Global Constraints

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

Plan

- General Principles of Constraint Programming
- Global Constraints: advantages
- How to write a filtering algorithm?
- Examples: sports scheduling and car sequencing
- Over-constrained problems
- Discussion: quality of a FA, incrementality, closure, incomplete algorithms, power of a FA
- Conclusion

Plan

- **General Principles of Constraint Programming**
- Global Constraints: advantages
- How to write a filtering algorithm?
- Examples: sports scheduling and car sequencing
- Over-constrained problems
- Discussion: quality of a FA, incrementality, closure, incomplete algorithms, power of a FA
- Conclusion

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

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)

Alldiff and GCC Constraints

- **Alldiff(X): the variables of X must be pairwise different (i.e. $\forall$ x, y $\in$ X: x $\neq$ y)**
- **GCC**(X,{l_i},{u_i}): the number of times each value v_i can be taken must be in a given interval [l_i, u_i]
- Example: D(x1)={a,b,c,d}, D(x2)={a,b,c,d}, D(x3)={b,c},D(x4)={c,d}. Values b and c must be taken at most 2, values a and d must be taken at least 1.

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

- A global constraint is equal to a conjunction of constraints
- Example: alldiff and Gcc constraints
- $G=\wedge\{C_1,C_2,..,C_k\}$
  The set of tuples of G is equal to the set of solutions of the problem: $(\cup_i X(C_i), DX(G),\{C_1,C_2,..,C_k\})$

Plan

- General Principles of Constraint Programming
- **Global Constraints: advantages**
- How to write a filtering algorithm?
- Examples: sports scheduling and car sequencing
- Over-constrained problems
- Discussion: quality of a FA, incrementality, closure, incomplete algorithms, power of a FA
- Conclusion

Global Constraints: advantages

- **Expressiveness: It is more convenient to define one constraint corresponding to a set of constraints than to define independently each constraint**
- **Better understanding of the problem structure:** some part of the structure is immediatly identified
- **Powerful filtering algorithms:** the set of constraint can be taken into account as a whole

Global constraint: expressiveness

- Example Rostering Problem

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

Mrs. Blue

M. Red

M. Yellow

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

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

Rostering

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
D							
E							
N							

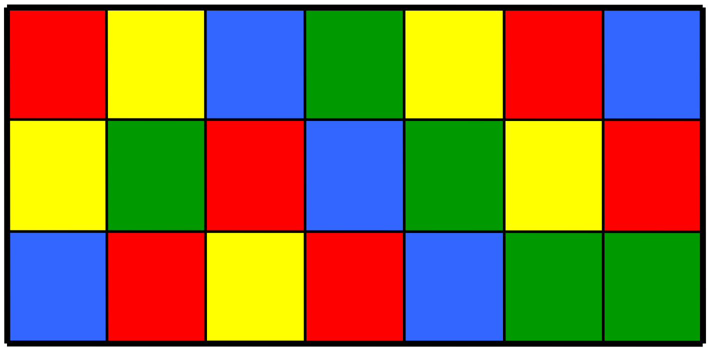
M. Green
M. Red
Mrs. Blue
M. Yellow

```
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]
```

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])

Global constraint: underlined structure

- A global cardinality constraint is equivalent to a set of atmost/atleast constraint.
- From the simultaneous presence of these constraints new deductions can be made (detailed in this talk)

Global constraint: powerful filtering algorithms

- 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: ?????????????

Some Global Constraints

- Cumulative, diff-n, cycle, sort, alldiff and permutation, symmetric alldiff, global cardinality, global cardinality with costs, sum and scalar product of alldiff variables, sequence, stretch, minimum global distance, k-diff, number of distinct values, lexicographic ordering, regular.

Cumulative constraint

- Scheduling problems:
- Activities having a start time (S var), a duration (D var), a consumption (H var)
- At each time the summation of the consumption must be less than a var u.
- Non preemptive scheduling (interruption is forbidden)

Sort

- Two types of variables: X var and Y var
- The Y var represents the X var when there are sorted.
- Example: x1=[0,5], x2=[0,5], x3=[0,5]
- We want to have a strict ordering: y1=[0,3], y2=[1,4], y3=[2,5]

Symmetric alldiff

- Goal: group by pair
- Alldiff + If i is assigned to j then j is assigned to I
- Vars= entities and Values =entities
- 3 entitites: a,b,c: 3 vars xa,xb,xc and 3 values a,b,c
- If xa=c then xc=a
- No solution here (the alldiff does not find this result).

Nvalue

- Nvalue(X,k): the number of distinct values assigned to X must be equal to k.
- (x1=a,x2=b,x3=c,x4=b,x5=a,x6=a): k=3 (a,b,c)
- NP-Complete constraint equivalent to set-cover problem.

Stretch

- Run length coding
- Consecutives values
- Defines by size of group of values (with min and max)
- Idea: var are ordered and if xi is blue then xi belongs to a group of k consecutive var taken blue as value.

Regular

- Defined by automata
- Kind of table constraints whose list of tuples is defined by an automata (instead of being explicitly given).

Plan

- General Principles of Constraint Programming
- Global Constraints: advantages
- **How to write a filtering algorithm?**
- Examples: sports scheduling and car sequencing
- Over-constrained problems
- Discussion: quality of a FA, incrementality, closure, incomplete algorithms, power of a FA
- Conclusion

Filtering Algorithm

- based on constraints addition (from the simultaneous presence of constraints, new constraints are added)
- general (GAC-Schema)
- ad-hoc (integration of OR algorithm)

Constraints addition

- 5 variables: X={x1,x2,x3,x4,x5}
- domains: [0..4]
- constraints: atleast(X, 1,1) , atleast(X,1,2) atleast(X,1,3), atleast(X, 1,4)
- What will happen ?

Constraints addition

- 5 variables: X={x1,x2,x3,x4,x5}
- domains: [0..4]
- constraints: atleast(X, 1,1) , atleast(X,1,2) atleast(X,1,3), atleast(X, 1,4)
- atleast(X,1,val): local view: while val belongs to 1+1=2 domains of variable, nothing can be deduced

Constraints addition

- 5 variables: X={x1,x2,x3,x4,x5}
- domains: [0..4]
- constraints: atleast(X, 1,1) , atleast(X,1,2) atleast(X,1,3), atleast(X, 1,4)
- atleast(X,1,val): local view: while val belongs to 1+1=2 domains of variable, nothing can be deduced
- x1=0, x2=0, x3=0 is ok, any combination of 3 variables containing 2 times val 0 is ok. This is stupid

Constraints addition

- 5 variables: X={x1,x2,x3,x4,x5}
- domains: [0..4]
- constraints: atleast(X, 1,1) , atleast(X,1,2)
  atleast(X,1,3), atleast(X, 1,4)
- From the simultaneous presence of constraints we can deduce other constraints:
  if (4 values must be taken atleast 1) then the other values can be taken at most n-4=5-4=1
  New constraint: atmost(X,1,0)

Constraints addition

- Done for the global cardinality constraint (the unconstrained values becomes constrained)
- Therefore, done for the alldiff constraint and permutation (each value has to be taken exactly once)

Constraints addition

- Another example [Roy and Pachet CP’99]:
  Union of set variables:
  E = A union B
  Good implementation:

Constraints addition

- Another example [Roy and Pachet CP'99]:
  Union of set variables:
  E = A union B
  Good implementation: **think to the intersection**
  I = A intersect B, and
  card(E) = card(A) + card(B) - card(I)

Constraints addition

- There is no new filtering algorithm
- Only implicit constraints are added
- The previous problem is solved!
- Easy and really interesting: a kind of presolve.

Filtering Algorithm

- *based on constraints addition (from the simultaneous presence of constraints, new constraints are added)*
- **general (GAC-Schema)**
- ad-hoc (integration of OR algorithm)

GAC-Schema

- A generic framework for achieving AC for any kind of constraint (can be non binary).
  Bessiere and Regin, IJCAI'97, CP'99
- You just have to say how to compute a solution.
- **Manages the incrementality for you** (notion of support).

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

Example

- X(C)={x1,x2,x3} D(xi)={a,b}
- T(C)={(a,a,a),(a,b,b),(b,b,a),(b,b,b)}

Example

- X(C)={x1,x2,x3} D(xi)={a,b}
- T(C)={(a,a,a),(a,b,b),(b,b,a),(b,b,b)}
- Support for (x1,a): (a,a,a) is computed and (a,a,a) is added to S(x2,a) and S(x3,a), (x1,a) in (a,a,a) is marked as supported.

Example

- X(C)={x1,x2,x3} D(xi)={a,b}
- T(C)={(a,a,a),(a,b,b),(b,b,a),(b,b,b)}
- Support for (x1,a): (a,a,a) is computed and (a,a,a) is added to S(x2,a) and S(x3,a), (x1,a) in (a,a,a) is marked as supported.
- Support for (x2,a): (a,a,a) is in S(x2,a) it is valid, therefore it is a support. (Multidirectionnality). **No need to compute a solution**

Example

- X(C)={x1,x2,x3} D(xi)={a,b}
- T(C)={(a,a,a),(a,b,b),(b,b,a),(b,b,b)}
- Support for (x1,a): (a,a,a) is computed and (a,a,a) is added to S(x2,a) and S(x3,a), (x1,a) in (a,a,a) is marked as supported.
- Value a is removed from x2, then all the tuple in S(x2,a) are no longer valid: (a,a,a) for instance. The validity of the values supported by this tuple must be reconsidered.

Example

- X(C)={x1,x2,x3} D(xi)={a,b}
- T(C)={(a,a,a),(a,b,b),(b,b,a),(b,b,b)}
- Support for (x1,a): (a,a,a) is computed and (a,a,a) is added to S(x2,a) and S(x3,a), (x1,a) in (a,a,a) is marked as supported.
- Support for (x1,b): (b,b,a) is computed, and update ...

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).**

GAC-Schema: complexity

- After 1 modification:
  consistency in O(CC))
  arc consistency in O(ndCC)
- After k modifications
  consistency in O(CC)
  arc consistency in O(ndCC)

Table Constraint: An example

- 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?

Table Constraint: results

	#bk	time
standard model	1,361,709	430
Table Constraint	12,659	9.7

Filtering Algorithm

- *based on constraints addition (from the simultaneous presence of constraints, new constraints are added)*
- *general (GAC-Schema)*
- **ad-hoc (integration of OR algorithm but not only)**

Structure of a constraint

- Speed-up the search for a support (solution which contain a value (x,a))

Structure of a constraint

- Speed-up the search for a support (solution which contain a value (x,a)):
  x < y, D(x)=[0..10000], D(y)=[0..10000]
  support for (x,9000): immediate any value greater than 9000 in D(y)

Structure of a constraint

- Design of ad-hoc filtering algorithm:
  $x < y$:
  (a) $\max(x) = \max(y) - 1$
  (b) $\min(y) = \min(x) + 1$

Structure of a constraint

- Design of ad-hoc filtering algorithm:
  $x < y$:
  (a) $\max(x) = \max(y) - 1$
  (b) $\min(y) = \min(x) + 1$
- Triggering of the filtering algorithm:
  no possible pruning of D(x) while max(y) is not modified
  no possible pruning of D(y) while min(x) is not modified

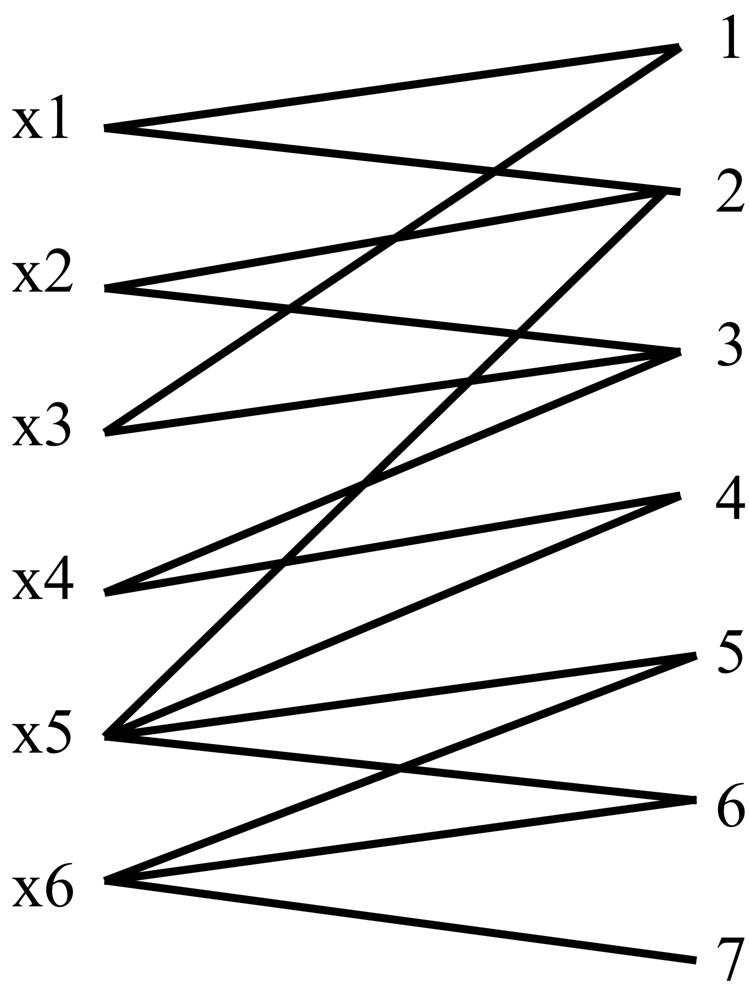
Structure of a constraint

- Speed-up the search for a support (solution which contain a value (x,a))
- Design of specific algorithm
- Incrementality

