isSourcePossible(IlcInt node)const;
IlcBool isSinkRequired(IlcInt node)const;

IlcBool isSinkPossible(IlcInt node)const;
```

Digraph Variable

- **Class DigraphVar** {

```
// modificators
void removeAllOutgoingArcs(IlcInt node)const;
void removeAllIncomingArcs(IlcInt node)const;
void removeAllOutgoingArcsButArc(IlcInt node, IlcInt arc)const;
void removeAllIncomingArcsButArc(IlcInt node, IlcInt arc)const;
void removeArcPossible(IlcInt arc)const;
void addArcRequired(IlcInt arc)const;
void removeNodePossible(IlcInt node)const;
void addNodeRequired(IlcInt node)const;
void removeSinkPossible(IlcInt node)const;
void addSinkRequired(IlcInt node)const;
void removeSourcePossible(IlcInt node)const;
void addSourceRequired(IlcInt node)const;
```

Digraph Variable

- **Class DigrapVar**{

```
// for iterations
IlcInt getFirstOutgoingArc(IlcInt node)const;
IlcInt getNextOutgoingArc(IlcInt node, IlcInt arc)const;
IlcInt getFirstIncomingArc(IlcInt node)const;
IlcInt getNextIncomingArc(IlcInt node, IlcInt arc)const;

IlcDigraph getDigraph()const;
IlcInt getNbIncomingArcs(IlcInt node)const;
IlcInt getNbOutgoingArcs(IlcInt node)const;
```

Digraph Variable

- **Class DigraphVar**{

```
// graph functions
IlcInt getFirstArcPossibleOnShortestPath(IlcIntDistanceFunctionI* d,
                                         const IlcInt source,
                                         const IlcInt sink,
                                         IlcInt dem=1)const;
IlcInt computeShortestPathDistance(IlcIntDistanceFunctionI* dist,
                                   const IlcInt source,
                                   const IlcInt sink,
                                   IlcInt dem=1)const;
IlcIntArray computeShortestPath(IlcIntDistanceFunctionI* dist,
                                const IlcInt source,
                                const IlcInt sink,
                                IlcInt dem=1)const;
```

Distance Function

- class IlcIntDistanceFunctionI{

```
class IlcIntDistanceFunctionI{
        IlcIntDistanceFunctionI(IlcDigraph g,IlcInt maxCost);
        virtual IlcInt getCost(IlcDigraphVar var,
                               IlcInt arc,
                               IlcInt dem)=0;
};
```

Path Constraints

- IlcConstraint IlcSimplePath(IlcDigraphVar g,
  IlcInt source,
  IlcInt sink);

- IlcConstraint IlcShortestPath(IlcDigraphVar g,
  IlcInt source,
  IlcInt sink,
  IlcIntVar obj,
  IlcIntDistanceFunctionI* dist);

Element constraint

- ```
  enum IlcGraphProperty {
          IlcNodeRequired=0L,
          IlcSourceRequired=1,
          IlcSinkRequired=2,
          IlcEmanatingRequired=3,
          IlcTerminatingRequired=4,
  IlcTraversedRequired=5,
          IlcArcRequired=20};
  ```

- ```
  IlcConstraint IlcGraphElement(IlcInt item,
                                IlcDigraphVarArray gvs,
                                IlcIntSetVar var,
                                IlcGraphProperty pte);
  ```

Selectors

- ```
  class IlcDigraphSelectDigraphVarI {
          IlcDigraphSelectDigraphVarI(){}
          virtual IlcInt select(IlcDigraphVarArray vars)=0;
  };
  ```

- ```
  class IlcDigraphSelectArcI {
          IlcDigraphSelectArcI(){}
          virtual IlcInt select(IlcDigraphVarArray vars, IlcInt index)=0;
  };
  ```

Selectors (cont'd)

```
class IlcDigraphSelectShortestPathArcI : IlcDigraphSelectArcI {
        IlcDigraphSelectShortestPathArcI(IlcIntDistanceFunctionI* fn);
        virtual IlcInt select(IlcDigraphVarArray vars, IlcInt index);
        virtual IlcInt getSource(IlcDigraphVarArray vars,
                                                  IlcInt index)=0;
        virtual IlcInt getSink(IlcDigraphVarArray vars, IlcInt index)=0;
        virtual IlcInt getDemand(IlcDigraphVarArray vars,
                                                    IlcInt index)=0;
};
```

Goals

- IlcGoal IlcDigraphRequireArc(IlcManager m,
  IlcDigraphVar digraph,
  IlcInt arcIndex);

- IlcGoal IlcDigraphRemoveArc(IlcManager m,
  IlcDigraphVar digraph,
  IlcInt arcIndex);

- IlcGoal IlcDigraphAddArc(IlcManager m,
  IlcDigraphVarArray vars,
  IlcInt index,
  IlcDigraphSelectArcI* selectArc);

Goals (cont'd)

- ```
  IlcGoal IlcDigraphInstantiate(IlcManager m,
                                IlcDigraphVarArray vars,
                                IlcInt index,
                    IlcDigraphSelectArcI* selectArc);
  ```

- ```
  IlcGoal IlcDigraphGenerate(IlcManager m,
                             IlcDigraphVarArray vars,
                             IlcDigraphSelectDigraphVarI* selectD,
                  IlcDigraphSelectArcI* selectArc);
  ```

Why not PathVar?

