Constraint Programming

Lecture 3. Global Constraints. Solving CSPs.

Ruslan Sadykov

INRIA Bordeaux—Sud-Ouest

20 January 2022

Lignes directrices

Global constraints

- « Simple » constraints
- `All-diff`
- `GCC`
- Constraints for scheduling

« Traditional » algorithms to solve CSPs

Parameterising the algorithms

Importance of global constraints

A global constraint is a union of simple constraints.

Use of global constraints

- facilitates the modeling
  (smaller number of constraints, libraries of constraints) ;
- accelerates the solving
  (specialised, and thus efficient, algorithms for propagation).

Important

Global constraints contribute a lot to the succes of Constraint Programming in practice.

Catalogue of global constraints :
http ://sofdem.github.io/gccat/

Importance of global constraints

A global constraint is a union of simple constraints.

Use of global constraints

- facilitates the modeling
  (smaller number of constraints, libraries of constraints) ;
- accelerates the solving
  (specialised, and thus efficient, algorithms for propagation).

Important

Global constraints contribute a lot to the succes of Constraint Programming in practice.

Catalogue of global constraints :
http ://sofdem.github.io/gccat/

Importance of global constraints

A global constraint is a union of simple constraints.

Use of global constraints

- facilitates the modeling
  (smaller number of constraints, libraries of constraints) ;
- accelerates the solving
  (specialised, and thus efficient, algorithms for propagation).

Important

Global constraints contribute a lot to the succes of Constraint Programming in practice.

Catalogue of global constraints :
http ://sofdem.github.io/gccat/

Importance of global constraints

A global constraint is a union of simple constraints.

Use of global constraints

- facilitates the modeling
  (smaller number of constraints, libraries of constraints) ;
- accelerates the solving
  (specialised, and thus efficient, algorithms for propagation).

Important

Global constraints contribute a lot to the succes of Constraint Programming in practice.

Catalogue of global constraints :
http ://sofdem.github.io/gccat/

Importance of global constraints

A global constraint is a union of simple constraints.

Use of global constraints

- facilitates the modeling
  (smaller number of constraints, libraries of constraints) ;
- accelerates the solving
  (specialised, and thus efficient, algorithms for propagation).

Important

Global constraints contribute a lot to the succes of Constraint Programming in practice.

Catalogue of global constraints :
http ://sofdem.github.io/gccat/

Lignes directrices

- Global constraints
  - « Simple » constraints
  - `All-diff`
  - `GCC`
  - Constraints for scheduling

- « Traditional » algorithms to solve CSPs

- Parameterising the algorithms

Global constraint `ScalProd`

$$\texttt{scal\_prod}(X_1, \ldots, X_n, c_1, \ldots, c_n, v)$$

- Equivalent to

$$\sum_{i=1}^{n} c_i X_i = v.$$

- Rules for achieving arc-B-consistency $(D_{X_i} = [\underline{x_i}, \overline{x_i}], c_i > 0)$

$$\underline{x_i} \leftarrow \max\left\{ \underline{x_i}, \frac{v - \sum\limits_{1\leq j\leq n:\ j\neq i} \max\left\{c_j\underline{x_j}, c_j\overline{x_j}\right\}}{c_i} \right\}$$

$$\overline{x_i} \leftarrow \min\left\{ \overline{x_i}, \frac{v - \sum\limits_{1\leq j\leq n:\ j\neq i} \min\left\{c_j\underline{x_j}, c_j\overline{x_j}\right\}}{c_i} \right\}$$

Global constraint `ScalProd`

$$\texttt{scal\_prod}(X_1, \ldots, X_n, c_1, \ldots, c_n, v)$$

- Equivalent to

$$\sum_{i=1}^{n} c_i X_i = v.$$

- Rules for achieving arc-B-consistency $\left(D_{X_i} = [\underline{x_i}, \overline{x_i}], c_i > 0\right)$

$$\underline{x_i} \leftarrow \max \left\{ \underline{x_i}, \quad \frac{v - \sum\limits_{1 \le j \le n:\ j \ne i} \max\left\{c_j \underline{x_j}, c_j \overline{x_j}\right\}}{c_i} \right\}$$

$$\overline{x_i} \leftarrow \min \left\{ \overline{x_i}, \quad \frac{v - \sum\limits_{1 \le j \le n:\ j \ne i} \min\left\{c_j \underline{x_j}, c_j \overline{x_j}\right\}}{c_i} \right\}$$

Global constraint `Element`

$\texttt{element}(X, v_1, \ldots, v_n, Y)$

- Equivalent to

$$X = v_Y.$$

  We should have $D_Y \subseteq \{1, \ldots, n\}$.
- This constraint allows one to use variables as indices.
- Can be represented as the following explicit constraint :

$$(X, Y) \in \{(v_i, i)\}_{1 \leq i \leq n}.$$

- So, the arc-consistency can be achieved using classic algorithms (AC-3, AC-4, etc).

Global constraint `Element`

$\texttt{element}(X, v_1, \ldots, v_n, Y)$

- Equivalent to
  $$X = v_Y.$$
  We should have $D_Y \subseteq \{1, \ldots, n\}$.
- This constraint allows one to use variables as indices.
- Can be represented as the following explicit constraint :
  $$(X, Y) \in \{(v_i, i)\}_{1 \leq i \leq n}.$$
- So, the arc-consistency can be achieved using classic algorithms (AC-3, AC-4, etc).

Global constraint `Element`

$\texttt{element}(X, v_1, \ldots, v_n, Y)$

- Equivalent to
$$X = v_Y.$$
We should have $D_Y \subseteq \{1, \ldots, n\}$.
- This constraint allows one to use variables as indices.
- Can be represented as the following explicit constraint :
$$(X, Y) \in \{(v_i, i)\}_{1 \leq i \leq n}.$$
- So, the arc-consistency can be achieved using classic algorithms (AC-3, AC-4, etc).

Lignes directrices

Global constraints

- « Simple » constraints
- `All-diff`
- `GCC`
- Constraints for scheduling

« Traditional » algorithms to solve CSPs

Parameterising the algorithms

Global constraint `all-different`

$\texttt{all-different}(X_1, \ldots, X_n)$

- Replaces $\frac{n^2}{2}$ binary constraints

$$X_i \neq X_j, \quad 1 \leq i < j \leq n.$$

- The most used constraint in practice : assignment, permutation,...
- We can achieve arc-consistency in time $O(\sqrt{n}nd)$ using graph theory (Régin, 94) tools (to compare with complexity $O(n^2d^2)$ if we take constraints one by one).

Global constraint `all-different`

$$\texttt{all-different}(X_1, \ldots, X_n)$$

- Replaces $\frac{n^2}{2}$ binary constraints

$$X_i \neq X_j, \quad 1 \leq i < j \leq n.$$

- The most used constraint in practice : assignment, permutation,...
- We can achieve arc-consistency in time $O(\sqrt{n}nd)$ using graph theory (Régin, 94) tools (to compare with complexity $O(n^2d^2)$ if we take constraints one by one).

Global constraint `all-different`

$\texttt{all-different}(X_1, \ldots, X_n)$

- Replaces $\frac{n^2}{2}$ binary constraints
$$X_i \neq X_j, \quad 1 \leq i < j \leq n.$$
- The most used constraint in practice : assignment, permutation,...
- We can achieve arc-consistency in time $O(\sqrt{n}nd)$ using graph theory (Régin, 94) tools (to compare with complexity $O(n^2d^2)$ if we take constraints one by one).

Matching problem in a graph

Bipartite graph is a graph vertices of which can be partitioned in two subsets U and V such that each edge incident to one vertex in U and to one in V.

Matching in a graph is a set of disjoint edges (which do have common incident vertices)

Maximum matching in a graph is a maximum size matching.

Alternating path (given a matching) is a path in which the edges belong alternatively to the matching and not to the matching.