Alldiff constraint

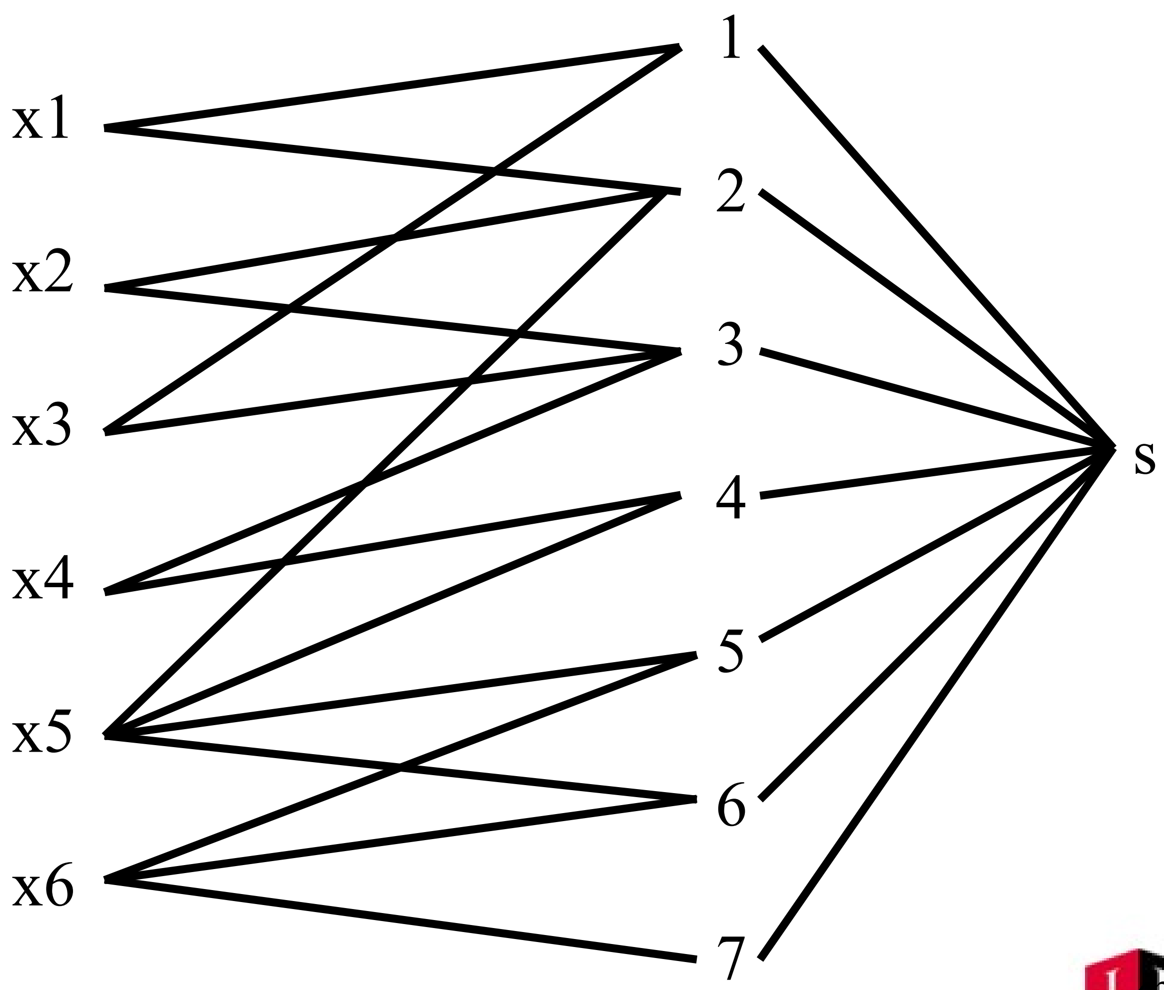
The value graph:

D(x1)={1,2}
D(x2)={2,3}
D(x3)={1,3}
D(x4)={3,4}
D(x5)={2,4,5,6}
D(x6)={5,6,7}

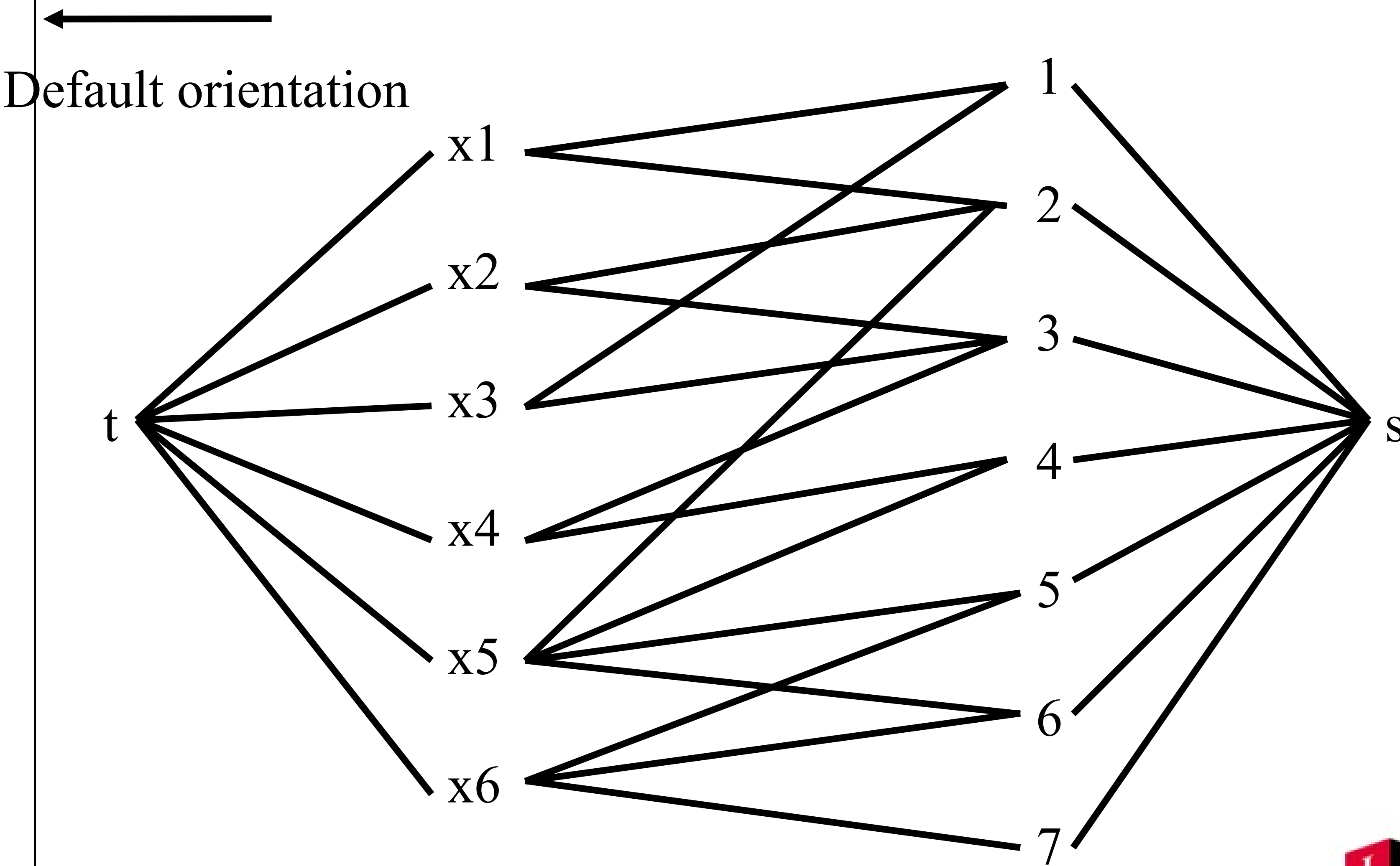


Alldiff constraint

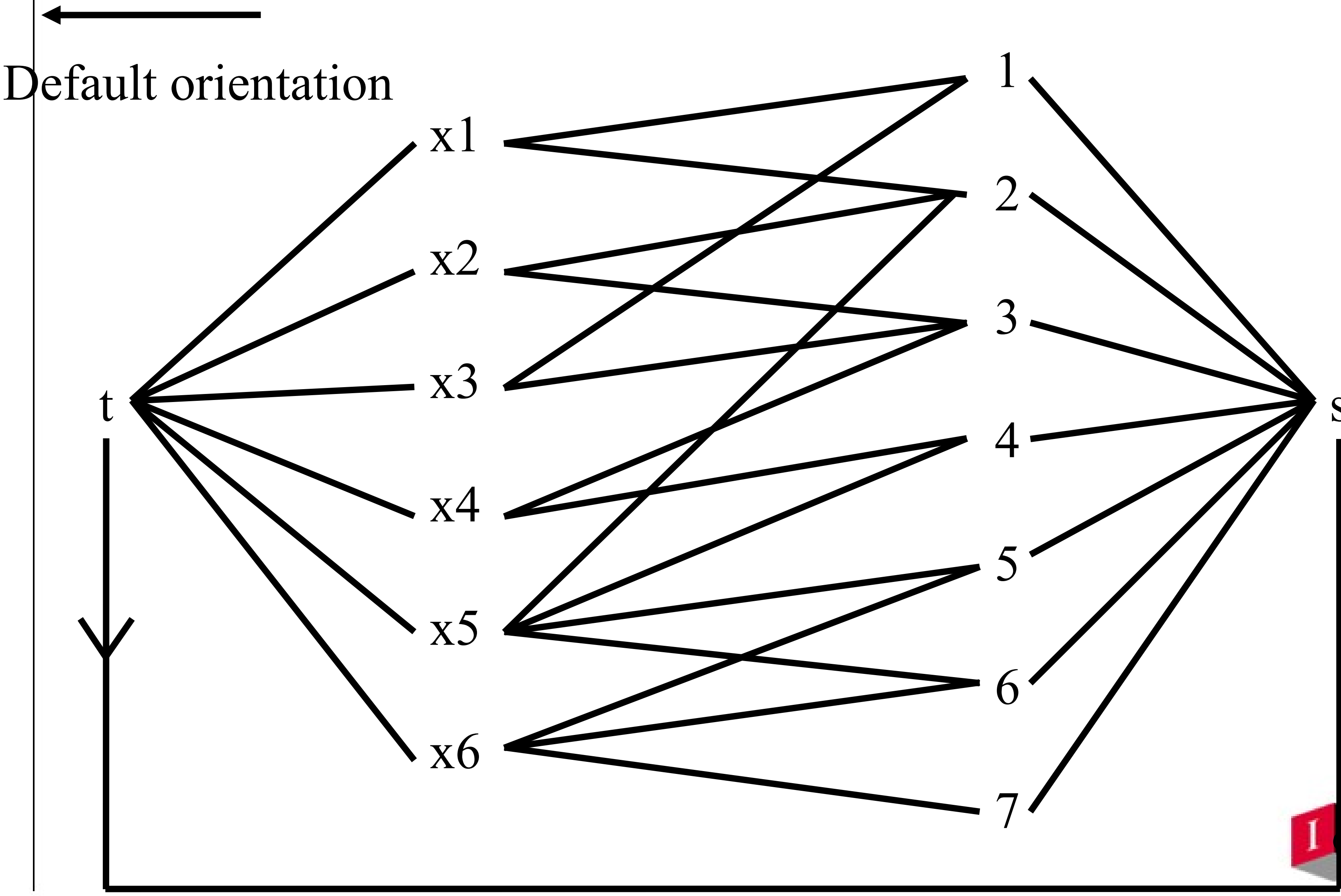
Default orientation



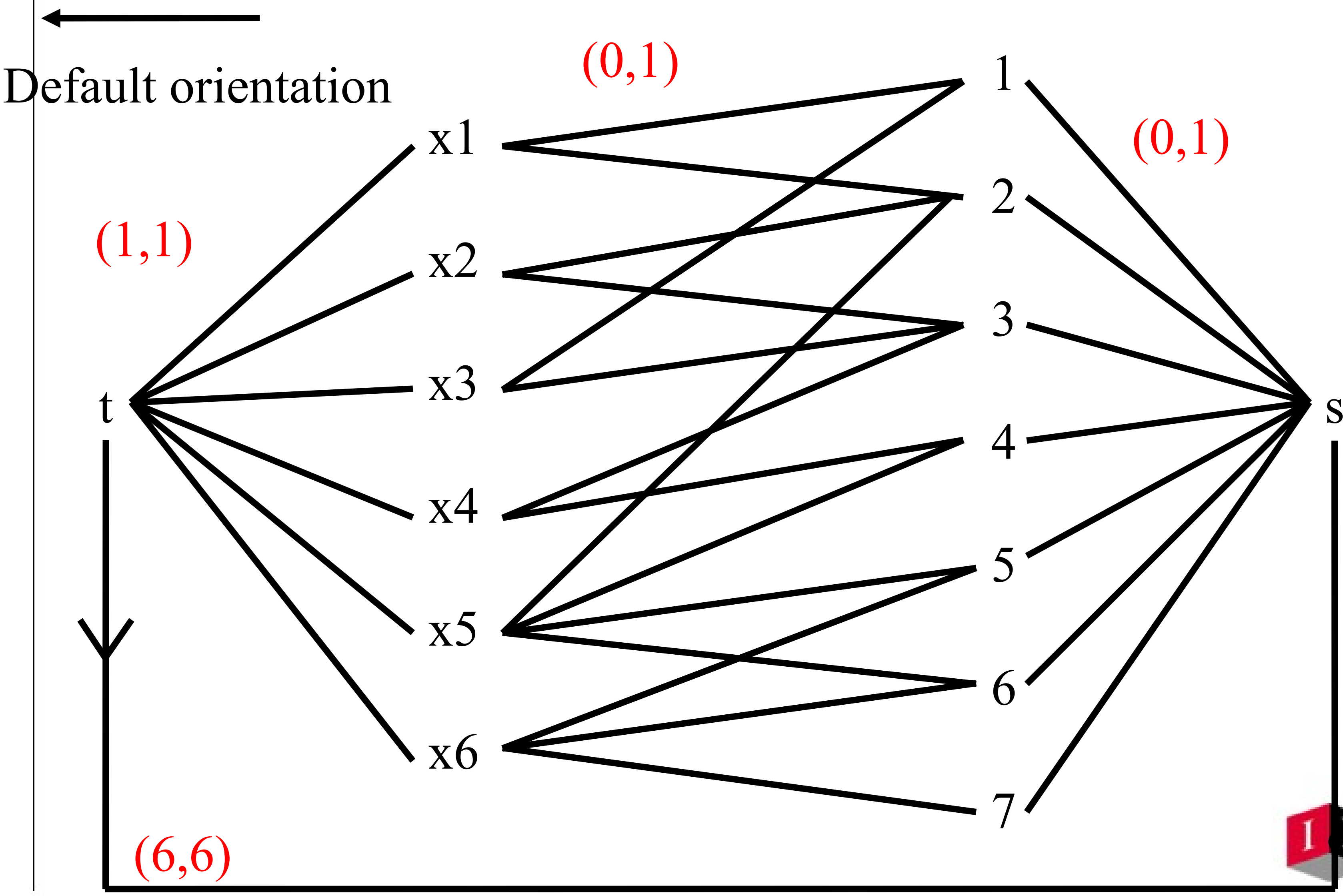
Alldiff constraint



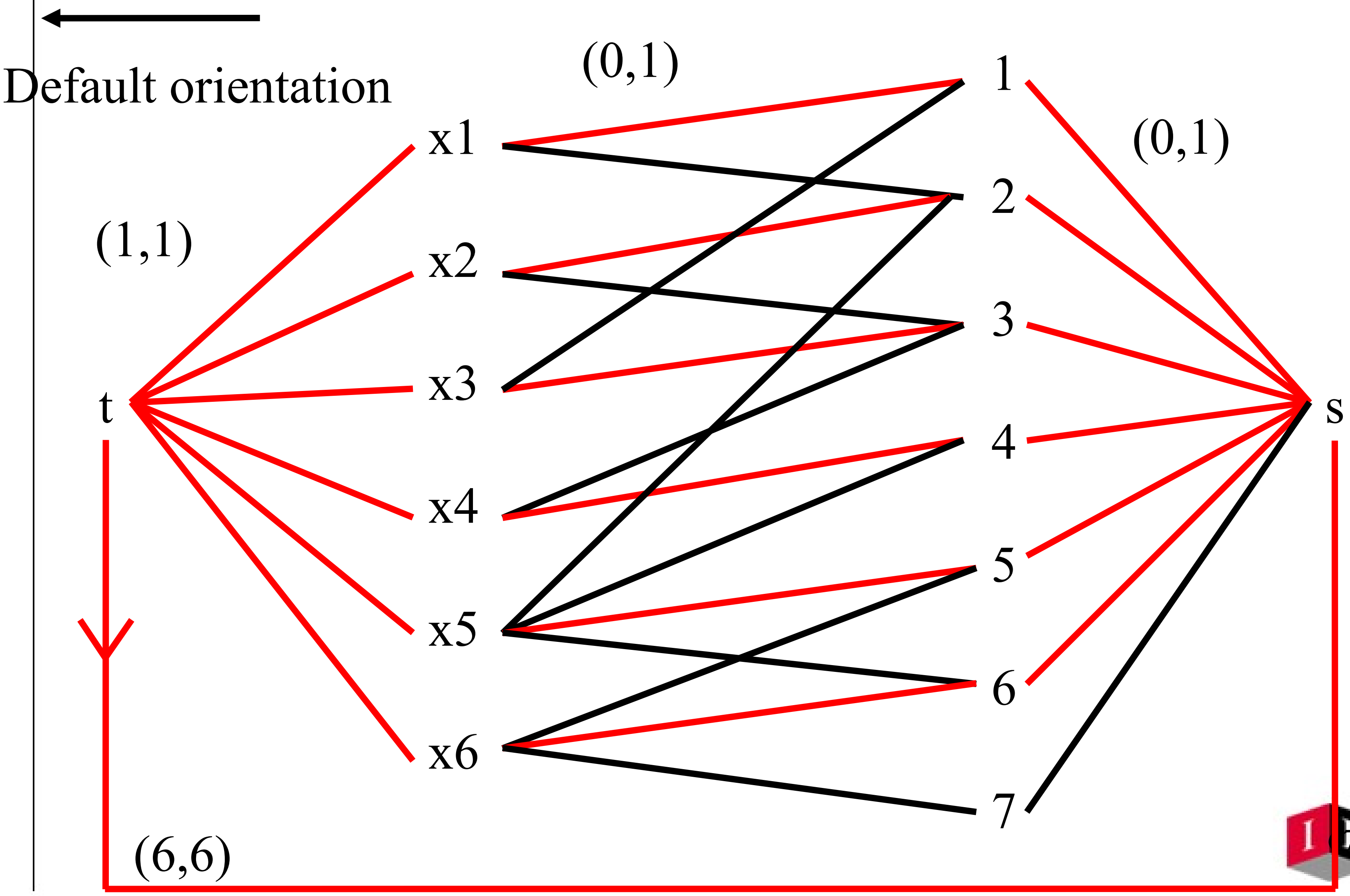
Alldiff constraint



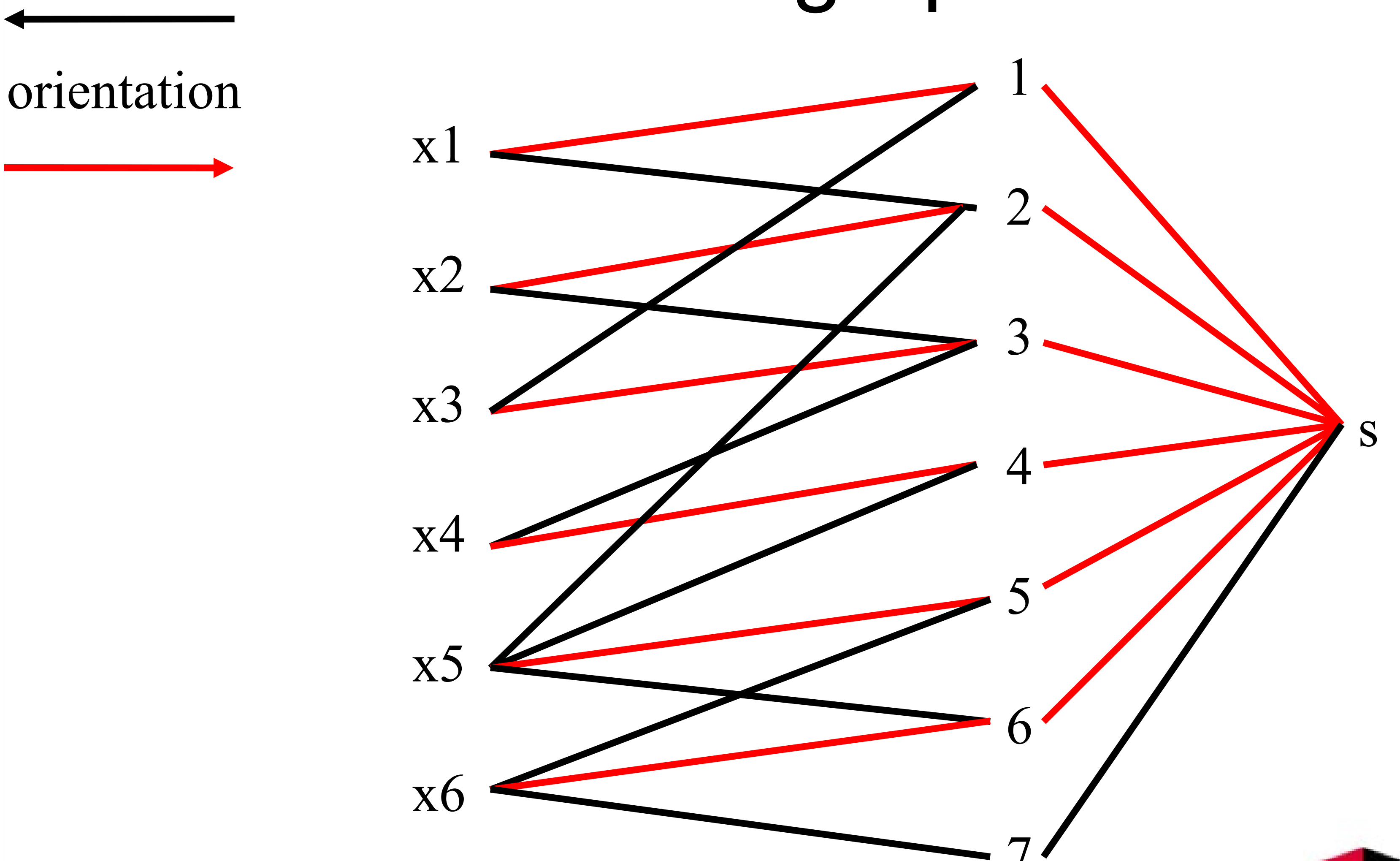
Value network



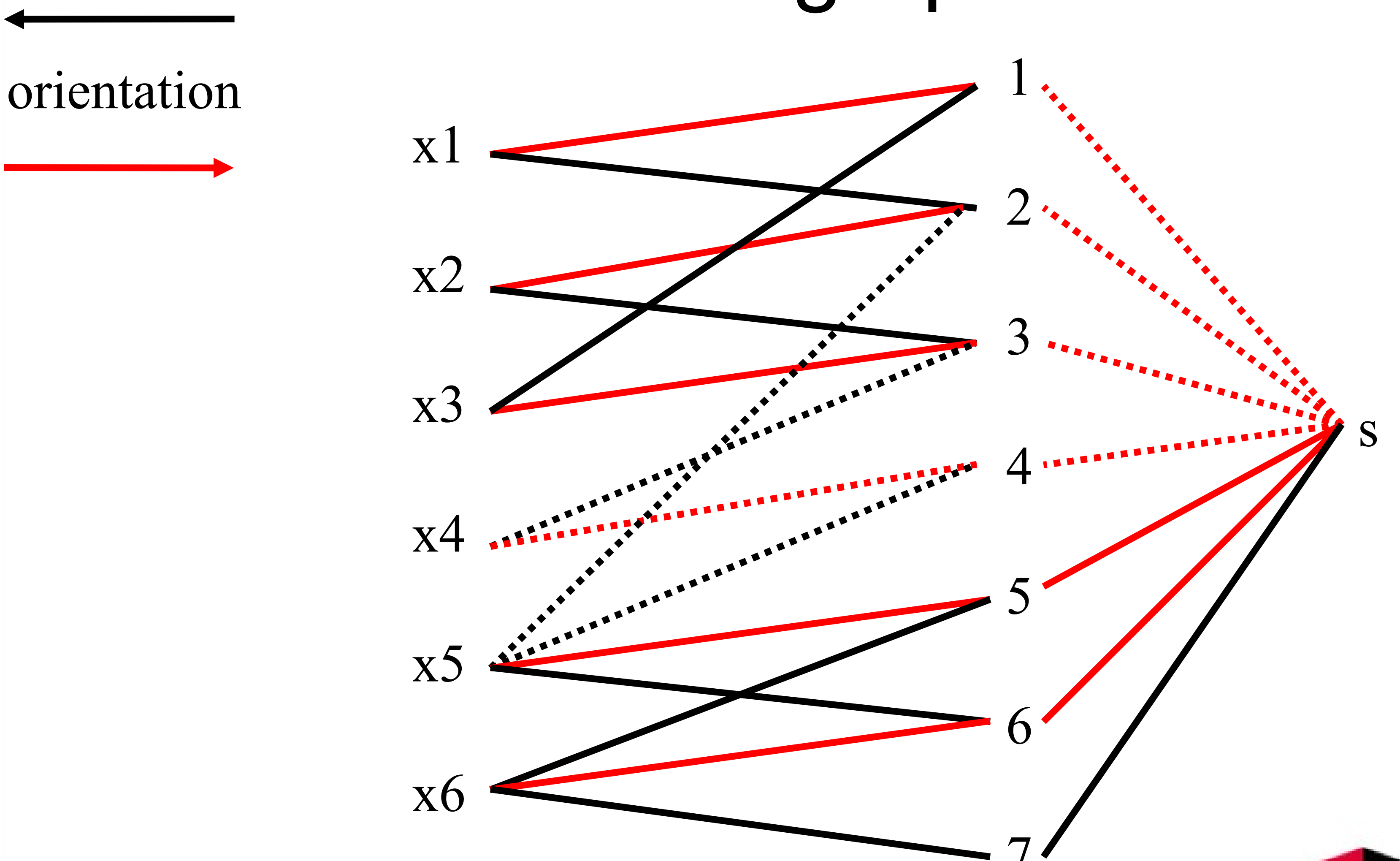
A feasible flow



Residual graph



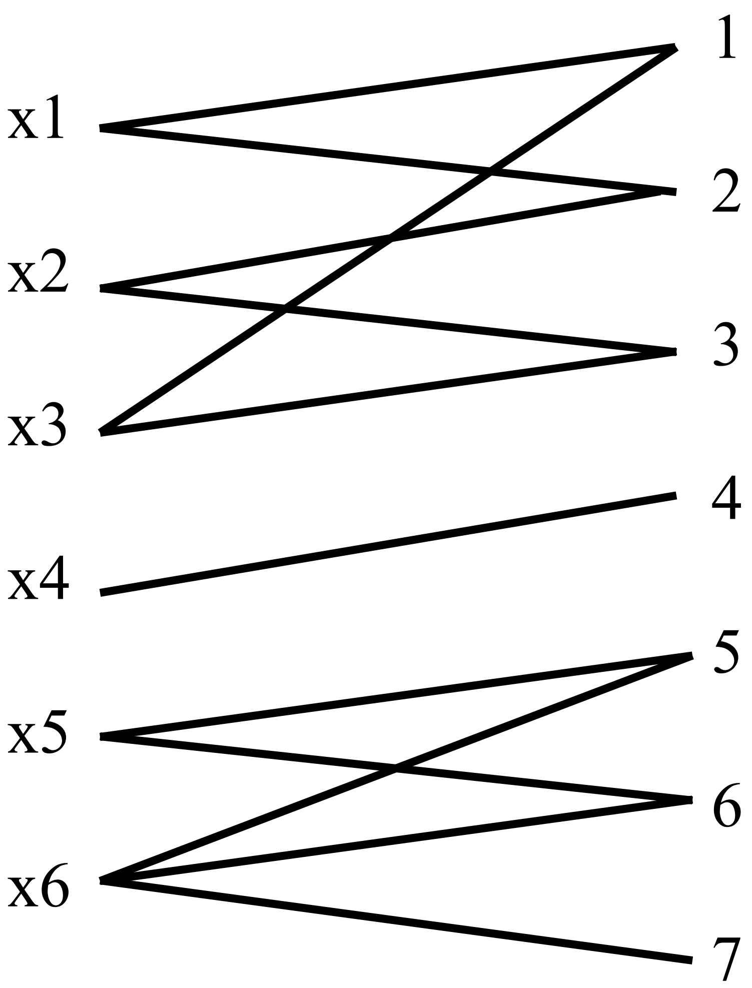
Residual graph



Arc consistency

The value graph:

D(x1)={1,2}
D(x2)={2,3}
D(x3)={1,3}
D(x4)={4}
D(x5)={5,6}
D(x6)={5,6,7}



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: ?????????????

Alldiff constraint

- Compute a feasible flow
- Compute the strongly connected components
- Remove every arc of flow value 0 for which the ends belong to two different components
- Linear algorithm achieving arc consistency
- Idem for global cardinality constraints
- work well due to (0,1) arcs

Alldiff constraint: complexity

- After 1 modification:
  - consistency computed in $O(nd)$
  - arc consistency computed in $O(nd)$
- After k modifications:
  - consistency in $O(nd \sqrt{k})$
  - arc consistency in $O(nd \sqrt{k} + nd)$

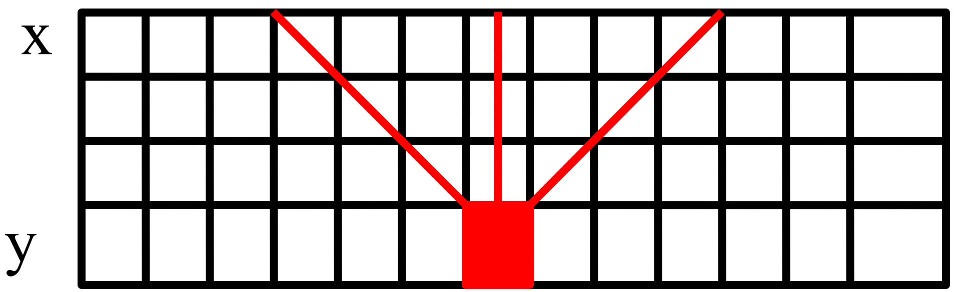
Alldiff constraint

- **Relations between GAC, AC, etc…**:

  Arc Consistency for the global constraints corresponds to the arc consistency of a CN with an exponential number of constraints:
  - for k=1..n: for every group G of k variables:
  we must have :
  $|D(G)| \geq k$
  and if $|D(G)|=k$ then $D(X) \leftarrow D(X) - D(G)$

Ad-hoc algorithm: N-queens problems

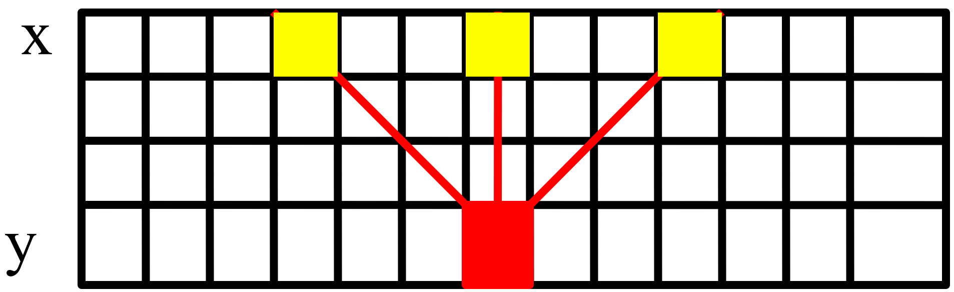
- variables: a var = possible columns for a row
  Rule: [Regin]
  If the domain of a var contains more than 3 values, this var cannot cause any deletion



3 directions for every value of y
if x contains 4 values: no problem

Ad-hoc algorithm: N-queens problems

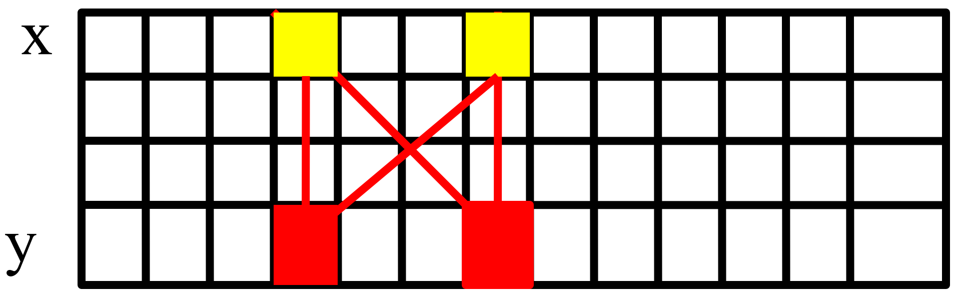
- variables: a var = possible columns for a row
  Rule: [Regin]
  If the domain of a var contains exactly 3 values, this var can cause only specific deletions



The red value of y is deleted
only if x contains yellows values

Ad-hoc algorithm: N-queens problems