- Path is a property of a graph. We prefer to express properties by constraint
- In any cases, we need to be able to test if an object is a path/tree/cycle …

Rococo in CP

- Advantages of digraph variables:
  - Simple
  - Open to many additional constraints
  - Much more efficient than basic constraint programming (combines constraint programming with optimization algorithms on graphs)

Rococo in CP

- Search strategy: select the most important demand and the path for which the additional (marginal) cost for routing this demand is minimal
  - Shortest path problem with constraints
  - Successive constraints: impose the last arc, then the previous arc, ..., and finally the first arc of the shortest path
  - Each of these added constraints leads to creating a choice point: upon backtracking, the imposed arc is forbidden and a new shortest path, taking this interdiction into account, computed

Improvements of CP

- Direct constraint between variables representing the paths and variables representing the traffic through each node
- Use of Parallel Solver
  - A few lines of code
- Modification of the tree-search traversal strategy
  - Branch more close to the root of the tree

Results

#pb	CP deviation	MIP deviation	GC deviation
A04	0.00%	0.00%	0.00%
A05	0.00%	0.00%	0.00%
A06	0.00%	0.00%	0.00%
A07	0.01%	1.42%	0.60%
A08	0.69%	9.06%	5.11%
A09	1.25%	19.44%	12.85%
A10	1.57%	fail	fa
B10	10.62%	12.04%	13.40%
B11	19.20%	12.46%	11.70%
B12	13.49%	13.32%	9.62%
C10	1.84%	3.24%	2.72%
C11	5.90%	9.11%	17.83%
C12	16.20%	fail	12.26%

Results

- France Telecom considers that CP gives the most interesting result.
- CP approach has been optimized mainly for A series.
- A lot of work could be done for the other series
- Result of Column Generation comes from a PhD thesis (A. Chabrier) mainly dedicated to this problem

Pros and Cons of Different Techniques (1)

- Constraint Programming:
  - \+ Global constraints on paths
  - − The overall cost is a sum of many step functions (almost no propagation)
- Mixed Integer Programming:
  - \+ Sum objective handled with a global view
  - − No good model for mono-routing (in the relaxation, the LP solver provides a flow)
  - − Bad continuous relaxation of the step functions
- Column Generation:
  - \+ Sum objective handled with a global view
  - \+ A column is a path
  - − Bad continuous relaxation of the step functions

Pros and Cons of Different Techniques (2)

- Security
  - ± CP: Easy to model with logical constraints but no global propagation
  - − MIP, CG: Leads to lots of fractional values in the relaxation (e.g., routing a demand on two paths, each made of half-secure and half-unsecured links)
- No line multiplication
  - + CP, MIP, CG: Smaller problem
  - − MIP, CG: Impact on the continuous relaxation of the step functions
- Symmetric routing
  - + CP, MIP, CG: Smaller problem

Pros and Cons of Different Techniques (3)

- Number of bounds (hops)
  - \+ CG: Much less potential paths and paths much easier to generate (especially when the number of bounds is really small)
  - ± CP: More propagation but with more complex algorithm
  - − MIP: Easy to model (sum of 0-1 variables representing the presence of each arc in a path) but more fractional values in the relaxation
- Number of ports
  - ± CP: Easy to model with logical constraints but no global propagation
  - − MIP, CG: Requires additional integer variables (with fractional values in the relaxation)
- Maximal traffic
  - \+ MIP, CG: Linear constraints
  - − CP: Linear constraints with no global propagation

Plan

- Principles of Constraint Programming
- A rostering problem
- Modeling in CP: Principles
- A difficult problem
- A Network Design problem
- **Modeling Over-constrained problems**
- Discussion
- Conclusion

Over Constrained Problems

- No solution satisfies all the constraints
- What can we do?
- Some constraints have to be relaxed
  - Hard constraints: must be satisfied
  - Soft constraints: can be relaxed

Over constrained problems: outline

- Two problems
- Soft constraint and Filtering algorithm
- Applications involving global constraints that can be violated vs applications involving only local constraints that can be violated
- Constraints on violations
- How to model an over-constrained problem?
  - How to relax a constraint?
  - How to model constraints on violations?
- Discussion

Over constrained problems: outline

- **Two problems**
- Soft constraint and Filtering algorithm
- Applications involving global constraints that can be violated vs applications involving only local constraints that can be violated
- Constraints on violations
- How to model an over-constrained problem?
  - How to relax a constraint?
  - How to model constraints on violations?
- Discussion

Car sequencing

- Problem : computes the sequencing order of cars that will be built on an assembly line
- Many different types of cars can be built on an assembly line.
- **A car = a basic car + options** (color, motor, telephone, seats, …).
- **A car = a configuration of options**

Capacity of an option

- For practical reasons: a given option cannot be installed on every vehicle on the line.
- Consequence of smoothing constraints: local limits are imposed. Minimum granularity.