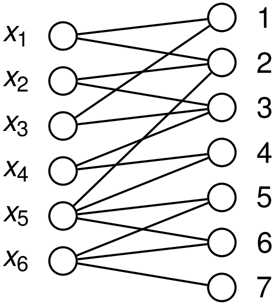
`All-different` constraint : propagation

Algorithm :

1. We construct a bipartite « value » graph.
2. We search for a maximum matching (if its size is $< n$, then there is no solution).
3. Given this matching, we establish edges which do not belong to
    - an alternating circuit,
    - an alternating path such that one of its extremities is a free vertex,
    - the matching found.
4. We remove values which correspond to these edges.

Exemple :

$D_{x1} = \{1,2\}$, $D_{x2} = \{2,3\}$,
$D_{x3} = \{1,3\}$, $D_{x4} = \{3,4\}$,
$D_{x5} = \{2,4,5,6\}$,
$D_{x6} = \{5,6,7\}$
$\texttt{all-different}(x_1,\ldots,x_6)$



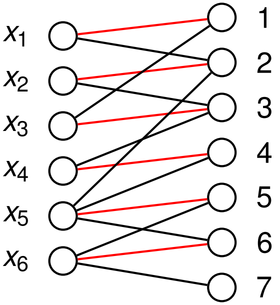
`All-different` constraint : propagation

Algorithm :

1. We construct a bipartite « value » graph.
2. We search for a maximum matching (if its size is $< n$, then there is no solution).
3. Given this matching, we establish edges which do not belong to
   - an alternating circuit,
   - an alternating path such that one of its extremities is a free vertex,
   - the matching found.
4. We remove values which correspond to these edges.

Exemple :

$D_{x1} = \{1, 2\}$, $D_{x2} = \{2, 3\}$,
$D_{x3} = \{1, 3\}$, $D_{x4} = \{3, 4\}$,
$D_{x5} = \{2, 4, 5, 6\}$,
$D_{x6} = \{5, 6, 7\}$
`all-different`$(x_1, \ldots, x_6)$



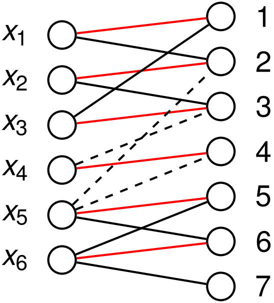
`All-different` constraint : propagation

Algorithm :

1. We construct a bipartite « value » graph.
2. We search for a maximum matching (if its size is $< n$, then there is no solution).
3. Given this matching, we establish edges which do not belong to
    - an alternating circuit,
    - an alternating path such that one of its extremities is a free vertex,
    - the matching found.
4. We remove values which correspond to these edges.

Exemple :

$D_{x1} = \{1, 2\}$, $D_{x2} = \{2, 3\}$,
$D_{x3} = \{1, 3\}$, $D_{x4} = \{3, 4\}$,
$D_{x5} = \{2, 4, 5, 6\}$,
$D_{x6} = \{5, 6, 7\}$
`all-different`$(x_1, \ldots, x_6)$



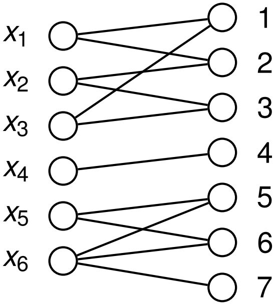
`All-different` constraint : propagation

Algorithm :

1. We construct a bipartite « value » graph.
2. We search for a maximum matching (if its size is $< n$, then there is no solution).
3. Given this matching, we establish edges which do not belong to
    - an alternating circuit,
    - an alternating path such that one of its extremities is a free vertex,
    - the matching found.
4. We remove values which correspond to these edges.

Exemple :

$D_{x1} = \{1, 2\}$, $D_{x2} = \{2, 3\}$,
$D_{x3} = \{1, 3\}$, $D_{x4} = \{\cancel{3}, 4\}$,
$D_{x5} = \{\cancel{2}, \cancel{4}, 5, 6\}$,
$D_{x6} = \{5, 6, 7\}$
$\texttt{all-different}(x_1, \ldots, x_6)$



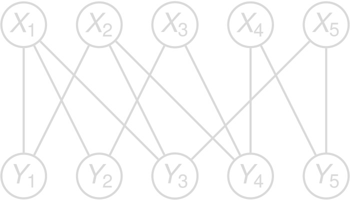
Global constraint `assignment`

`assignment`$(X_1, \ldots, X_n, Y_1, \ldots, Y_n)$

- It is the symmetric `all-different` constraint :

$$X_i = j \Leftrightarrow Y_j = i, \quad 1 \leq i, j \leq n.$$

  We should have $D_X \subseteq \{1, \ldots, n\}, D_Y \subseteq \{1, \ldots, n\}$.
- Used for mutual assignment.
- We can achieve arc-consistency using the same algorithm as for the `all-different` constraint.



Global constraint `assignment`

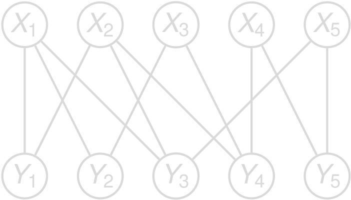
`assignment`$(X_1, \ldots, X_n, Y_1, \ldots, Y_n)$

- It is the symmetric `all-different` constraint :

$$X_i = j \Leftrightarrow Y_j = i, \quad 1 \leq i, j \leq n.$$

  We should have $D_X \subseteq \{1, \ldots, n\}, D_Y \subseteq \{1, \ldots, n\}$.

- Used for mutual assignment.
- We can achieve arc-consistency using the same algorithm as for the `all-different` constraint.



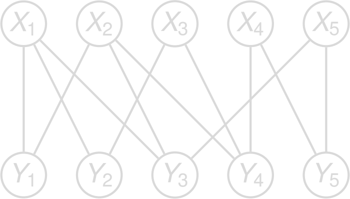
Global constraint `assignment`

`assignment`$(X_1, \ldots, X_n, Y_1, \ldots, Y_n)$

- It is the symmetric `all-different` constraint :

$$X_i = j \Leftrightarrow Y_j = i, \quad 1 \leq i, j \leq n.$$

  We should have $D_X \subseteq \{1, \ldots, n\}, D_Y \subseteq \{1, \ldots, n\}$.
- Used for mutual assignment.
- We can achieve arc-consistency using the same algorithm as for the `all-different` constraint.



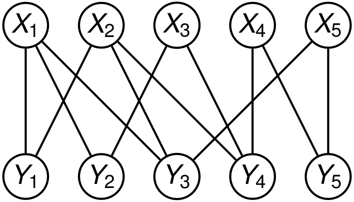
Global constraint `assignment`

$$\texttt{assignment}(X_1, \ldots, X_n, Y_1, \ldots, Y_n)$$

- It is the symmetric `all-different` constraint :

  $$X_i = j \Leftrightarrow Y_j = i, \quad 1 \leq i, j \leq n.$$

  We should have $D_X \subseteq \{1, \ldots, n\}, D_Y \subseteq \{1, \ldots, n\}$.
- Used for mutual assignment.
- We can achieve arc-consistency using the same algorithm as for the `all-different` constraint.



Lignes directrices

Global constraints

- « Simple » constraints
- `All-diff`
- `GCC`
- Constraints for scheduling

« Traditional » algorithms to solve CSPs

Parameterising the algorithms

Global constraint GCC

$$\texttt{GCC}(X_1, \ldots, X_n, v_1, \ldots, v_k, l_1, \ldots, l_k, u_1, \ldots, u_k)$$

- This global cardinality constraint is a generalisation of `all-different` : the number of times each value v_j is taken should be inside interval $[l_j, u_j]$ (for `all-different`, $l_j = 0$, $u_j = 1$, $\forall j$).
- Often used in practice : complicated assignment, distribution,...
- We can achieve the arc-consistency for constraint GCC in time $O(n^2 d)$ by using the maximum flow algorithm (Régin, 99).

Global constraint GCC

$$\texttt{GCC}(X_1, \ldots, X_n, v_1, \ldots, v_k, l_1, \ldots, l_k, u_1, \ldots, u_k)$$

- This global cardinality constraint is a generalisation of `all-different` : the number of times each value v_j is taken should be inside interval $[l_j, u_j]$ (for `all-different`, $l_j = 0$, $u_j = 1$, $\forall j$).
- Often used in practice : complicated assignment, distribution,...
- We can achieve the arc-consistency for constraint GCC in time $O(n^2d)$ by using the maximum flow algorithm (Régin, 99).