- variables: a var = possible columns for a row
  Rule: [Regin]
  If the domain of a var contains exactly 2 values, this var can cause only specific deletions



The red values of y are deleted
only if x contains yellows values

Plan

- General Principles of Constraint Programming
- Global Constraints: advantages
- How to write a filtering algorithm?
- **Examples: sports scheduling and car sequencing**
- Over-constrained problems
- Discussion: quality of a FA, incrementality, closure, incomplete algorithms, power of a FA
- Conclusion

Examples

- Sports scheduling
- Car sequencing

The problem

- n teams and n-1 weeks and n/2 periods
- every two teams play each other exactly once
- every team plays one game in each week
- no team plays more than twice in the same period

	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7
Period 1	0 vs 1	0 vs 2	4 vs 7	3 vs 6	3 vs 7	1 vs 5	2 vs 4
Period 2	2 vs 3	1 vs 7	0 vs 3	5 vs 7	1 vs 4	0 vs 6	5 vs 6
Period 3	4 vs 5	3 vs 5	1 vs 6	0 vs 4	2 vs 6	2 vs 7	0 vs 7
Period 4	6 vs 7	4 vs 6	2 vs 5	1 vs 2	0 vs 5	3 vs 4	1 vs 3

The problem

- n teams and n-1 weeks and n/2 periods
- every two teams play each other exactly once
- every team plays one game in each week
- no team plays more than twice in the same period

	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7
Period 1	0 vs 1	0 vs 2	4 vs 7	3 vs 6	3 vs 7	1 vs 5	2 vs 4
Period 2	2 vs 3	1 vs 7	0 vs 3	5 vs 7	1 vs 4	0 vs 6	5 vs 6
Period 3	4 vs 5	3 vs 5	1 vs 6	0 vs 4	2 vs 6	2 vs 7	0 vs 7
Period 4	6 vs 7	4 vs 6	2 vs 5	1 vs 2	0 vs 5	3 vs 4	1 vs 3

- Problem 10teams of the MIPLIB
  (n=10 and the objective function is dummy)
- MIP is not able to find a solution for n=14
- CP finds a solution for n=10 in 0.06s, n=14 in 0.2, n=40 in 6[illegible]

The problem

- n teams and n-1 weeks and n/2 periods
- every two teams play each other exactly once
- every team plays one game in each week
- no team plays more than twice in the same period

	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7
Period 1	0 vs 1	0 vs 2	4 vs 7	3 vs 6	3 vs 7	1 vs 5	2 vs 4
Period 2	2 vs 3	1 vs 7	0 vs 3	5 vs 7	1 vs 4	0 vs 6	5 vs 6
Period 3	4 vs 5	3 vs 5	1 vs 6	0 vs 4	2 vs 6	2 vs 7	0 vs 7
Period 4	6 vs 7	4 vs 6	2 vs 5	1 vs 2	0 vs 5	3 vs 4	1 vs 3

For 40 teams: 800 variables with 39 possible values for each variable.