- **Capacity of an option**: ratio p/q, for any sequence of q cars on the line, at most p of them can have the option
- When p=1 called distance constraint

Car sequencing

opt	cap	configurations 0	1	2	3	4	5
0	1/2	X				X	X
1	2/3			X	X		X
2	1/3	X				X	
3	2/5	X	X		X		
4	1/5			X			
#cars		1	1	2	2	2	2

Car sequencing

-

opt	cap	configurations					
		0	1	2	3	4	5
0	1/2	X				X	X
1	2/3			X	X		X
2	1/3	X				X	
3	2/5	X	X		X		
4	1/5			X			
#cars		1	1	2	2	2	2

- Sequences 4,4 or 4,5 or 0,4 or 0,5 are forbidden

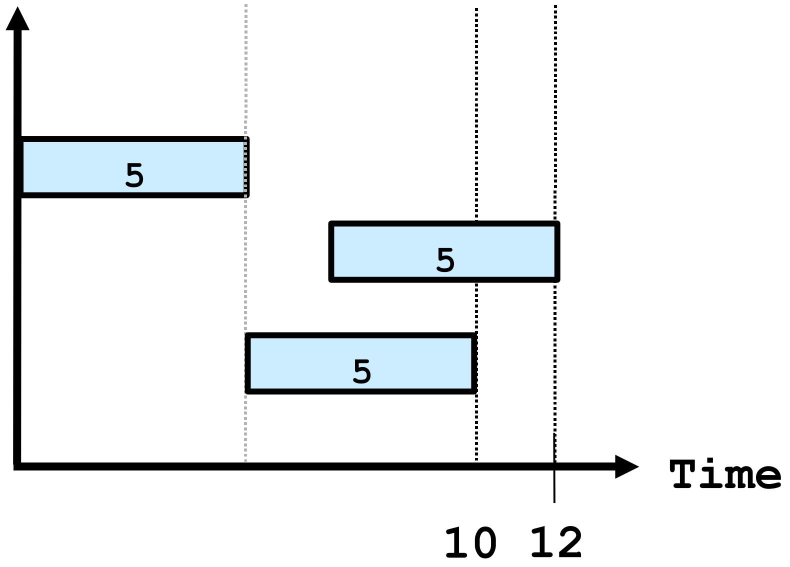
Car sequencing

opt	cap	configurations 0	1	2	3	4	5
0	1/2	X				X	X
1	2/3			X	X		X
2	1/3	X				X	
3	2/5	X	X		X		
4	1/5			X			
#cars		1	1	2	2	2	2

- Sequences 2,2,1 or 2,3,0 are allowed
- Sequences 2,2,3 or 5,3,2 are forbidden

Scheduling application

- Activities `A1, A2, A3` require a unary resource `R`
- Temporal constraints
  - The duration of each `Ai` is `5`
  - `A1` and `A2` start before `10`
  - `A3` ends at `12`



Over constrained problems: outline

- Two problems
- **Soft constraint and Filtering algorithm**
- Applications involving global constraints that can be violated vs applications involving only local constraints that can be violated
- Constraints on violations
- How to model an over-constrained problem?
  - How to relax a constraint?
  - How to model constraints on violations?
- Discussion

Soft constraint

- A soft constraint is a constraint that can be violated
- The violation can be associated with a cost that can be:
  - The same for any violation
  - Depends on the violation
- Example: $x < y$, if $x \geq y$ we can have
  - A fixed cost: cost = c
  - A cost depending on the violation: cost = x –y or cost = $(x-y)^2$

Soft constraint and Filtering algorithm

- When the violation is accepted this means that **we accept that any combination of values satisfies the constraint**.

Soft constraint and Filtering algorithm

- When the violation is accepted this means that **we accept that any combination of values satisfies the constraint**.