Global constraint `GCC`

$$\texttt{GCC}(X_1, \ldots, X_n, v_1, \ldots, v_k, l_1, \ldots, l_k, u_1, \ldots, u_k)$$

- This global cardinality constraint is a generalisation of `all-different` : the number of times each value v_j is taken should be inside interval $[l_j, u_j]$ (for `all-different`, $l_j = 0$, $u_j = 1$, $\forall j$).
- Often used in practice : complicated assignment, distribution,...
- We can achieve the arc-consistency for constraint `GCC` in time $O(n^2 d)$ by using the maximum flow algorithm (Régin, 99).

Maximum flow in a graph

Notations

Let $D = (V, A)$ be an directed graph in which to each arc $(i, j) \in A$ we associate a capacity u_{ij}.

Flow definition

A flow in graph D is a function f defined on arcs of D :

- $0 \leq f_{ij} \leq u_{ij}, \forall (i, j) \in A,$
- $\sum\limits_{i:\ (i,j) \in A} f_{ij} = \sum\limits_{i:\ (j,i) \in A} f_{ji}, \forall j \in V \setminus \{s, d\}.$

Problem

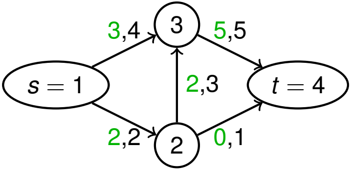
Find the maximum flow which can be sent from s to t.

Construction of maximum flow

Algorithm :

1. We start with a feasible flow f (for example, zero flow).
2. We construct the residual directed graph $R = (V, A')$:
   - $f_{ij} > 0 \Leftrightarrow (j, i) \in A'$;
   - $f_{ij} < u_{ij} \Leftrightarrow (i, j) \in A'$.
3. If there exists a path from s to t in the residual graph, we increase the flow along this path as much as possible.
4. If such path does not exists, the flow is maximum.

Example :

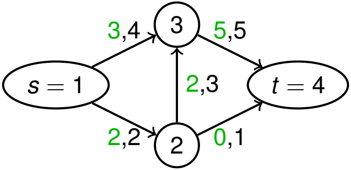


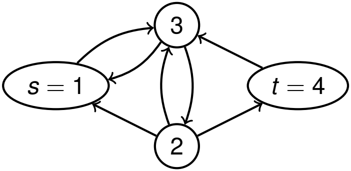
Construction of maximum flow

Algorithm :

1. We start with a feasible flow f (for example, zero flow).
2. We construct the residual directed graph $R = (V, A')$:
    - $f_{ij} > 0 \Leftrightarrow (j, i) \in A'$;
    - $f_{ij} < u_{ij} \Leftrightarrow (i, j) \in A'$.
3. If there exists a path from s to t in the residual graph, we increase the flow along this path as much as possible.
4. If such path does not exists, the flow is maximum.

Example :



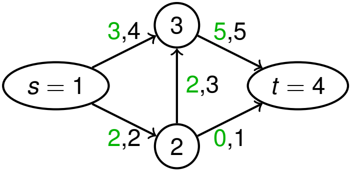


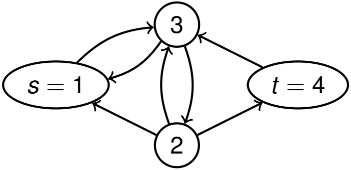
Construction of maximum flow

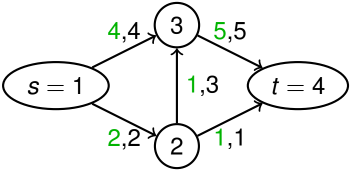
Algorithm :

1. We start with a feasible flow f (for example, zero flow).
2. We construct the residual directed graph $R = (V, A')$:
    - $f_{ij} > 0 \Leftrightarrow (j, i) \in A'$;
    - $f_{ij} < u_{ij} \Leftrightarrow (i, j) \in A'$.
3. If there exists a path from s to t in the residual graph, we increase the flow along this path as much as possible.
4. If such path does not exists, the flow is maximum.

Example :





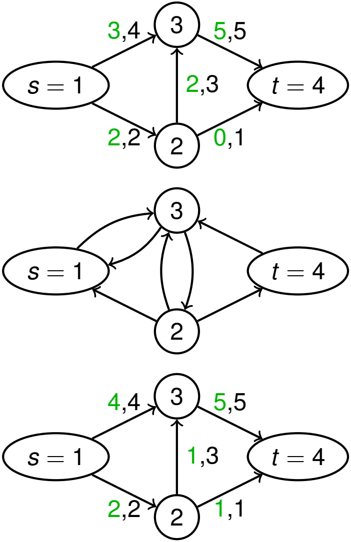


Construction of maximum flow

Algorithm :

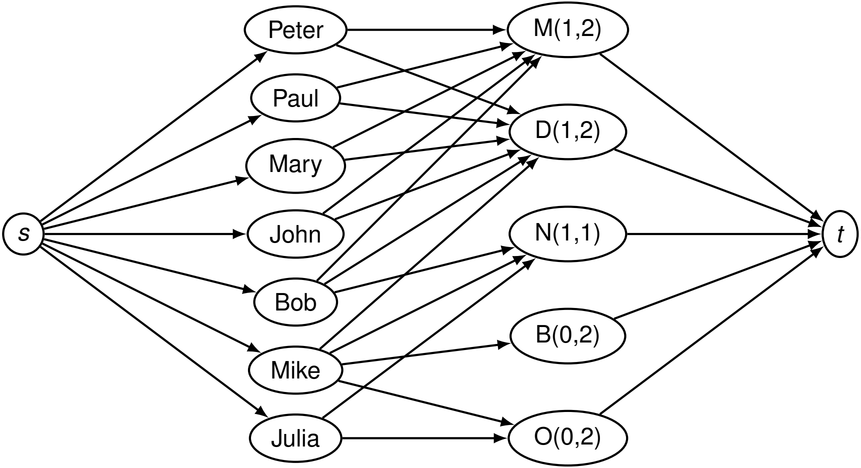
1. We start with a feasible flow f (for example, zero flow).
2. We construct the residual directed graph $R = (V, A')$:
   - $f_{ij} > 0 \Leftrightarrow (j, i) \in A'$;
   - $f_{ij} < u_{ij} \Leftrightarrow (i, j) \in A'$.
3. If there exists a path from s to t in the residual graph, we increase the flow along this path as much as possible.
4. If such path does not exists, the flow is maximum.

Example :



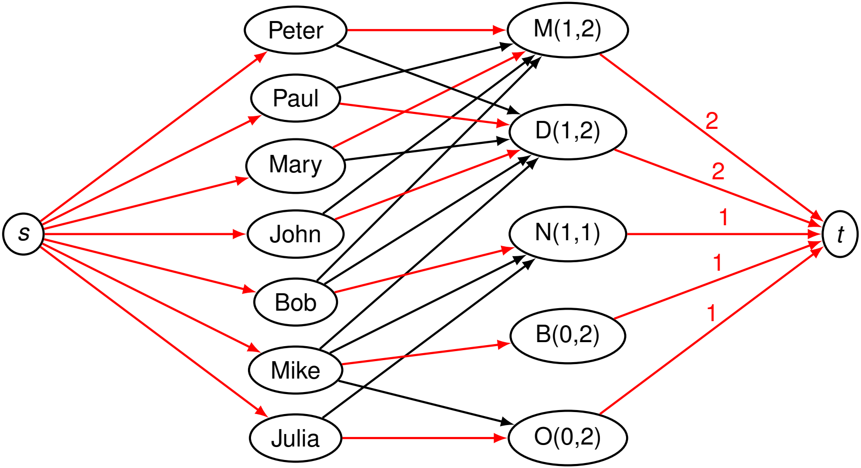
Propagation of constraint GCC I

We construct the « value » graph :