CP model: variables

For each slot: 2 variables represent the teams and 1 variable represents the match are defined

	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7
Period 1	0 vs 1	0 vs 2	4 vs 7	3 vs 6	3 vs 7	1 vs 5	2 vs 4
Period 2	2 vs 3	1 vs 7	0 vs 3	5 vs 7	1 vs 4	0 vs 6	5 vs 6
Period 3	4 vs 5	3 vs 5	1 vs 6	0 vs 4	2 vs 6	2 vs 7	0 vs 7
Period 4	6 vs 7	4 vs 6	2 vs 5	1 vs 2	0 vs 5	3 vs 4	1 vs 3

1 vs 6

M33 variable (M33=12)

T33a variable (T33a=6)

T33h variable (T33h=1)

Mij=1 <=> 0 vs 1 or 1 vs 0

Mij=12 <=> 1 vs 6 or 6 vs1

CP model: T variables

	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7
Period 1	T11h vs T11a	T12h vs T12a	T13h vs T13a	T14h vs T14a	T15h vs T15a	T16h vs T16a	T17h vs T17a
Period 2	T21h vs T21a	T22h vs T22a	T23h vs T23a	T24h vs T24a	T25h vs T25a	T26h vs T26a	T27h vs T27a
Period 3	T31h vs T31a	T32h vs T32a	T33h vs T33a	T34h vs T34a	T35h vs T35a	T36h vs T36a	T37h vs T37a
Period 4	T41h vs T41a	T42h vs T42a	T43h vs T43a	T44h vs T44a	T45h vs T45a	T46h vs T46a	T47h vs T47a

D(Tija)=[1,n-1]

D(Tijh)=[0,n-2]

Tijh < Tija

CP model: M variables

	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7
Period 1	M11	M12	M13	M14	M15	M16	M17
Period 2	M21	M22	M23	M24	M25	M26	M27
Period 3	M31	M32	M33	M34	M35	M36	M37
Period 4	M41	M42	M43	M44	M45	M46	M47

$$D(M_{ij})=[1,n(n-1)/2]$$

CP model: constraints

- n teams and n-1 weeks and n/2 periods
- every two teams play each other exactly once
- every team plays one game in each week
- no team plays more than twice in the same period

	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7
Period 1	M11	M12	M13	M14	M15	M16	M17
Period 2	M21	M22	M23	M24	M25	M26	M27
Period 3	M31	M32	M33	M34	M35	M36	M37
Period 4	M41	M42	M43	M44	M45	M46	M47

CP model: constraints

- n teams and n-1 weeks and n/2 periods
- every two teams play each other exactly once
- every team plays one game in each week
- no team plays more than twice in the same period

	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7
Period 1	M11	M12	M13	M14	M15	M16	M17
Period 2	M21	M22	M23	M24	M25	M26	M27
Period 3	M31	M32	M33	M34	M35	M36	M37
Period 4	M41	M42	M43	M44	M45	M46	M47

Alldiff constraints defined on M variables

CP model: constraints

- n teams and n-1 weeks and n/2 periods
- every two teams play each other exactly once
- every team plays one game in each week
- no team plays more than twice in the same period

	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7
Period 1	T11h vs T11a	T12h vs T12a	T13h vs T13a	T14h vs T14a	T15h vs T15a	T16h vs T16a	T17h vs T17a
Period 2	T21h vs T21a	T22h vs T22a	T23h vs T23a	T24h vs T24a	T25h vs T25a	T26h vs T26a	T27h vs T27a
Period 3	T31h vs T31a	T32h vs T32a	T33h vs T33a	T34h vs T34a	T35h vs T35a	T36h vs T36a	T37h vs T37a
Period 4	T41h vs T41a	T42h vs T42a	T43h vs T43a	T44h vs T44a	T45h vs T45a	T46h vs T46a	T47h vs T47a

CP model: constraints

- n teams and n-1 weeks and n/2 periods
- every two teams play each other exactly once
- every team plays one game in each week
- no team plays more than twice in the same period

	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7
Period 1	T11h vs T11a	T12h vs T12a	T13h vs T13a	T14h vs T14a	T15h vs T15a	T16h vs T16a	T17h vs T17a
Period 2	T21h vs T21a	T22h vs T22a	T23h vs T23a	T24h vs T24a	T25h vs T25a	T26h vs T26a	T27h vs T27a
Period 3	T31h vs T31a	T32h vs T32a	T33h vs T33a	T34h vs T34a	T35h vs T35a	T36h vs T36a	T37h vs T37a
Period 4	T41h vs T41a	T42h vs T42a	T43h vs T43a	T44h vs T44a	T45h vs T45a	T46h vs T46a	T47h vs T47a

For each week w:
Alldiff constraint defined
on {Tpwh, p=1..4} U {Tpwa, p=1..4}

CP model: constraints

- n teams and n-1 weeks and n/2 periods
- every two teams play each other exactly once
- every team plays one game in each week
- no team plays more than twice in the same period

	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7
Period 1	T11h vs T11a	T12h vs T12a	T13h vs T13a	T14h vs T14a	T15h vs T15a	T16h vs T16a	T17h vs T17a
Period 2	T21h vs T21a	T22h vs T22a	T23h vs T23a	T24h vs T24a	T25h vs T25a	T26h vs T26a	T27h vs T27a
Period 3	T31h vs T31a	T32h vs T32a	T33h vs T33a	T34h vs T34a	T35h vs T35a	T36h vs T36a	T37h vs T37a
Period 4	T41h vs T41a	T42h vs T42a	T43h vs T43a	T44h vs T44a	T45h vs T45a	T46h vs T46a	T47h vs T47a

CP model: constraints

- n teams and n-1 weeks and n/2 periods
- every two teams play each other exactly once
- every team plays one game in each week
- no team plays more than twice in the same period

	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7
Period 1	T11h vs T11a	T12h vs T12a	T13h vs T13a	T14h vs T14a	T15h vs T15a	T16h vs T16a	T17h vs T17a
Period 2	T21h vs T21a	T22h vs T22a	T23h vs T23a	T24h vs T24a	T25h vs T25a	T26h vs T26a	T27h vs T27a
Period 3	T31h vs T31a	T32h vs T32a	T33h vs T33a	T34h vs T34a	T35h vs T35a	T36h vs T36a	T37h vs T37a
Period 4	T41h vs T41a	T42h vs T42a	T43h vs T43a	T44h vs T44a	T45h vs T45a	T46h vs T46a	T47h vs T47a

For each period p:
Global cardinality constraint defined on
{Tpwh, w=1..7} U {Tpwa, w=1..7}
every team t is taken at most 2

CP model: constraints

- For each slot the two T variables and the M variable must be linked together; example:
  M12 = game T12h vs T12a

CP model: constraints

- For each slot the two T variables and the M variable must be linked together; example:
  M12 = game T12h vs T12a
- For each slot we add Cij a ternary constraint defined on the two T variables and the M variable; example:
  C12 defined on {T12h,T12a,M12}

CP model: constraints

- For each slot the two T variables and the M variable must be linked together; example:
  M12 = game T12h vs T12a
- For each slot we add Cij a ternary constraint defined on the two T variables and the M variable; example:
  C12 defined on {T12h,T12a,M12}
- Cij are defined by the list of allowed tuples:
  for n=4: {(0,1,1),(0,2,2),(0,3,3),(1,2,4),(1,3,5),(2,3,6)}
  (1,2,4) means game 1 vs 2 is the game number 4

CP model: constraints

- For each slot the two T variables and the M variable must be linked together; example:
  M12 = game T12h vs T12a
- For each slot we add Cij a ternary constraint defined on the two T variables and the M variable; example:
  C12 defined on {T12h,T12a,M12}
- Cij are defined by the list of allowed tuples:
  for n=4: {(0,1,1),(0,2,2),(0,3,3),(1,2,4),(1,3,5),(2,3,6)}
  (1,2,4) means game 1 vs 2 is the game number 4
- All these constraints have the same list of allowed tuples
- Efficient arc consistency algorithm for this kind of constraint is known

First model

Introduction of a dummy column

	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Dummy
Period 1	0 vs 1	0 vs 2	4 vs 7	3 vs 6	3 vs 7	1 vs 5	2 vs 4	. vs .
Period 2	2 vs 3	1 vs 7	0 vs 3	5 vs 7	1 vs 4	0 vs 6	5 vs 6	. vs .
Period 3	4 vs 5	3 vs 5	1 vs 6	0 vs 4	2 vs 6	2 vs 7	0 vs 7	. vs .
Period 4	6 vs 7	4 vs 6	2 vs 5	1 vs 2	0 vs 5	3 vs 4	1 vs 3	. vs .

First model

Introduction of a dummy column

	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Dummy
Period 1	0 vs 1	0 vs 2	4 vs 7	3 vs 6	3 vs 7	1 vs 5	2 vs 4	. vs .
Period 2	2 vs 3	1 vs 7	0 vs 3	5 vs 7	1 vs 4	0 vs 6	5 vs 6	. vs .
Period 3	4 vs 5	3 vs 5	1 vs 6	0 vs 4	2 vs 6	2 vs 7	0 vs 7	. vs .
Period 4	6 vs 7	4 vs 6	2 vs 5	1 vs 2	0 vs 5	3 vs 4	1 vs 3	. vs .

We can prove that:

- each team occurs exactly twice for each period

First model

Introduction of a dummy column

	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Dummy
Period 1	0 vs 1	0 vs 2	4 vs 7	3 vs 6	3 vs 7	1 vs 5	2 vs 4	5 vs 6
Period 2	2 vs 3	1 vs 7	0 vs 3	5 vs 7	1 vs 4	0 vs 6	5 vs 6	. vs .
Period 3	4 vs 5	3 vs 5	1 vs 6	0 vs 4	2 vs 6	2 vs 7	0 vs 7	. vs .
Period 4	6 vs 7	4 vs 6	2 vs 5	1 vs 2	0 vs 5	3 vs 4	1 vs 3	. vs .

We can prove that:

- each team occurs exactly twice for each period

First model

Introduction of a dummy column

	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Dummy
Period 1	0 vs 1	0 vs 2	4 vs 7	3 vs 6	3 vs 7	1 vs 5	2 vs 4	5 vs 6
Period 2	2 vs 3	1 vs 7	0 vs 3	5 vs 7	1 vs 4	0 vs 6	5 vs 6	2 vs 4
Period 3	4 vs 5	3 vs 5	1 vs 6	0 vs 4	2 vs 6	2 vs 7	0 vs 7	1 vs 3
Period 4	6 vs 7	4 vs 6	2 vs 5	1 vs 2	0 vs 5	3 vs 4	1 vs 3	0 vs 7

We can prove that:

- each team occurs exactly twice for each period

First model

Introduction of a dummy column

	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Dummy
Period 1	0 vs 1	0 vs 2	4 vs 7	3 vs 6	3 vs 7	1 vs 5	2 vs 4	5 vs 6
Period 2	2 vs 3	1 vs 7	0 vs 3	5 vs 7	1 vs 4	0 vs 6	5 vs 6	2 vs 4
Period 3	4 vs 5	3 vs 5	1 vs 6	0 vs 4	2 vs 6	2 vs 7	0 vs 7	1 vs 3
Period 4	6 vs 7	4 vs 6	2 vs 5	1 vs 2	0 vs 5	3 vs 4	1 vs 3	0 vs 7

We can prove that:

- each team occurs exactly twice for each period
- each team occurs exactly once in the dummy column

First model

Introduction of a dummy column

	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Dummy
Period 1	0 vs 1	0 vs 2	4 vs 7	3 vs 6	3 vs 7	1 vs 5	2 vs 4	5 vs 6
Period 2	2 vs 3	1 vs 7	0 vs 3	5 vs 7	1 vs 4	0 vs 6	5 vs 6	2 vs 4
Period 3	4 vs 5	3 vs 5	1 vs 6	0 vs 4	2 vs 6	2 vs 7	0 vs 7	1 vs 3
Period 4	6 vs 7	4 vs 6	2 vs 5	1 vs 2	0 vs 5	3 vs 4	1 vs 3	0 vs 7

- The problem is exactly the same
- The solver is helped by such constraint. It can deduce some inconsistencies more quickly

First model: strategies

- Break symmetries: 0 vs w appears in week w

First model: strategies

- Break symmetries: 0 vs w appears in week w
- Teams are instantiated:
  - the most instantiated team is chosen
  - the slots that has the less remaining possibilities
  (Tijh or Tija is minimal) is instantiated with that team

First model: results

# teams	# fails	Time (in s)
4	2	0.01
6	12	0.03
8	32	0.08
10	417	0.8
12	41	0.2
14	3,514	9.2
16	1,112	4.2
18	8,756	36
20	72,095	338
22	6,172,672	10h
24	6,391,470	12h

MIPLIB

MIP solver limit

Car sequencing

- Car sequencing pbs arise on assembly lines in factories in the automotive industry
- 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
- **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

The problem

- Determine in which order cars should be assembled, while:
  - building a certain number of cars per configuration
  - satisfying the capacity of each option.
- A real life problems: 1000 cars, 444 configurations, 25 options.

Example

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

Example

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 forbbiden

Example

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 fobbiden

Car sequencing: model

- A variable for each car
- The domain of a variable = the set of all possible configurations
- Constraints:
  - 1 global cardinality constraint which defines the number of time each configuration has to be built
  - 1 global sequencing constraint for each option

Car sequencing with CP

- **Global Sequencing Constraint**: GSC(X,V,min,max,q,{l_i},{u_i})
- A GCC (Global cardinality constraint) for the values of V + constraints stating that for each sequence S of q consecutive variables, at least min and at most max variables of S takes their values in V.