- Roughly, the constraint become an universal constraint associating a cost with any tuple, so **we loose the structure of the constraint**

Soft constraint and Filtering algorithm

- When the violation is accepted this means that **we accept that any combination of values satisfies the constraint**.
- Roughly, the constraint become an universal constraint associating a cost with any tuple, so **we loose the structure of the constraint**
- Problem with filtering algorithm (FA):
  - FA exploits the structure of the constraints
  - FA are not efficient when everything is possible!

Soft constraint and Filtering algorithm

- When the violation is accepted this means that **we accept that any combination of values satisfies the constraint**.
- Roughly, the constraint become an universal constraint associating a cost with any tuple, so **we loose the structure of the constraint**
- Problem with filtering algorithm (FA):
  - FA exploits the structure of the constraints
  - FA are not efficient when everything is possible!
- Filtering for soft depends mainly on back propagation. Problem with global constraints

Over constrained problems: outline

- Two problems
- Soft constraint and Filtering algorithm
- **Applications involving global constraints that can be violated vs applications involving only local constraints that can be violated**
- Constraints on violations
- How to model an over-constrained problem?
  - How to relax a constraint?
  - How to model constraints on violations?
- Discussion

Over Constrained Problem

Relaxation

- OC aspect in the propagation
- Use of soft constraints : that is constraints + cost
- Global objective function on the cost associated with constraints

⇒ **1 over-constrained problem is solved**

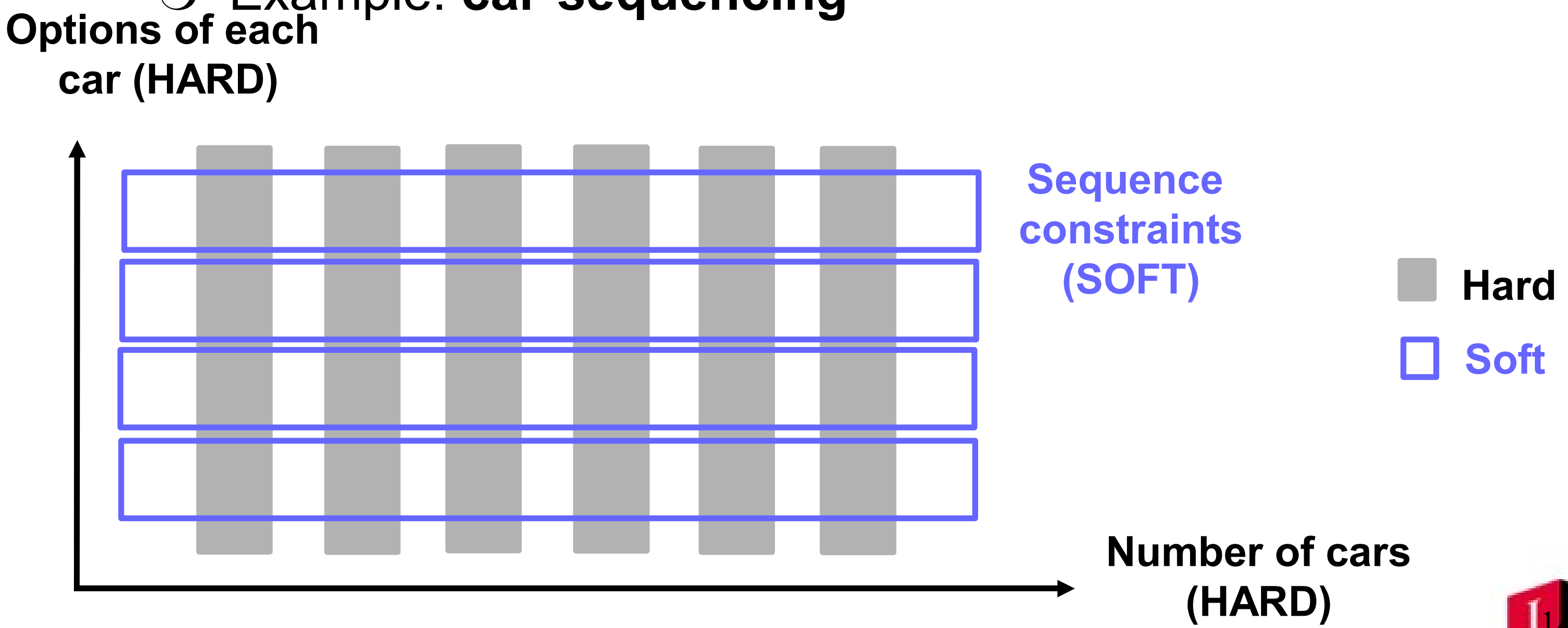
Decomposition

- No Soft Constraint. Only hard constraint
- A constraint which is violated is replaced by a relaxation of the constraint