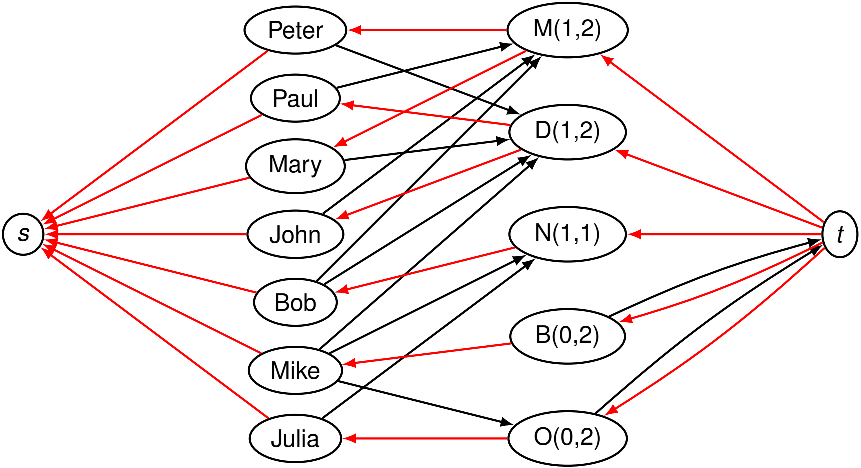
Propagation of constraint GCC II

We find the maximum flow :



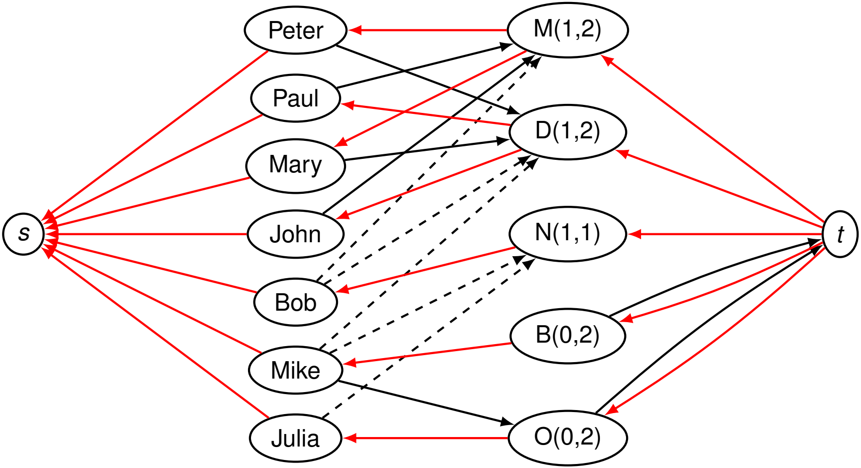
Propagation of constraint GCC III

We construct the residual graph induces by the maximum flow :



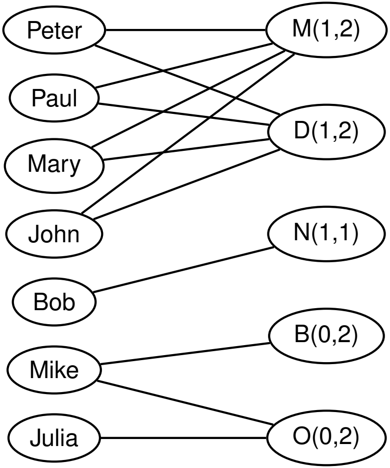
Propagation of constraint GCC IV

We establish non-saturated arcs which between variables and values which do not belong to any circuit in the residual graph (decomposition in strongly connected components) :



Propagation of constraint GCC V

We remove values which correspond to these edges :



Lignes directrices

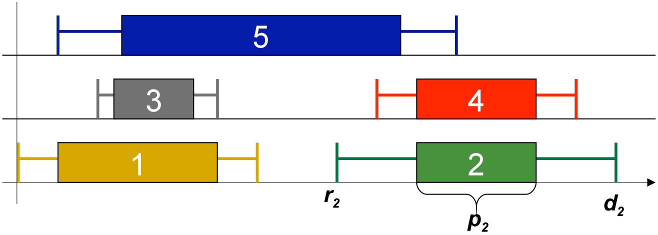
Global constraints

- « Simple » constraints
- `All-diff`
- `GCC`
- Constraints for scheduling

« Traditional » algorithms to solve CSPs

Parameterising the algorithms

Global constraint `disjunctive`



$$\texttt{disjunctive}(X_1, \ldots, X_n, p_1, \ldots, p_n)$$

- Replaces $\frac{n^2}{2}$ logical binary constraints :

$$X_i + p_i \leq X_j \bigvee X_i \geq X_j + p_j, \quad \forall i, j : \; i \neq j.$$

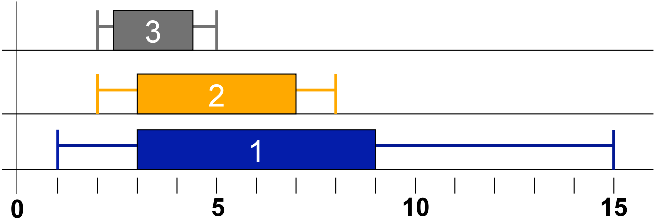
- Often used for scheduling problems (often with $D_{x_i} = [r_i, d_i - p_i]$)

Global constraint `disjunctive` : propagation I

- Achieving arc-B-consistency for this constraint is *NP*-hard.
- Weaker (« *Edge-Finding* ») propagation is used :

$$\max_{i \in \Omega} d_i - \min_{i \in \Omega \cup \{k\}} r_i < \sum_{\Omega \cup \{k\}} p_i \quad \Rightarrow \quad \Omega \text{ précède } k$$

$$\Rightarrow \quad r_k \leftarrow \max_{\Omega' \subseteq \Omega} \left\{ \min_{i \in \Omega'} r_i + \sum_{i \in \Omega'} p_i \right\}$$



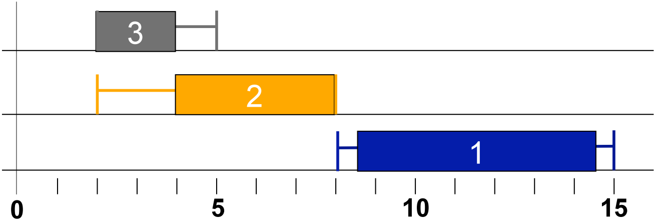
$k = 1$
$\Omega = \{2, 3\}$
$r_k \leftarrow 8$

Global constraint `disjunctive` : propagation I

- Achieving arc-B-consistency for this constraint is *NP*-hard.
- Weaker (« *Edge-Finding* ») propagation is used :

$$\max_{i \in \Omega} d_i - \min_{i \in \Omega \cup \{k\}} r_i < \sum_{\Omega \cup \{k\}} p_i \quad \Rightarrow \quad \Omega \text{ précède } k$$

$$\Rightarrow \quad r_k \leftarrow \max_{\Omega' \subseteq \Omega} \left\{ \min_{i \in \Omega'} r_i + \sum_{i \in \Omega'} p_i \right\}$$



$k = 1$

$\Omega = \{2, 3\}$

$r_k \leftarrow 8$

Global constraint `disjunctive` : propagation II

- « *Edge-Finding* » detects if job k should be executed before (after) **all jobs** in set Ω.
  We can verify all possible pairs (Ω, k) in time $O(n \log n)$.
  (Carlier & Pinson, 94).
- « *Not-First/Not-Last* » detects if job k should be execute before (after) **at least one job** in set Ω.
  We can verify all possible pairs (Ω, k) in time $O(n \log n)$.
  (Vilím, 04).
- « *Detectable precedences* »,...

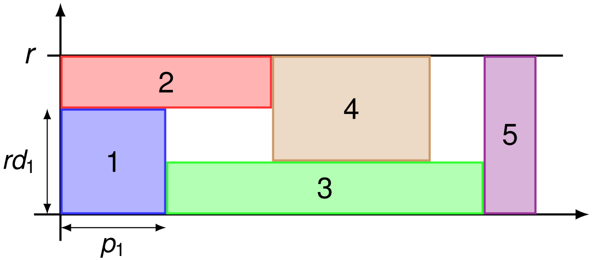
Global constraint `disjunctive` : propagation II

- « *Edge-Finding* » detects if job k should be executed before (after) **all jobs** in set Ω.
  We can verify all possible pairs (Ω, k) in time $O(n \log n)$.
  (Carlier & Pinson, 94).
- « *Not-First/Not-Last* » detects if job k should be execute before (after) **at least one job** in set Ω.
  We can verify all possible pairs (Ω, k) in time $O(n \log n)$.
  (Vilím, 04).
- « *Detectable precedences* »,...