Global Sequencing Constraint

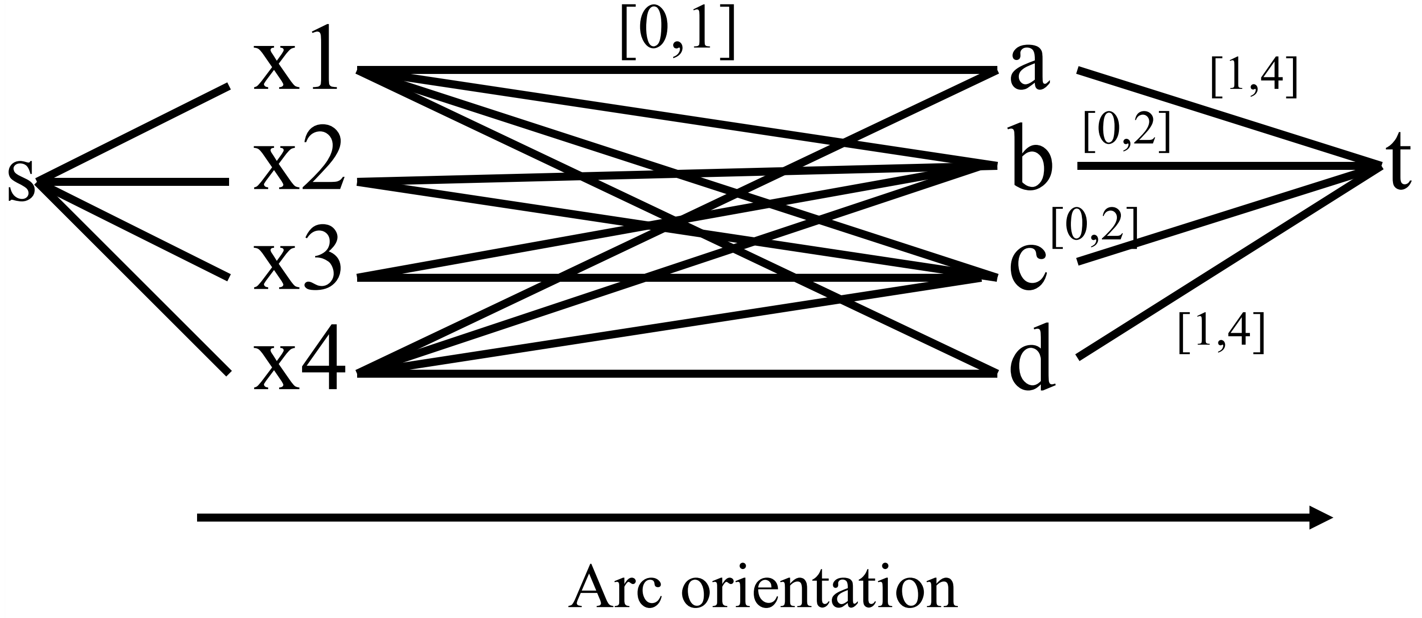
- GSC(X,V,min,max,q,{l_i},{u_i})
- Idea of the filtering algorithm: represent a GSC by a set of GCC + an ad hoc constraint that will link these GCCs

Global Cardinality Constraint

- $GCC(X,\{l_i\},\{u_i\})$
- Defined on a set X of variables, the number of times each value v_i can be taken must be in a given interval $[l_i, u_i]$
- Example: D(x1)={a,b,c,d}, D(x2)={a,b,c,d}, D(x3)={b,c},D(x4)={c,d}. Values b and c must be taken at most 2, values a and d must be taken at least 1.

Filtering algorithm for GCC

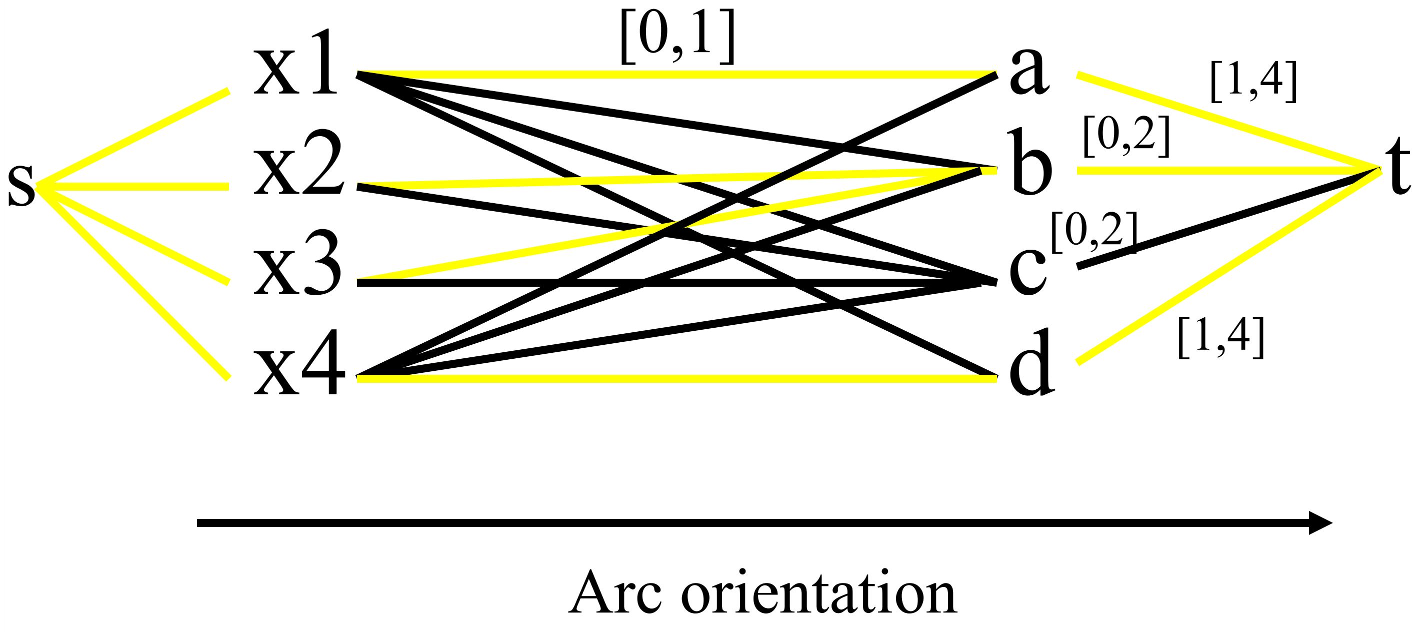
- Can be represented by a flow problem:



Flow value: |X| =4

Filtering algorithm for GCC

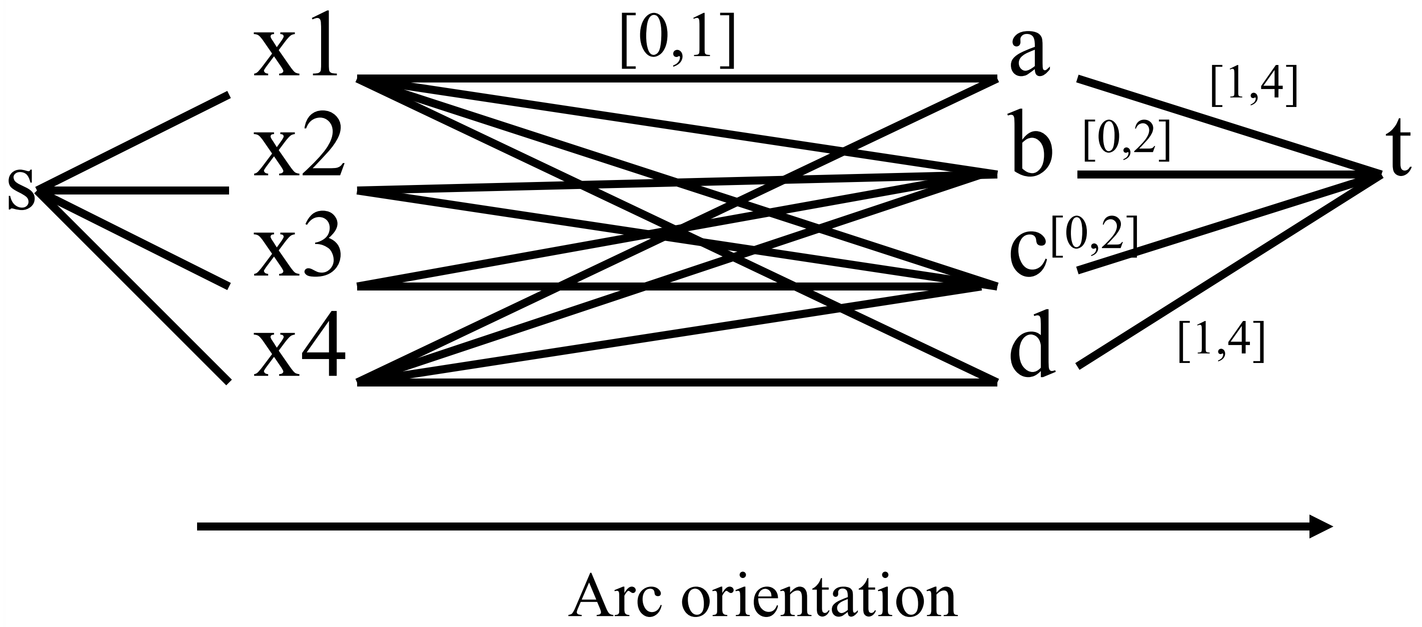
- A solution:



Flow value: |X| =4

Filtering algorithm for GCC

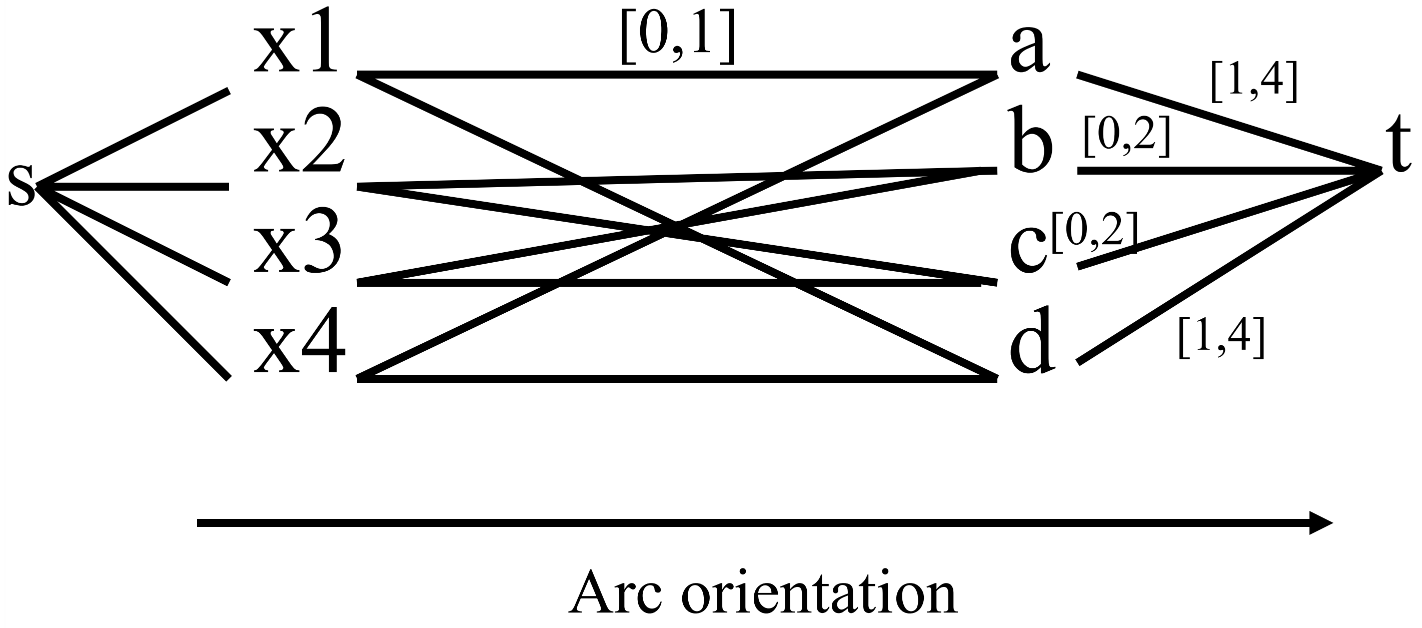
- Arc from variable to value that does not belong to any solution with a flow value =4 can be removed.



Flow value: |X| =4

Filtering algorithm for GCC

- Arc from variable to value that does not belong to any solution with a flow value =4 can be removed.