- The relaxation is handled "by hand"

⇒ **n satisfaction problems are solved**

Applications

- Applications involving global constraints that can be violated
  - Each global constraint affects widely the problem
  - Example: **car-sequencing**



Applications

- Applications involving “global” soft constraints
  - ⇒ **Difficult to solve with a pure relaxation approach**
  - ⇒ **Decomposition methods are more adapted**

Applications

- Applications involving "global" soft constraints
  - ⇒ **Difficult to solve with a pure relaxation approach**

- A global constraint = conjunction of constraint. Violation of a global = violation of any constraint of the conjunction
- The problem is not easy even with efficient global constraints, so if the global constraints are removed and replaced by a lot of constraints that can be violated then we loose the strong filtering algorithms

Applications

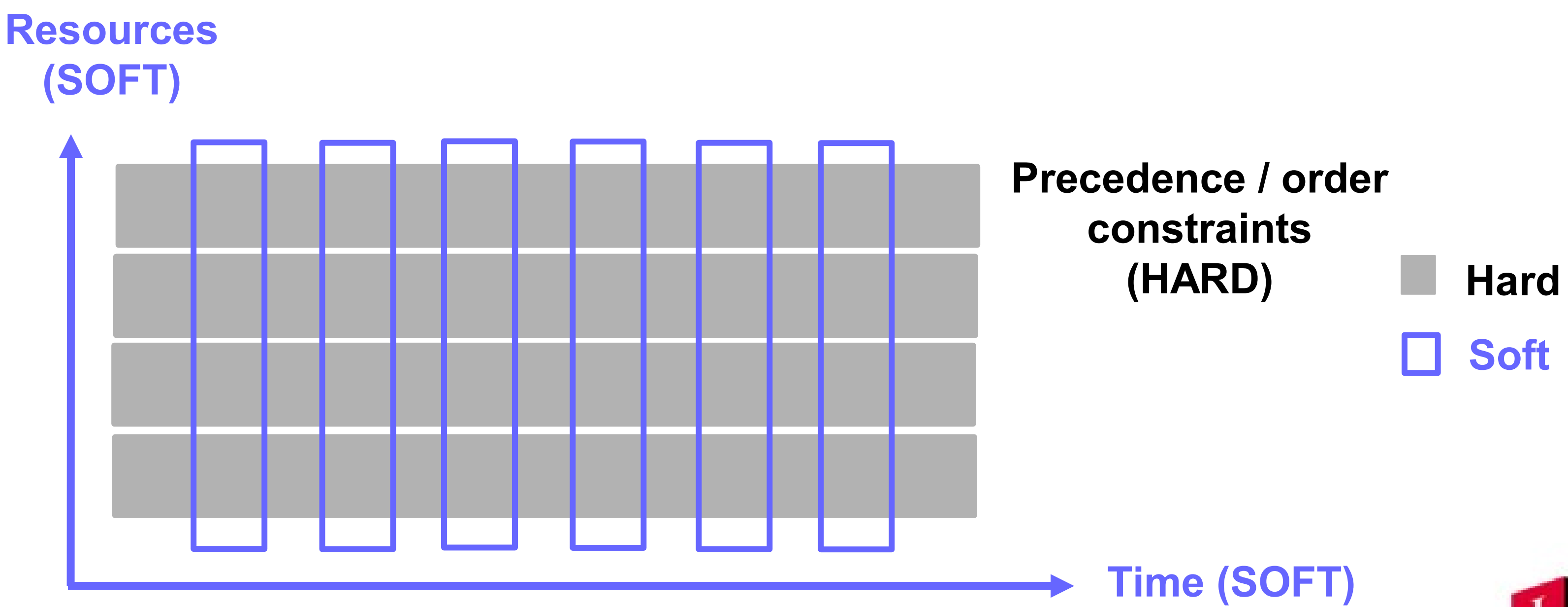
- Applications involving "global" soft constraints

  ⇒ **Decomposition methods are more adapted**

- A succession of problems are considered. In each problem the global constraint that are violated are relaxed by hand:
  - another global constraint replaces the previous one but it is less constrained. For instance a p/q option will be replaced by a p+1/q option. We will manage the relaxation
  - There is no objective, this is the user that controls the list of the problems that will be considered

Applications

- Application involving "Local" constraints that can be violated
  - Example: **scheduling**

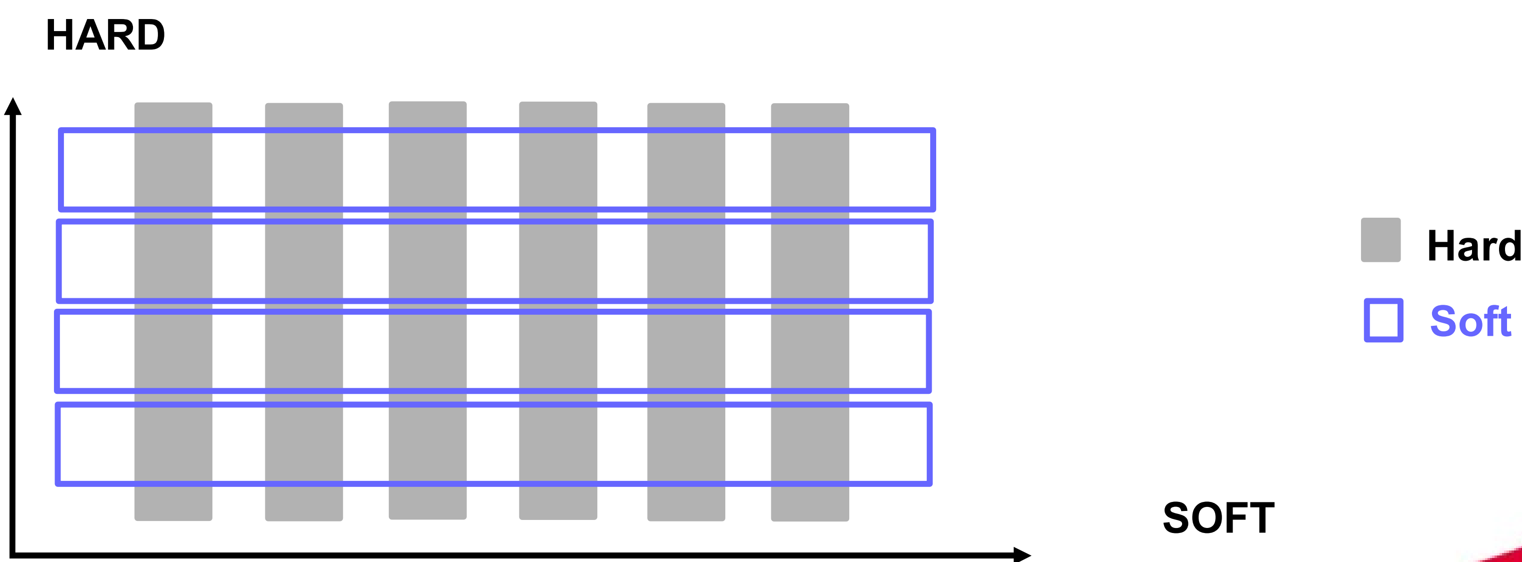


Applications

- Application involving “Local” soft constraints
  - ⇒ **Decomposition methods can be used**
  - ⇒ **Relaxation methods can be used**

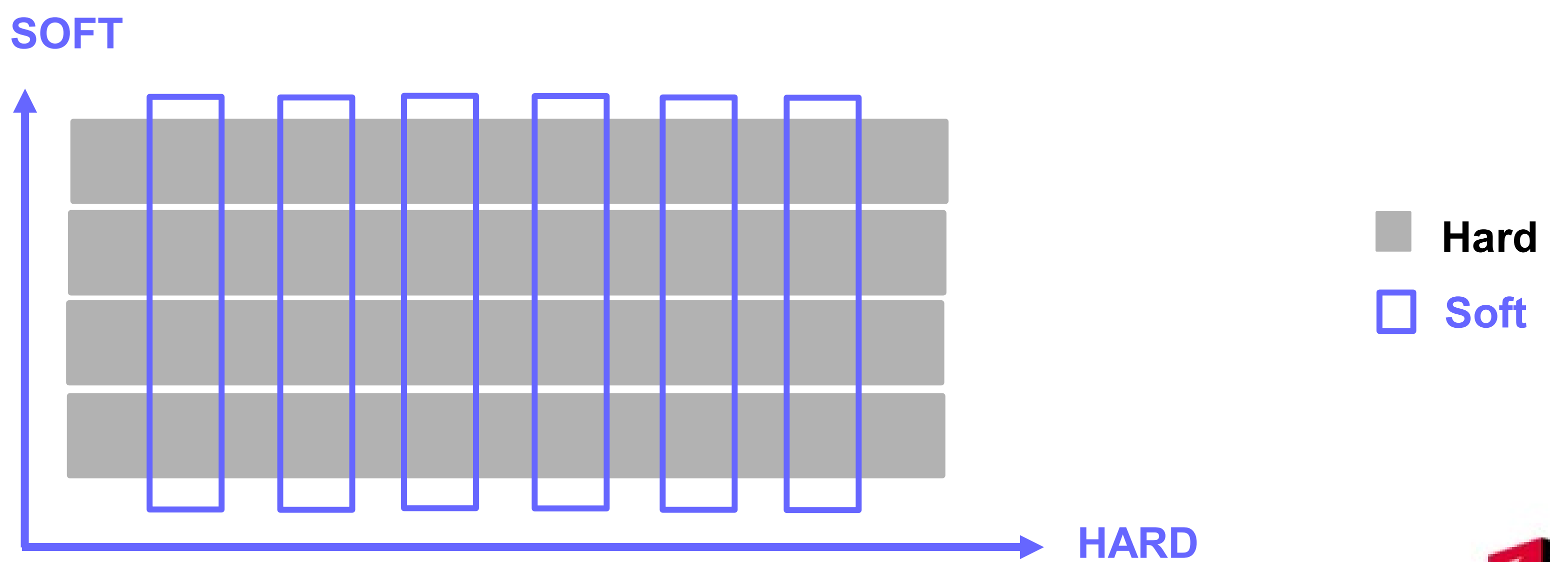
Applications involving global constraints that can be violated

- Over constrained problems solved by a succession of satisfiability problems. Each problem is managed by the user.