Global constraint `disjunctive` : propagation II

- « *Edge-Finding* » detects if job k should be executed before (after) **all jobs** in set Ω.
  We can verify all possible pairs (Ω, k) in time $O(n \log n)$.
  (Carlier & Pinson, 94).
- « *Not-First/Not-Last* » detects if job k should be execute before (after) **at least one job** in set Ω.
  We can verify all possible pairs (Ω, k) in time $O(n \log n)$.
  (Vilím, 04).
- « *Detectable precedences* »,...

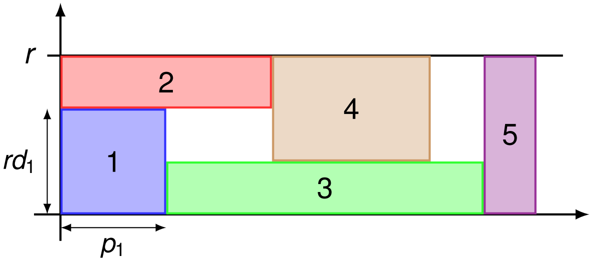
Global constraint `Cumulative`



$$\texttt{cumulative}(X_1, \ldots, X_n, p_1, \ldots, p_n, rd_1, \ldots, rd_n, r)$$

- Jobs should not overlap ;
  $+$ each job i consumes rd_i units of the resource ;
  $+$ at each time moment we cannot use more than r units of the resource.
- It is a generalisation of `disjunctive`, for which $rd_i = 1, \forall i$, et $r = 1$.

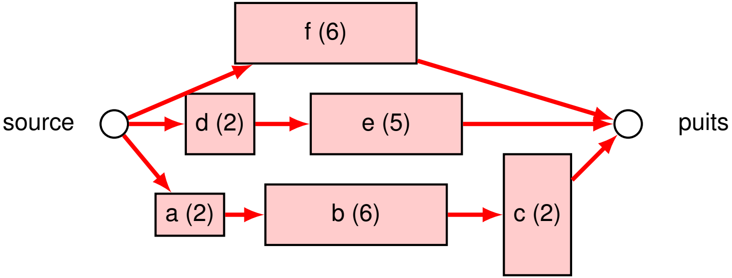
Global constraint `Cumulative`



$$\texttt{cumulative}(X_1, \ldots, X_n, p_1, \ldots, p_n, rd_1, \ldots, rd_n, r)$$

- Jobs should not overlap ;
  $+$ each job i consumes rd_i units of the resource ;
  $+$ at each time moment we cannot use more than r units of the resource.
- It is a generalisation of `disjunctive`, for which $rd_i = 1, \forall i$, et $r = 1$.

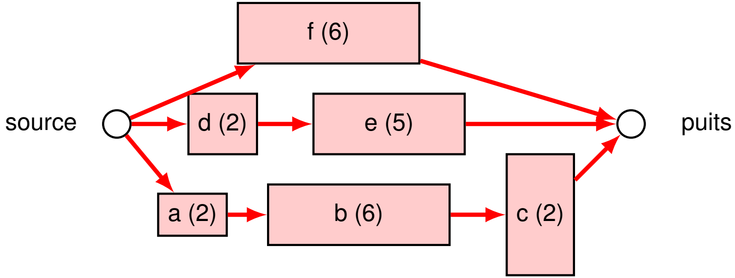
Global constraint `cumulative` : example



If $X_c \leq 9$, $X_e \leq 4$, $X_f \leq 14$, then, after propagation of precedence constraints

$$D(X_a) = [0,1], \quad D(X_b) = [2,3], \quad D(X_c) = [8,9],$$
$$D(X_d) = [0,2], \quad D(X_e) = [2,4], \quad D(X_f) = [0,14].$$

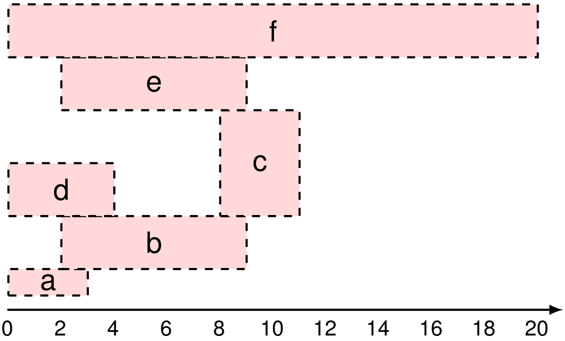
Global constraint `cumulative` : example



If $X_c \leq 9$, $X_e \leq 4$, $X_f \leq 14$, then, after propagation of precedence constraints

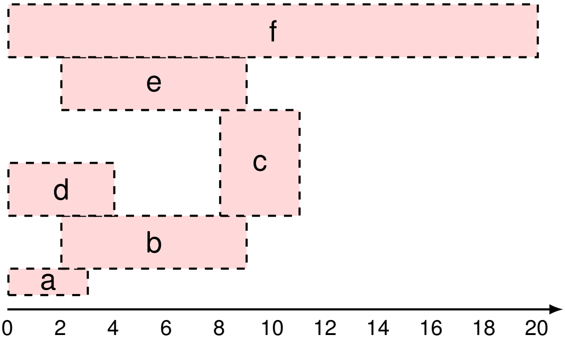
$$D(X_a) = [0,1], \quad D(X_b) = [2,3], \quad D(X_c) = [8,9],$$
$$D(X_d) = [0,2], \quad D(X_e) = [2,4], \quad D(X_f) = [0,14].$$

`Cumulative` : *Time-table* propagation (1)

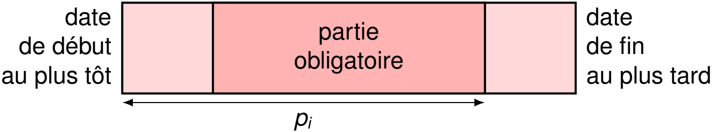


Obligatory parts

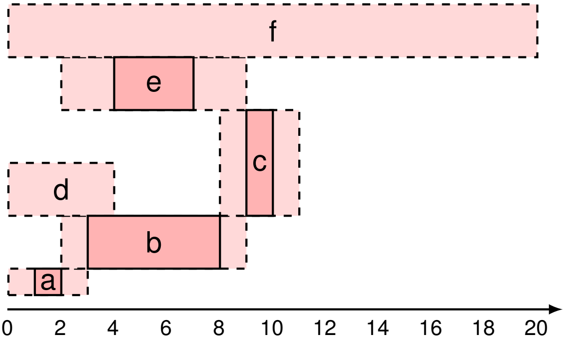
`Cumulative` : *Time-table* propagation (1)



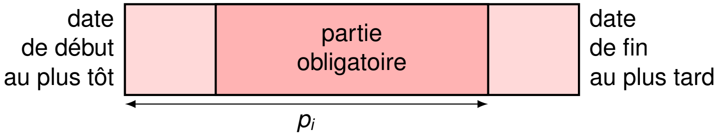
Obligatory parts



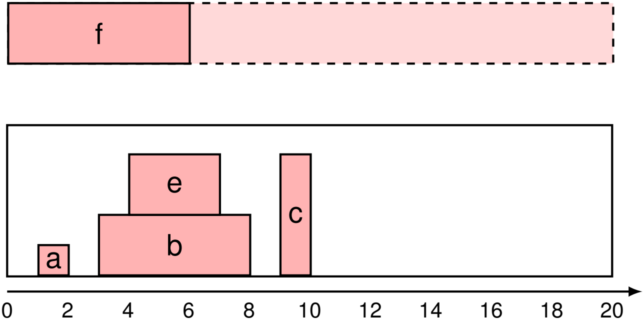
`Cumulative` : *Time-table* propagation (1)