Flow value: |X| =4

Filtering algorithm for GCC

- Compute a feasible flow
- Compute the strongly connected components
- Remove every arc with flow value =0 for which the ends belong to two different components
- Linear algorithm achieving arc consistency
- work well due to (0,1) arcs

Global Sequencing Constraint

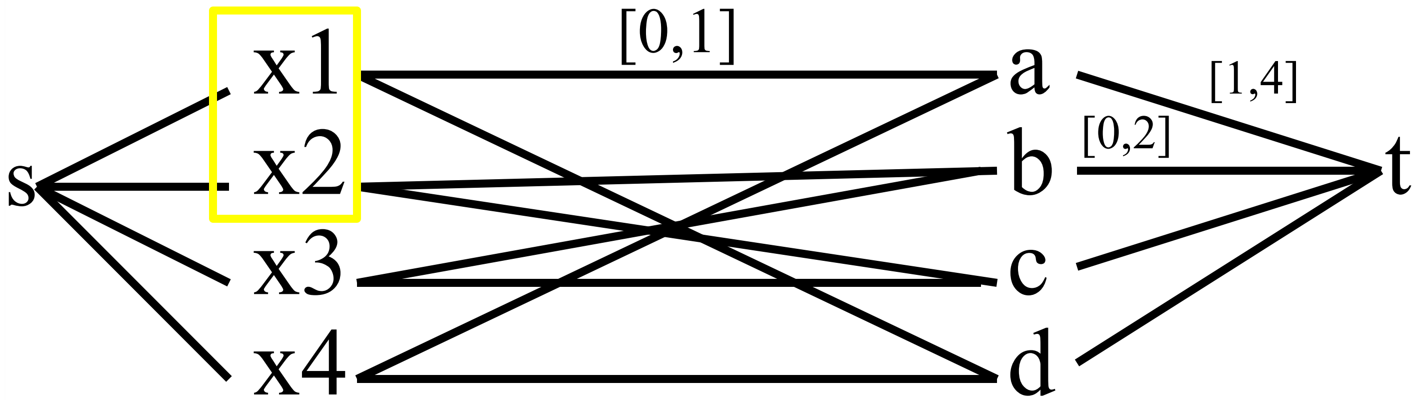
- GSC(X,V,min,max,q,{l_i},{u_i})
- A GCC (Global cardinality constraint) for the values of V + constraint stating that for each sequence S of q consecutive variables, at least min and at most max variables of S takes their values in V.

Abstract Values

- GSC(X,V,…): the values of D(X) - V are not constrained individually. For each sequence S they can be replaced (inside the constraint) by e(S) an abstract value.

Abstract Value

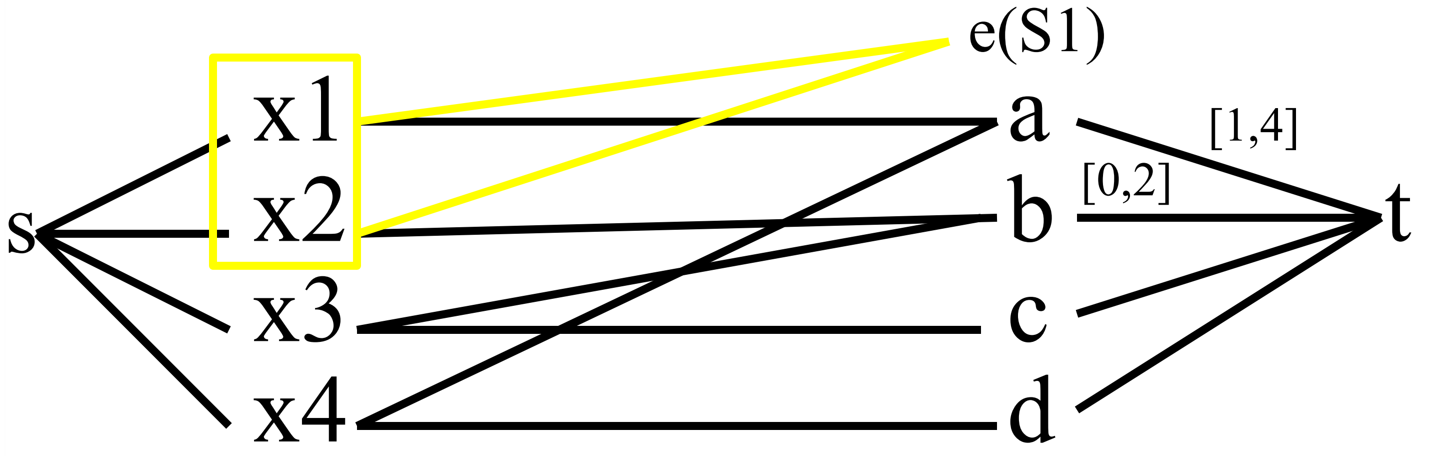
- GSC(X,V={a,b},min=0,max=1,q=2,…)



Constraints on values not in V are no longer considered

Abstract Value

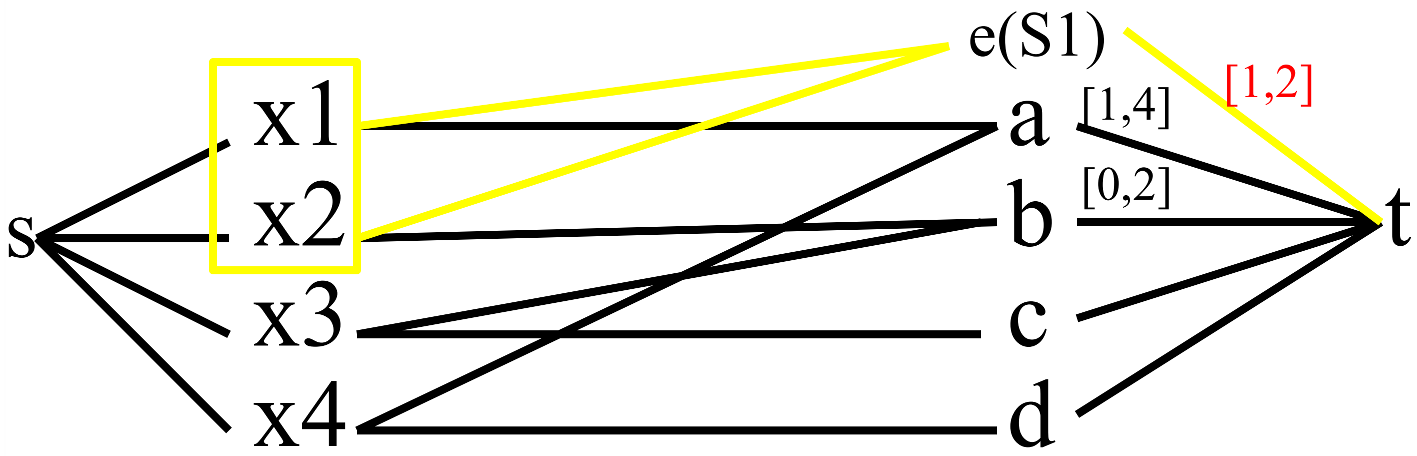
- GSC(X,V={a,b},min=0,max=1,q=2,…)



Values c and d does not belong to V, they are replaced by e(S1), for the sequence

Abstract Value

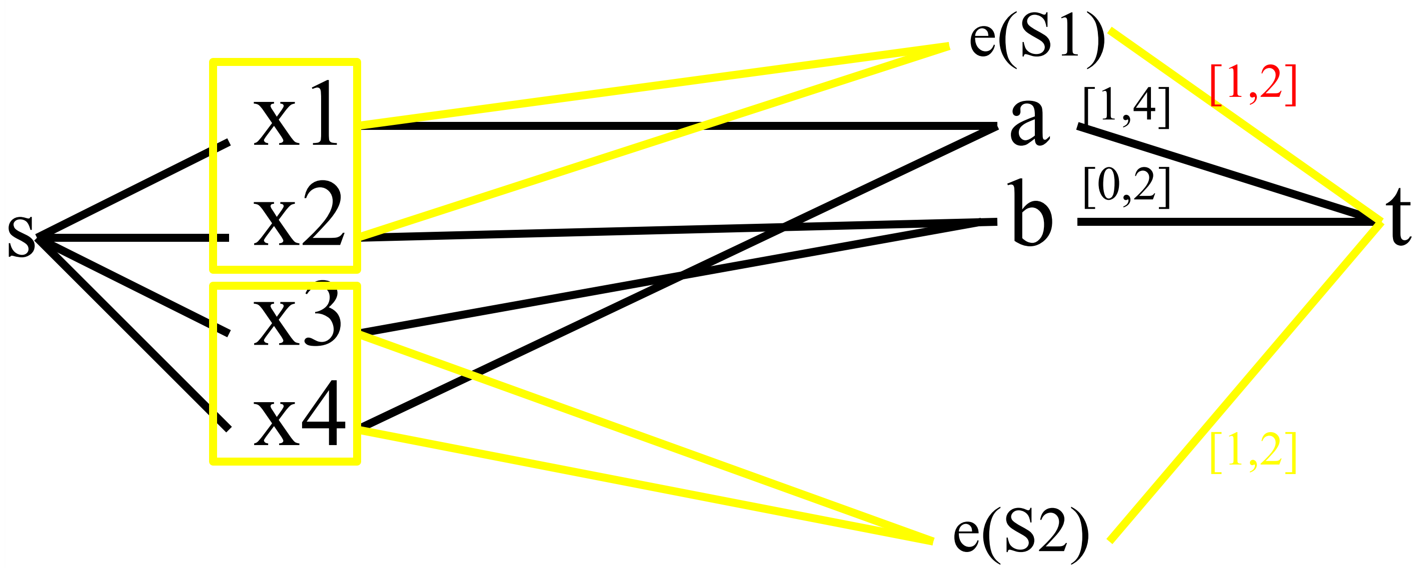
- GSC(X,V={a,b},min=0,max=1,q=2,…)



Value e(S1) must be taken at least |S|-max=2-1=1 and at most q -min = 2-0=2

Split of X into a partition of Sequence

- GSC(X,V={a,b},min=0,max=1,q=2,…)

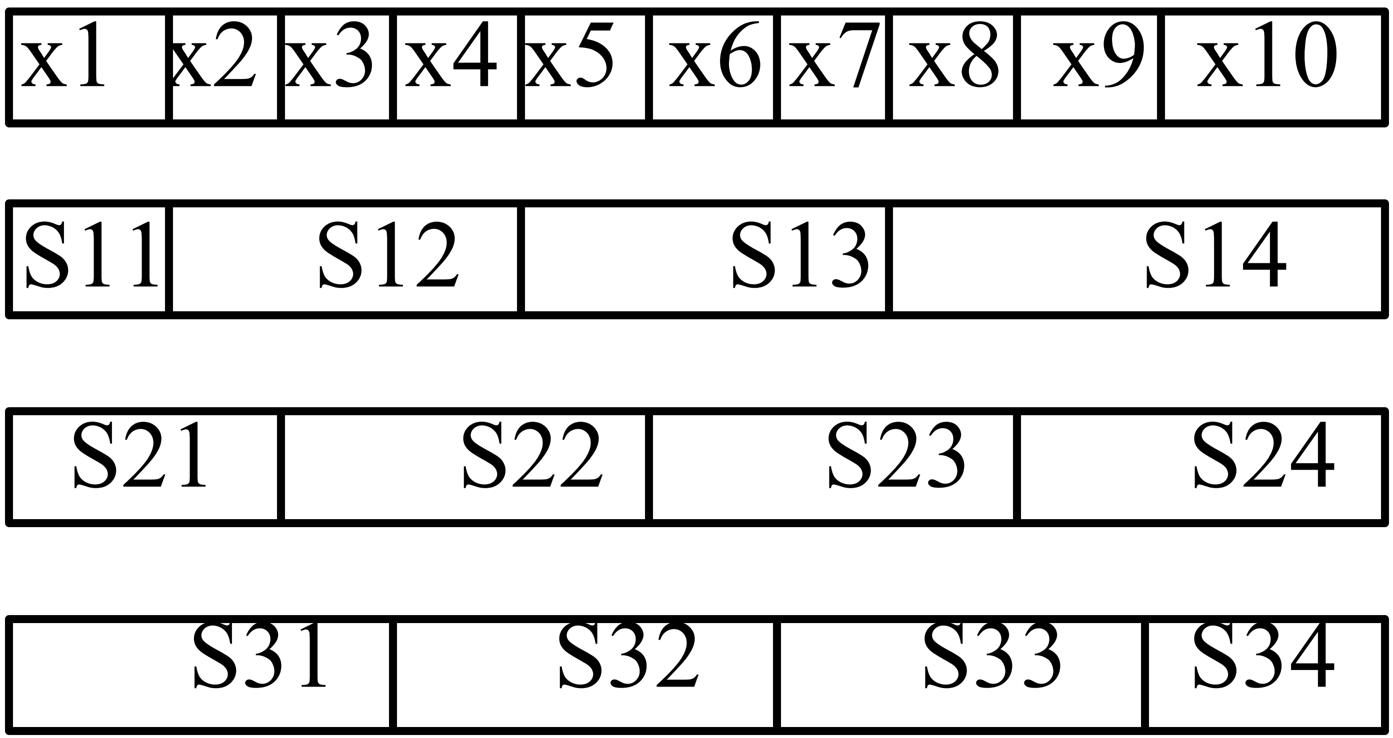


Red and Yellow arcs represent the constraints on sequences
Black arcs represent the global constraints on values of V

Split of X into partition of sequences

- Problem: an exponential number of partitions exist
- Solution: what is needed is just to have each sequence represented at least once. We propose to have |X| partitions simultaneously.
- For our example: P1={(x1,x2),(x3,x4)} and P2={(x1),(x2,x3),(x4)}