Applications involving local constraints that can be violated

- All the constraints are relaxed (that is we accept the violation) then an optimization problem is solved. The objective is to minimize the sum of the violation costs



Over constrained problems: outline

- Two problems
- Soft constraint and Filtering algorithm
- Applications involving global constraints that can be violated vs applications involving only local constraints that can be violated
- **Constraints on violations**
- How to model an over-constrained problem?
  - How to relax a constraint?
  - How to model constraints on violations?
- Discussion

Constraints on violations

- In real world problems, the goal is much more complex than minimizing the number of constraint violations

⇒ Rules on violations

  - Distinction between hard and soft constraints
  - Priorities
  - Control of distribution of violations in the constraint network : well balancing of violation (homogeneity) is generally required
  - Specific dependencies between constraints

Constraints on violations: Priorities

- All the constraints have not the same importance
- Goal: favor the satisfaction of the most important ones

(C1: hard constraint)
C2: crucial
C3: important
C4: low importance
C5: preference

Well balancing of violations

- In many real-world applications, violations must generally be homogeneously distributed in the constraint network
- More complex rules with respect to distribution of violations are sometimes required

Well balancing of violations

- Example
  - worker W_1: preferences = { C_{11}, C_{12}, C_{13}, ... }
  - worker W_2: preferences = { C_{21}, C_{22}, C_{23}, ... }
  - worker W_3: preferences = { C_{31}, C_{32}, C_{33}, ... }
- A schedule such that some workers have all their preferences satisfied and some other have no preference satisfied is not acceptable
- For each Wi:
  - At least j constraints satisfied
  - At least j constraints satisfied, and at least k constraint violated with degree < m, etc.
- General idea: avoid to have hard work periods and then almost nothing to do. True for people or for machine

Constraints on violations: Dependencies

- Expressing specific dependencies is generally required
- Example
    - « if C_1 and C_2 are violated then C_3 must be satisfied »

Over constrained problems: outline

- Two problems
- Soft constraint and Filtering algorithm
- Applications involving global constraints that can be violated vs applications involving only local constraints that can be violated
- Constraints on violations
- **How to model an over-constrained problem?**
  - **How to relax a constraint?**
  - **How to model constraints on violations?**
- Discussion

How to model over-constrained problems?

- How to relax a constraint?
- How to model usual constraints on violations

How to relax a constraint?

- Try to keep some structure in order to have efficient filtering algorithm
- Use meta constraints

Meta Constraint

- $s_i > 0$ expresses that C_i is violated (distance to satisfaction)
- $s_i = 0$ expresses that C_i is satisfied
- *D(*si*)* is an integer domain
- Each “soft” constraint is replaced by the disjunction:

$$[\ (s = 0) \wedge C\ ] \vee [\ (s > 0) \wedge \neg C\ ]$$

Meta Constraint

- Since valuations are expressed trough variables, constraints on these variables can be added in order to express “global rules” on violations

Max-SAT = Satisfiability Sum Constraint

- In the *ssc*, each constraint Ci is replaced by:

$$[\ (C_i \wedge (u_i = 0)) \vee (\neg C_i \wedge (u_i = 1))\ ]$$

- A variable *unsat* is used to express the objective:

$$[\ unsat = \sum_{i=1}^{\#Ci} u_i\ ]$$

Advantages of This Model

- Classical constraint optimization problem
  - Direct integration into a solver
  - Any search algorithm can be used, not only a Branch and Bound based one.
- When a value is assigned to $u_i \in U$, the filtering algorithm associated with C_i (resp. $\neg C_i$) can be used
- No hypothesis is made on constraints (arity)

Advantages of This Model

- **Integration of cost within the constraint**
  Costs as a variable:
  - the costs of violations have a structure:
    if ($x \leq y$) is violated then cost = $x - y$
    We can use this information.
- General definitions of cost of violations
- Global soft constraints
- Constraints on violations can be easily defined

Use of the structure of the violation: $x \leq y$

- Structure
  - If the constraint is satisfied then *cost* = 0
  - If the constraint is violated then *cost* = x - y
- Filtering Algorithm:
  - D(x) = [90000,100000], D(y) = [99990,200000]
  - We deduce immediately max(cost)= max(x) - min(y) = 10

General definition of the cost of violation

- Two different general costs:
  - Variables based violation cost
  - Primal Graph Based violation cost

- Some others see papers at CP-AI-OR’04 (Beldiceanu and Petit) and papers at workshop on soft constraints at CP’04.

Variable based violation cost

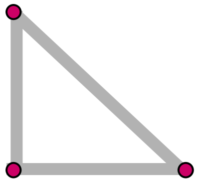
- How many variables must be removed to satisfy the constraint?