Obligatory parts



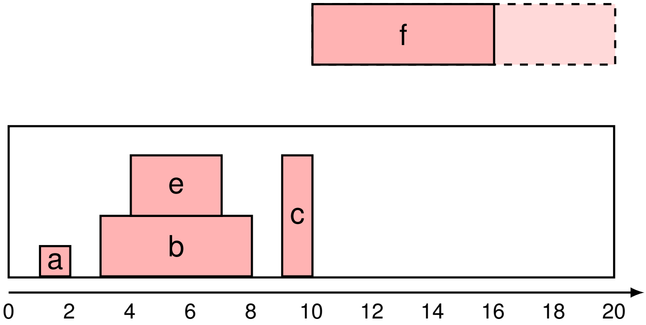
`Cumulative` : *Time-table* propagation (2)



Complexity

Time-table propagation can be done in time $O(n \log n)$ (Lahrichi, 82)

`Cumulative` : *Time-table* propagation (2)



Complexity

Time-table propagation can be done in time $O(n \log n)$ (Lahrichi, 82)

Constraint `Cumulative` : other « propagations »

- « *Edge-Finding* » detects if job k should start before (finish after) **all the jobs** in set Ω**.**
  We can verify all possible pairs (Ω, k) in time $O(n^2)$ (Kameugne et al, 11) or in time $O(rn \log n)$ (Vilim, 09)
- « *Not-First/Not-Last* » detects if job k should be execute after (before) **at least one job** in set Ω**.**
  We can verify all possible pairs (Ω, k) in time $O(n^3)$. (Baptiste et al., 01).

Constraint `Cumulative` : other « propagations »

- « *Edge-Finding* » detects if job k should start before (finish after) **all the jobs** in set Ω.
  We can verify all possible pairs (Ω, k) in time $O(n^2)$ (Kameugne et al, 11) or in time $O(rn \log n)$ (Vilim, 09)
- « *Not-First/Not-Last* » detects if job k should be execute after (before) **at least one job** in set Ω.
  We can verify all possible pairs (Ω, k) in time $O(n^3)$. (Baptiste et al., 01).

Lignes directrices

Global constraints

- « Simple » constraints
- `All-diff`
- `GCC`
- Constraints for scheduling

« Traditional » algorithms to solve CSPs

Parameterising the algorithms

Notations and definitions

n — number of variables

e — number of constraints

d — upper bounds for the domains size

An instantiation $I = \{\langle x_i, v_i \rangle\}_{i \in K}$ is an assignment of values $\{v_i\}_{i \in K}$ to variables $\{x_i\}_{i \in K}$.

An instantiation is complete if $K = \{1, \ldots, n\}$.

Notations and definitions

n — number of variables

e — number of constraints

d — upper bounds for the domains size

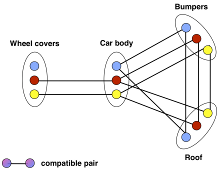
An instantiation $I = \{\langle x_i, v_i \rangle\}_{i \in K}$ is an assignment of values $\{v_i\}_{i \in K}$ to variables $\{x_i\}_{i \in K}$.

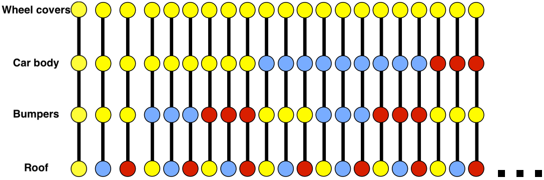
An instantiation is complete if $K = \{1, \ldots, n\}$.

Trivial solution

Algorithme 1 : Generate and test

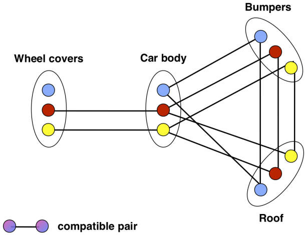
```
foreach complete instantiation I do
    if I satisfies all constraints then
        return I
return « no solutions »
```

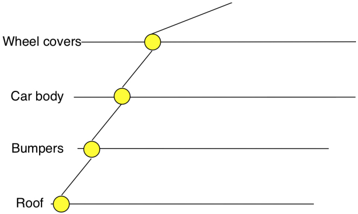




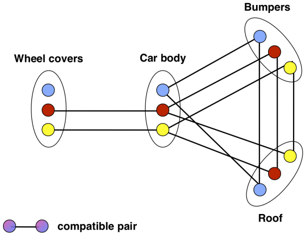
Complexity : $O(ed^n)$

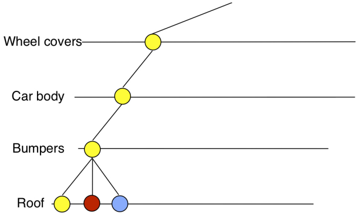
Chronological backtrack



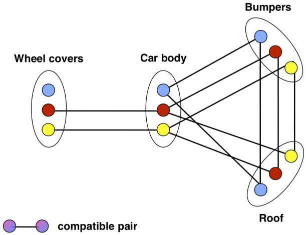


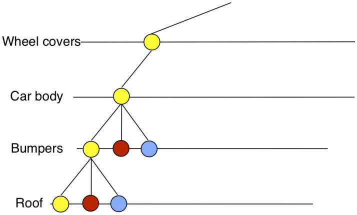
Chronological backtrack



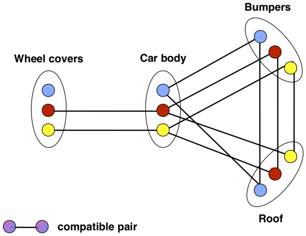


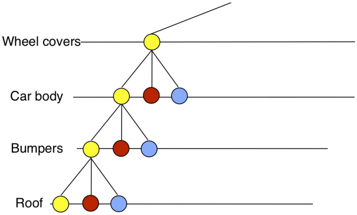
Chronological backtrack



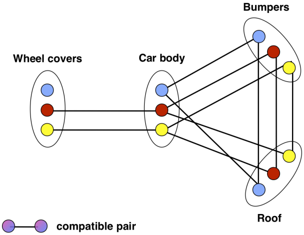


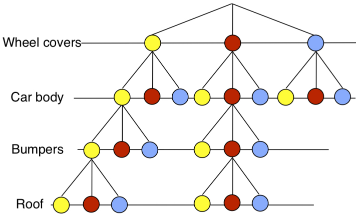
Chronological backtrack





Chronological backtrack





Algorithme 2 : `Backtrack`(I,k,v)

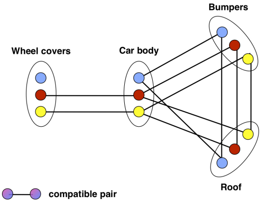
```
$I \leftarrow I \cup \{\langle x_k, v\rangle\}$ ;
if $I$ satisfies all constraints then
    if $I$ is complete $(k = n)$ then $I$ is a solution ; exit ;
    else
        foreach $a \in D_{x_{k+1}}$ do Backtrack $(I, k+1, a)$ ;
```

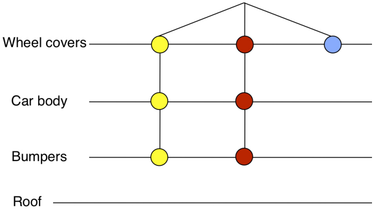
Complexity : $O(ed^n)$, but better in practice

Forward Checking

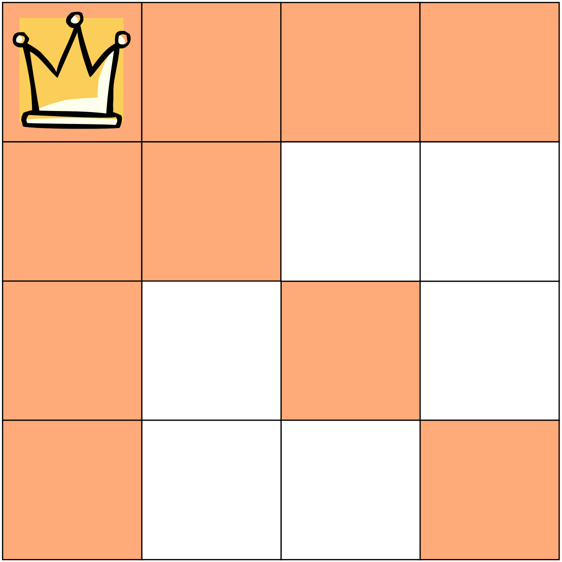
Algorithme 3 : `ForwardCheck`(I,D,k,v)

```
$I \leftarrow I \cup \{\langle x_k, v\rangle\}$ ;
remove from $D$ all values incompatible with $x_k = v$ ;
if there is no empty domain then
    if $I$ is complete then $I$ is a solution ; exit ;
    else
        foreach $a \in D_{x_{k+1}}$ do
            ForwardCheck $(I,D,k+1,a)$
```