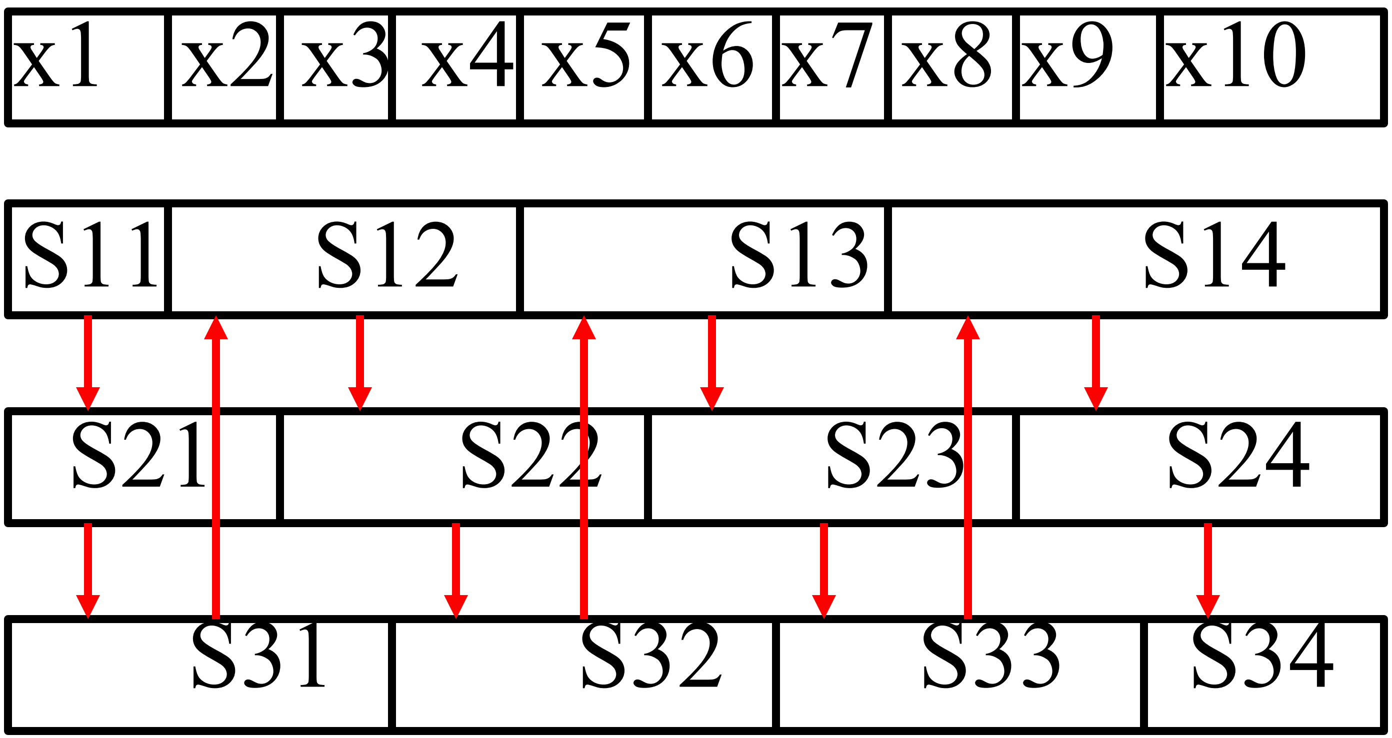
Partitions of sequences



Partitions of sequences

- Communication between sequences is necessary to solve the problem

Partitions of sequences



Si → Sj means Sj is a successor of Si

Successor of a sequence

- The successor of a sequence is the same sequence translated by one variable.

  S1: x1 x2 x3
  S2:   x2 x3 x4

- #e(S1) and #e(S2) can be linked:
  |#e(S1) - #e(S2)| $\leq$ 1

Constraints between sequences

- $|\#e(S1) - \#e(S2)| \leq 1$ must be implemented carefully:
  S1: x1 x2 x3
  S2: x2 x3 x4

- $x1=e(S1)$ and $x4 \neq e(S2) \Leftrightarrow \#e(S1) = \#e(S2) +1$
- $(x1=e(S1)$ and $x4=e(S2))$ or $(x1 \neq e(S1)$ and $x4 \neq e(s2))$
  $\Leftrightarrow \#e(S1) = \#e(S2)$
- $x1 \neq e(S1)$ and $x4=e(S2) \Leftrightarrow \#e(S1) = \#e(S2) - 1$

Car sequencing

- For each option a Global Sequencing Constraint is defined
- The filtering algorithm previously presented is used

Variable-Value Strategy

- How can we choose the next variable to instantiate and the value to assign to this variable?

Variable-Value Strategy

- How can we choose the next variable to instantiate and the value to assign to this variable?
- 1) Choose the most constraint option (e.g. 50 cars needs option 0 with a capacity 1/2)
  2) Choose the configurations (i.e. values) that requires this option
  3) Begin by the cars (i.e. variable) in the middle of the assembly line

Results

- Instances provided by Barbara Smith:
  100 cars, 25 configurations, 5 options
- We proved (in 1997) that:
  one instance has no solution in 3.5s
  one instance has no solution in 422s.
  As far as we know, this is currently the only one method which is able to obtain this result
- We solve some other open problems

Plan

- General Principles of Constraint Programming
- Global Constraints: advantages
- How to write a filtering algorithm?
- Examples: sports scheduling and car sequencing
- **Over-constrained problems**
- Discussion: quality of a FA, incrementality, closure, incomplete algorithms, power of a FA
- Conclusion

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

Meta Constraint

- si > 0 expresses that Ci is violated (distance to satisfaction)
- si = 0 expresses that Ci 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:

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

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

$$[\ \mathit{unsat} = \sum_{i=1}^{\#\,C_i} u_i\ ]$$

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

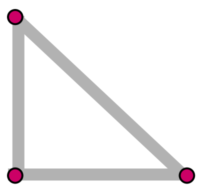
- How many variables must be removed to satisfy the constraint?
- Alldiff({x1,x2,x3,x4,x5})
  (a,a,a,b,b) cost = 3
  (a,a,a,a,b) cost = 3

Primal graph based partition cost

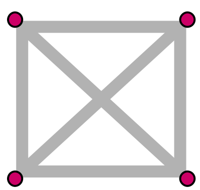
- For a global constraint corresponding to a conjunction of constraints. Number of the constraints in the conjunction that are violated
- Alldiff({x1,x2,x3,x4,x5})
  (a,a,a,b,b) cost = triangle(a,a,a) + pair (b,b)
  = 3 + 2 =5
  (a,a,a,a,b) cost = quadrangle (a,a,a,a)
  = 6

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

Soft global constraints

- For the alldiff, GCC, stretch, and regular constraints specific algorithms have been designed. These FA are able to take into account a cost variable w.r.t. the defined cost (see papers at CP conferences)

Global constraints with costs

- Integration of the costs within the constraint is **quite important**
- Alldiff with costs: quite important
  - [Caseau & Laburthe CP98] only consistency checking
  - [Focacci & Milano CP-AI-OR 99] filtering based on reduced costs
  - [Regin CP99] arc consistency

Plan

- General Principles of Constraint Programming
- Global Constraints: advantages
- How to write a filtering algorithm?
- Examples: sports scheduling and car sequencing
- Over-constrained problems
- **Discussion: quality of a FA, incrementality, closure, incomplete algorithms, power of a FA**
- Conclusion

Good Filtering Algorithm

- What is a good filtering algorithm ? Or what is a poor one ? (**The FA is considered and not the idea of using an OR algorithm**)
- GAC-Schema:
  - After 1 modification:
  consistency in O(CC))
  arc consistency in O(ndCC)
  - After k modifications
  consistency in O(CC)
  arc consistency in O(ndCC)

Good or Poor ?

- If O(CC) for consistency then arc consistency:
  - O(ndCC) poor
  (reached by GAC-Schema)

Good or Poor ?

- If O(CC) for consistency then arc consistency:
  - O(ndCC) poor
  (reached by GAC-Schema)
  - O(CC) good!

Good or Poor ?

- If O(CC) for consistency then arc consistency:
  - O(ndCC) poor
  (reached by GAC-Schema)
  - O(CC) good!
  - and between ? Not so bad ...

Good FA

- Alldiff,
- global cardinality constraint

Medium FA

- Algorithm such that arc consistency is in: O(nCC) (a factor of d is gained):
  - Global cardinality with cost
  - Symmetric alldiff (alldiff U {xi =j <=> xj =i})
  - sum with binary inequalities
  - edge-finder

Poor FA ?

- I hope there is none :-)
- Be careful with algorithms that successively try all values of variables (or ranges)

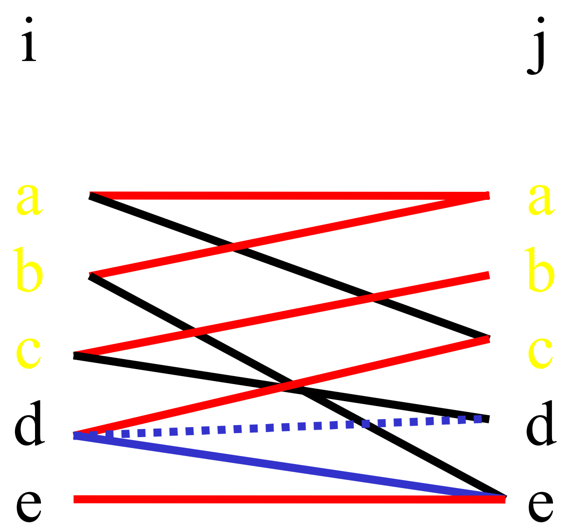
Perfect FA ?

- Idea: FA has no cost.
- Complexity of the FA **is always the same as** the complexity of the consistency checking algorithm
- All the possible cases are considered and not only the worst case
- Another possibility O(1) per deletion.
- Alldiff: 1 modification: if the deleted arc does not belong to the current maximum matching, then consistency in O(1), AC in O(nd): not perfect
- No perfect FA is known? (maybe only x < y)

Incremental algorithms

- An incremental approach is not always the best. (cf IJCAI-2001 paper on AC-2001)
- The consequence of the deletions is a good approach if the number of modifications is less than the number of remaining things. Otherwise it is not good.
- The incremental aspect is quite important for a FA

AC-2001 vs AC-6



Last=a	Sja={a,b}
Last=a	Sjb={c}
Last=b	Sjc={d}
Last=c	Sjd={}
Last=e	Sje={e}

If Δ(j)={a,b,c} and Δ(i)={a,b,c} then:
recomputation from scratch: 4 operations
study of the consequences of the deletions: 6 operations

Adaptive Algorithm: Adaptive AC

- $A = \Sigma_{v \text{ in } \Delta(j)} (|Svj| + 1)$
- $B = |D(i)|$
- If A < B then run AC-6
- If B < A then run AC-2001

If $2|\Delta(j)| < |D(i)|$ then run AC-6
else run AC-2001

Closure or not ?

- Is the FA closed w.r.t a property?
- Consider the values deleted by the FA. The consequence of these new deletions can be:
  1) taken into account by the same pass of the FA (alldiff)
  2) ignored by the same pass of the FA (Table)
- 1) no need to call again the FA
  2) need to call again the FA
- It is a choice.

Amortized Complexity

- It is possible to define the complexity in regards to the number of deletions (ex: O(CC) per deletion)
- Symmetric alldiff:
  AC: from every var, run algorithm A. Algorithm A can remove some values.
  AC Complexity: nO(A). Pb systematic.
- Other FA: pick one variable, run A, and set k=#deletions. You gain k runs!

  Complexity O(A) per deletion

Incomplete algorithms

- The constraint is an NP-Hard problem
- Try to characterize what is done
- useful in practice but sometimes difficult to handle:
  - no fixpoint (largest clique depends on the way the graph is defined):
  - less constraints can lead to more pruning
  - debug is difficult
- global sequencing constraint (NP-Hard with fixpoint)

Power of Filtering Algorithms

- Arc consistency is a strong property but it is sometimes costly
- Weaker consistencies exist: range consistency, bound consistency (see nice papers of C-G Quimper, A. Lopez-Ortiz et al about alldiff and GCC)
- However, arc consistency has some advantages

Advantages of Arc Consistency

- AC is much more robust. During the modeling phase it is useful to use strong FA. It is rare to be able to solve a problem with weaker consistency and not with AC
- There is a room for improvements of AC algorithms
- For binary constraints old story FC vs MAC
- We should study more strong properties like Singleton Arc Consistency.

Plan

- General Principles of Constraint Programming
- Global Constraints: advantages
- How to write a filtering algorithm?
- Examples: sports scheduling and car sequencing
- Over-constrained problems
- Discussion: quality of a FA, incrementality, closure, incomplete algorithms, power of a FA
- **Conclusion**

Conclusion

- Filtering algorithm are one of the main strength of CP. Define your model by using them
- If you write an FA:
  - try to write a good or a medium one. Do not forget that GAC-Schema exists
  - take care of the semantics of the constraint and especially the triggering of the FA

Conclusion

- Incremental algorithms are not always the best.
- General filtering algorithms are efficient in lack of other algorithms, when some predefined FA exist, use them.
- Over-constrained problems: use the constraint structure