- Alldiff({x1,x2,x3,x4,x5})
  (a,a,a,b,b) cost = 3
  (a,a,a,a,b) cost = 3

Primal graph based partition cost

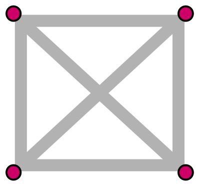
- For a global constraint corresponding to a conjunction of constraints. Number of the constraints in the conjunction that are violated
- Alldiff({x1,x2,x3,x4,x5})
  (a,a,a,b,b) cost = triangle(a,a,a) + pair (b,b)
  = 3 + 2 =5
  (a,a,a,a,b) cost = quadrangle (a,a,a,a)
  = 6

All Different constraint

The same value assigned to 2 variables $\rightarrow$ 1 violation



The same value assigned to 3 variables $\rightarrow$ 3 violations



The same value assigned to 4 variables $\rightarrow$ 6 violations

n variables $\rightarrow$ $n(n-1)/2$ violations

Meta-constraints for Expressing Homogeneity

- Example: time tabling problem
  - *n* office workers express some preferences
  - for each one at least *j* preferences should be satisfied

→ ***n* cardinality constraints,** one for each subset of state variables Si corresponding to the set of preference constraints of one worker Wi : « **at least *j* times value 0 assigned to variables in Si** »

Meta-constraints for Expressing Dependencies

- Example
  - If C_1 and C_2 are violated then C_3 must be satisfied

$$[((s_1 > 0) \wedge (s_2 > 0)) \Rightarrow (s_3 = 0)]$$

Over constrained problems: outline

- Two problems
- Soft constraint and Filtering algorithm
- Applications involving global constraints that can be violated vs applications involving only local constraints that can be violated
- Constraints on violations
- How to model an over-constrained problem?
  - How to relax a constraint?
  - How to model constraints on violations?
- **Discussion**

Discussion

- Well balancing of violations can be difficult to express.
- Real life example: Capacity constraints (smoothing constraints are soft): p/q.
  - Violation: $(p'-p)^2/q^2$, then minimization of the variance.
  - Signification: consider 1/5 and ½,
    - a violation of 1 is less important for 1/5 than for 1/2, so q^2 is considered
    - two violations of 1 are less important than one violation of 2, so $(p'-p)^2$ is considered
  - Difficult to model

Discussion

- A quite interesting approach has been proposed by L. Perron and P. Shaw (see in their CP'04 paper) for car sequencing
- 100 cars to schedule with a book order of 100 cars. Accept to have dummy cars (perfect cars), that is extend the book order -> 110 cars
- Then try to minimize the number of perfect cars
- Then replace perfect cars by real cars
- Seems very interesting because the model of the initial problems is not change at all and we work only with hard constraints

Meta constraints vs other models

- Valued CSP is not able to take into account the structure of the violated constraints
- Valued CSP contains no back propagation because the cost is not a variable
- Valued CSP are not able to manage constraints on violation
- If we represent constraints on violations by constraints we do not need to invent a new XXX Csp for every constraint on violations

Plan

- Principles of Constraint Programming
- A rostering problem
- Modeling in CP: Principles
- A difficult problem
- A Network Design problem
- Modeling Over-constrained problems
- **Discussion**
- Conclusion

Hard problem

- Consider a problem P that you are unable to solve. How can you improve the resolution?

Hard problem

- Consider a problem P that you are unable to solve. How can you improve the resolution?
- By identifying hard sub-problems H of P

Hard problem

- Consider a problem P that you are unable to solve. How can you improve the resolution?
- By identifying hard sub-problems H of P
- By improving the resolution of some sub-problems R of P and by using filtering algorithm for R.

Problem

- How can we identify a sub-problem R for which we can improve its resolution?
- How can we write a specific filtering algorithm for this sub-problem R?
- This is time-consuming and not necessarily worthwhile.

Improving the resolution of R

- R can be viewed as a global constraint:
  An allowed tuple of this constraint is a solution of R and conversely.
- Consistency of the constraint = R has a solution.

GAC-Schema: instantiation

- List of allowed tuples
- List of forbidden tuples
- Predicates
- Any OR algorithm
- Solver reentrace

GAC-Schema

- Idea:

  tuple = solution of the constraint

  support = valid tuple

  - while the tuple is valid: do nothing
  - if the tuple is no longer valid, then search for a new support for the values it contains

- a solution (support) can be computed by any OR algorithm. A solution is needed not only the fact that there is one.

GAC-Schema: complexity

- CC complexity to check consistency (seek in table, call to OR algorithm): seek for a Support costs CC
- n variables, d values:
  for each value: CC
  for all values: O(ndCC)
- **For any OR algorithm** which is able to compute a solution, **Arc consistency** can be achieved in **O(ndCC).**

AC for R

- All the possible solutions of R are computed once and for all. They are saved in a database. GAC-Schema + allowed is used
- Only the combinations of values that are not solution are saved. GAC-Schema + forbidden
- Solutions are computed on the fly when we want to know if a value belongs to a current solution of R.

Improvement of the resolution

- Cryptographic problem

x11	x12	x13	x14
x21			
x31			
x41			

$\sum x1i = r1$

$\sum xi1 = c1$

$+ \text{Alldiff}(xij)$

Improvement of the resolution

- Cryptographic problem

x11	x12	x13	x14	$\sum x1i = r1$
x21				
x31				
x41				

$\sum xi1 = c1$

$+ \text{Alldiff}(xij)$

What happens if we have the global constraint: ($\sum$ xi + Alldiff(xi))?

Improvement of the resolution

- Model 1: Sum for each row and each column
  + Alldiff(xij)
- Model 2: (∑ xi + Alldiff(xi)) for each row and each column
  + Alldiff(xij)

5 x 5 model1 model2 pred

easiest	4	0	0	0	0.5	8.5
average	2,373	127	127	0.3	1.7	18
hardest	9,985	591	591	1.36	7.2	51

#backtracks time

Improvement of the resolution

- Model 1: Sum for each row and each column
  + Alldiff(xij)
- Model 2: (∑ xi + Alldiff(xi)) for each row and each column
  + Alldiff(xij)

6 x 6

	#backtracks model1	#backtracks model2	#backtracks pred	time model1	time model2	time pred
easiest	3	0	0	0.03	2.5	too long
average	75,548	281	281	26.2	6.7	too long
hardest	1,623,557	2,598	2,598	520	42	too long

Identification of the difficulty

- Golomb ruler (see CSP lib):
  "A Golomb ruler may be defined as a set of n integers 0=x1 < x2 < … < xn s.t. the n(n-1)/2 differences (xj - xi) are distinct. Goal minimize xn."