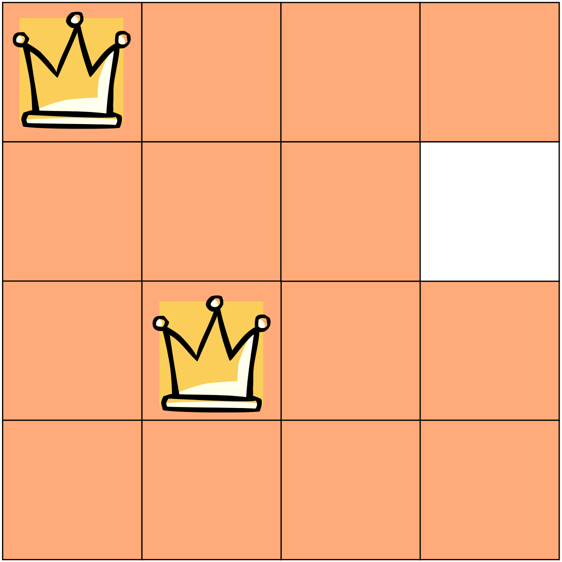




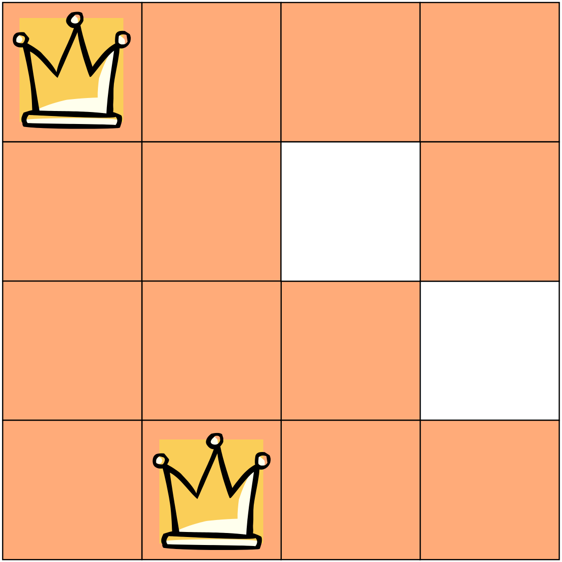
Forward Checking : *N* queens



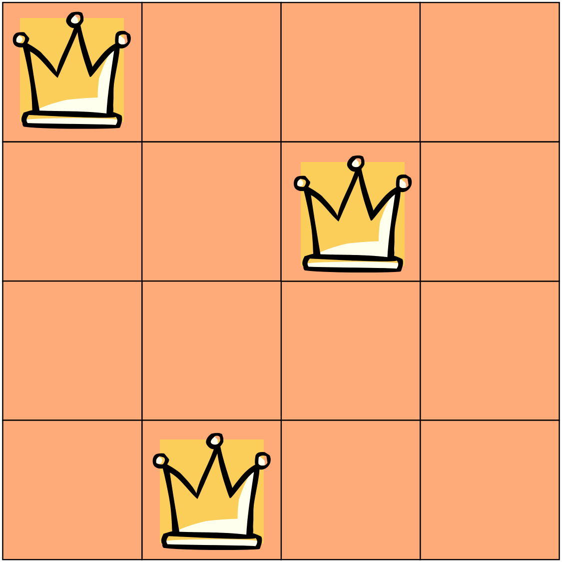
Forward Checking : *N* queens



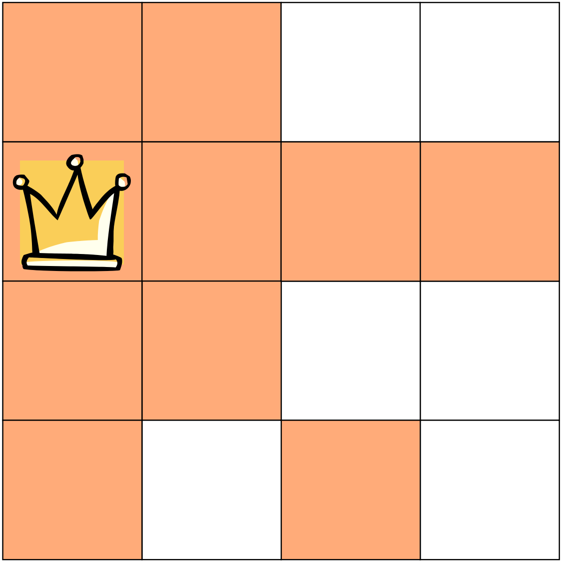
Forward Checking : *N* queens



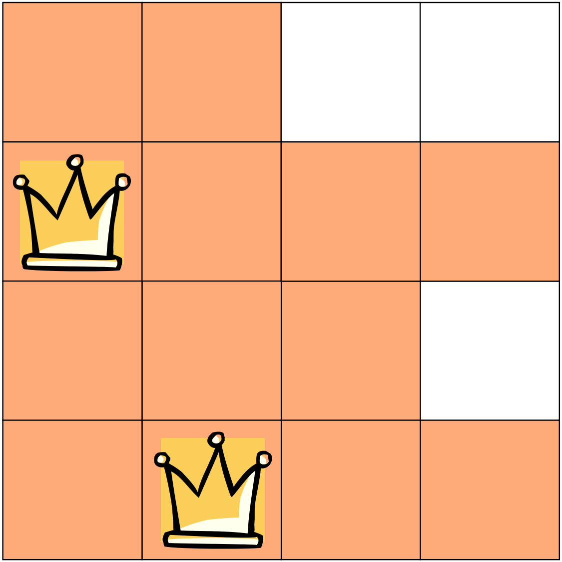
Forward Checking : *N* queens



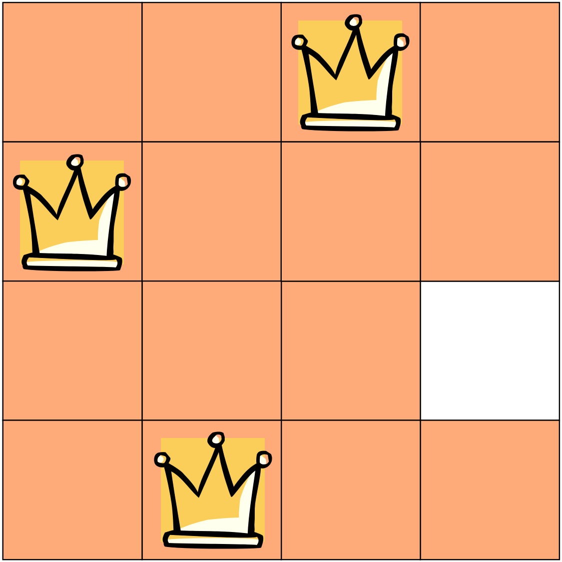
Forward Checking : *N* queens



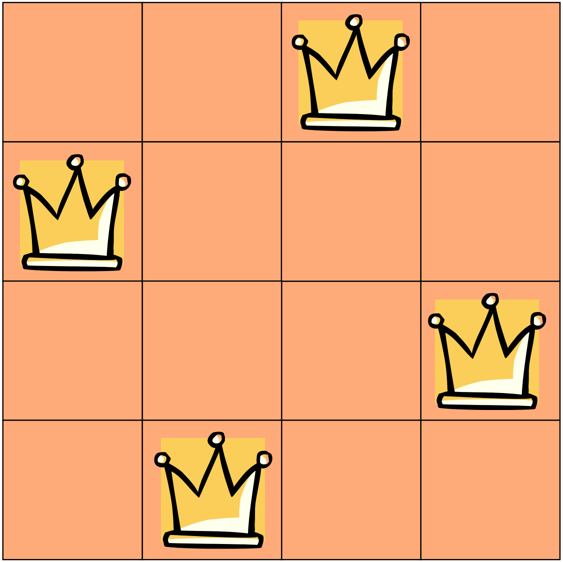
Forward Checking : *N* queens



Forward Checking : *N* queens

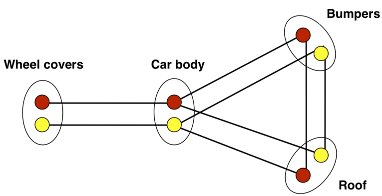


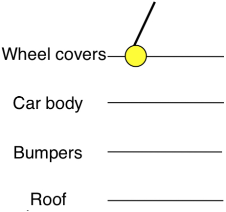
Forward Checking : *N* queens



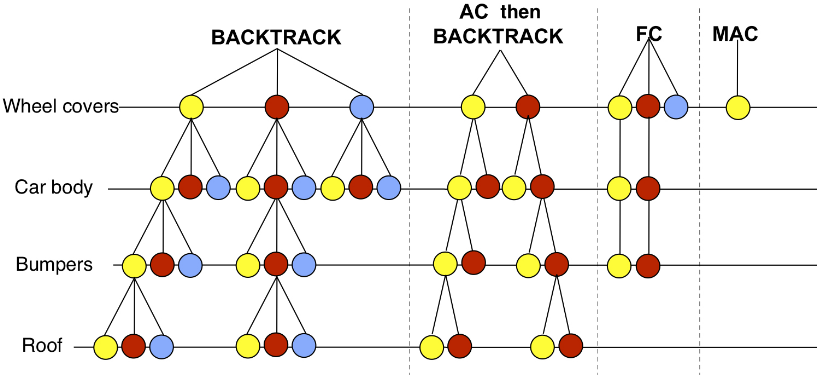
Maintaining Local Consistency

```
Algorithme 4 : MLC($I$,$D$,$k$,$v$)
$I \leftarrow I \cup \{\langle x_k, v\rangle\}$ ;
remove $\{\langle x_k, a\rangle\}_{a \in D_{x_k}, a \neq v}$ from $D$ and propagate ;
if all domains are not empty then
    si $I$ is complete alors  $I$ is a solution ; exit ;
    else
        foreach $a \in D_{x_{k+1}}$ do
            MLC ($I$,$D$,$k+1$,$a$) ;
            remove $\{\langle x_{k+1}, a\rangle\}$ from $D$ and propagate ;
```





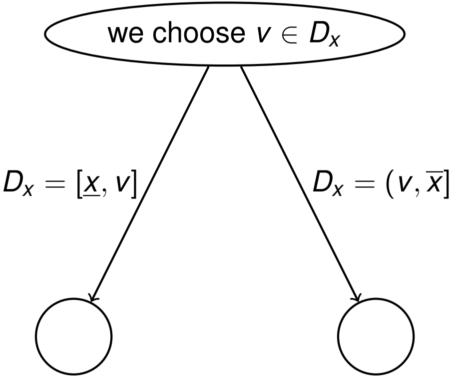
Comparing the algorithms



Problems with infinite domains

If there are interval domains, instead of instantiating variables, we divide such domain (usually in two)

Let $D_x = [\underline{x}, \overline{x}]$, then



Lignes directrices

Global constraints

- « Simple » constraints
- `All-diff`
- `GCC`
- Constraints for scheduling

« Traditional » algorithms to solve CSPs

Parameterising the algorithms

Algorithm options

- In the algorithms we have just seen, there are choices to be done :
    - in which order we instantiate variables ;
    - in which order we assign values ;
    - which local consistency we achieve.
- These decisions (called « **heuristics** ») are extremely important for efficiency of the algorithms.
    - If we dive into a branch without solutions, we can spend a lot of time before we understand this.
    - The first decisions are particularly important (when we are in upper part of the search tree).

Algorithm options

- In the algorithms we have just seen, there are choices to be done :
  - in which order we instantiate variables ;
  - in which order we assign values ;
  - which local consistency we achieve.
- These decisions (called « **heuristics** ») are extremely important for efficiency of the algorithms.
  - If we dive into a branch without solutions, we can spend a lot of time before we understand this.
  - The first decisions are particularly important (when we are in upper part of the search tree).

Algorithm options

- In the algorithms we have just seen, there are choices to be done :
  - in which order we instantiate variables ;
  - in which order we assign values ;
  - which local consistency we achieve.
- These decisions (called « **heuristics** ») are extremely important for efficiency of the algorithms.
  - If we dive into a branch without solutions, we can spend a lot of time before we understand this.
  - The first decisions are particularly important (when we are in upper part of the search tree).

Algorithm options

- In the algorithms we have just seen, there are choices to be done :
  - in which order we instantiate variables ;
  - in which order we assign values ;
  - which local consistency we achieve.
- These decisions (called « **heuristics** ») are extremely important for efficiency of the algorithms.
  - If we dive into a branch without solutions, we can spend a lot of time before we understand this.
  - The first decisions are particularly important (when we are in upper part of the search tree).

Algorithm options