- with CP difficult for n > 13.

Identification of the difficulty

- Model1: x1,…,xn = variables; (xi-xj)= variables.
  Alldiff involving all the variables.
- Model2: diff1i = xi - x(i-1)
  Model1 + global constraint:
  Sum(diff1i)=xn and Alldiff(diff1i)
- Model3: diff2i= xi - x(i-2)
  Model1 + global constraint:
  Sum(diff1i)=xn and Alldiff(diff1i U diff2i)

Identification of the difficulty

	n=8				n=9				n=10			
	xn=34		xn=33		xn=44		xn=43		xn=55		xn=54	
	#bk	t	#bk	t	#bk	t	#bk	t	#bk	t	#bk	t
model1	22	0.3	297	0.4	213	0.4	1298	2.7	844	2.2	5326	19
model2	3	0.3	122	2.8	48	2.4	343	18.2	183	16.6	1967	161
model3	0	1.4	5	1.5	4	10.5	25	18.1	16	120	96	226

Identification of the difficulty

	n=8		n=9				n=10					
	xn=34		xn=33		xn=44		xn=43		xn=55		xn=54	
	#bk	t	#bk	t	#bk	t	#bk	t	#bk	t	#bk	t
model1	22	0.3	297	0.4	213	0.4	1298	2.7	844	2.2	5326	19
model2	3	0.3	122	2.8	48	2.4	343	18.2	183	16.6	1967	161
model3	0	1.4	5	1.5	4	10.5	25	18.1	16	120	96	226

Gain: #bk/4

Identification of the difficulty

	n=8				n=9				n=10			
	xn=34		xn=33		xn=44		xn=43		xn=55		xn=54	
	#bk	t	#bk	t	#bk	t	#bk	t	#bk	t	#bk	t
model1	22	0.3	297	0.4	213	0.4	1298	2.7	844	2.2	5326	19
model2	3	0.3	122	2.8	48	2.4	343	18.2	183	16.6	1967	161
model3	0	1.4	5	1.5	4	10.5	25	18.1	16	120	96	226

Gain: problem almost solved!

Sum(diff1i)=xn and Alldiff(diff1i U diff2i): involves **only n variables**

Pre-resolution of some parts of the problem GAC-Schema + allowed

- Configuration problem:
  5 types of components: {glass, plastic, steel, wood, copper}
  3 types of bins: {red, blue, green} whose capacity is red 5, blue 5, green 6
  Constraints:
  - red can contain glass, cooper, wood
  - blue can contain glass, steel, cooper
  - green can contain plastic, copper, wood
  - wood require plastic; glass exclusive copper
  - red contains at most 1 of wood
  - green contains at most 2 of wood
  For all the bins there is either no plastic or at least 2 plastic
  Given an initial supply of 12 of glass, 10 of plastic, 8 of steel, 12 of wood and 8 of copper; what is the minimum total number of bins?

Pre-resolution of some parts of the problem GAC-Schema + allowed

	#bk	time
standard model	1,361,709	430
GAC+allowed	12,659	9.7

GAC Schema + allowed

- It is important to be able to generate all solutions of a problem
- There is almost no efficient algorithms that are available
- Any idea is welcome

Conclusion

- The first model usually does not work
- Use global constraints as much as possible
- Try to identify difficult parts of the problems
- Try to find relevant constraints to help the solver
- Try to avoid linear model and try to think constraint and filtering

Conclusion

- Be creative!
- Be imaginative!

Conclusion

- Be creative!
- Be imaginative!

Slides and papers are available at:

www.constraint-programming.com/people/regin

Some References

- Global constraints
  - Alldiff constraint: J-C. Regin, AAAI-94
  - Global Cardinality constraint: J-C. Regin, AAAI-96
  - GAC-Schema: C. Bessiere and J-C. Regin, IJCAI-97
  - Sequence: J-C. Regin and J-F. Puget, CP'97
  - Sum with binary inequalities: J-C Regin and M. Rueher, CP'00
  - Gcc with costs: J-C Regin, CP'99 (alldiff with cost, sum of alldiff var)
  - Symmetric alldiff: J-C Regin, IJCAI-99
  - AllMinDistance, J-C Regin, ILOG Solver
  - Global constraint on set variables, J-C Regin, J-F Puget, D. Mailharro, ILOG Solver
  - Cardinality Matrix Constraint, J-C Regin, C Gomes, CP'04
- Modelization and Problem resolution
  - Subgraph Isomorphism, J-C Regin, PhD thesis, 1995
  - Minimization of the number of breaks in Sport scheduling, J-C Regin, Dimacs 98
  - GAC-Schema with computation on the fly: C. Bessiere and J-C Regin, CP'99
  - Clique max in CP, J-C Regin, CP'03
  - Network Design, C LePape, L Perron, J-C Regin, P. Shaw, CP'02
- Over-constrained problems:
  - Meta constraints on violations, T. Petit, J-C Regin, C Bessiere, ICTAI 2000
  - Original constraint based approach for solving over-constrained problems, J-C Regin, T. Petit, C. Bessiere, J-F Puget, CP'00 and CP'01
  - Soft global constraints: T. Petit, J-C Regin, C. Bessiere, CP'01