- In the algorithms we have just seen, there are choices to be done :
  - in which order we instantiate variables ;
  - in which order we assign values ;
  - which local consistency we achieve.
- These decisions (called « **heuristics** ») are extremely important for efficiency of the algorithms.
  - If we dive into a branch without solutions, we can spend a lot of time before we understand this.
  - The first decisions are particularly important (when we are in upper part of the search tree).

Heuristics for the variables order

There are two types :

- Static — order is fixed before executing the algorithm.
- Dynamic — order may change during the algorithm (may be even different in different branches).

Possible objectives :

- Minimizing the research space
- Minimizing the average depth in the tree
- Minimizing the number of branches
- ...

Static heuristic are based on properties of the constraint network of the problem, especially its **width** and its **bandwidth.**

Heuristics for the variables order

There are two types :

- Static — order is fixed before executing the algorithm.
- Dynamic — order may change during the algorithm (may be even different in different branches).

Possible objectives :

- Minimizing the research space
- Minimizing the average depth in the tree
- Minimizing the number of branches
- ...

Static heuristic are based on properties of the constraint network of the problem, especially its **width** and its **bandwidth**.

Heuristics for the variables order

There are two types :

- Static — order is fixed before executing the algorithm.
- Dynamic — order may change during the algorithm (may be even different in different branches).

Possible objectives :

- Minimizing the research space
- Minimizing the average depth in the tree
- Minimizing the number of branches
- ...

Static heuristic are based on properties of the constraint network of the problem, especially its **width** and its **bandwidth.**

Dynamic order of variables I

Objectives

- Minimise the expected number of branches
- Minimise the expected depth of branches

Principle

First-fail : we choose variables which are « difficult to satisfy », we do not postpose difficult decisions

If a CSP is weakly constrained, opposite heuristic may be more efficient.

Dynamic order of variables I

Objectives

- Minimise the expected number of branches
- Minimise the expected depth of branches

Principle

First-fail : we choose variables which are « difficult to satisfy », we do not postpose difficult decisions

If a CSP is weakly constrained, opposite heuristic may be more efficient.

Dynamic order of variables I

Objectives

- Minimise the expected number of branches
- Minimise the expected depth of branches

Principle

First-fail : we choose variables which are « difficult to satisfy », we do not postpose difficult decisions

If a CSP is weakly constrained, opposite heuristic may be more efficient.

Dynamic order of variables II

$\mathrm{dom}(x|p)$: size of the domain of x after assignments p

Possible choice of a variable

- one which has the smallest domain : $\min \ \mathrm{dom}(x|p)$
- one which participates in the maximum number of constraints : $\max \ \mathrm{degree}(x)$
- Combination of two criteria : $\min \frac{\mathrm{dom}(x|p)}{\mathrm{degree}(x)}$
- Size of domain after propagation :

$$\min \sum_{a \in \mathrm{dom}(x)} \sum_{y} \mathrm{dom}(y|p \cup \{x = a\})$$

- Depending of the impact :
  - We store the impact of the domain reduction for each variable in the course of the algorithm
  - We choose a variables with the largest impact until now

Dynamic order of values

Order of values is less important than the order of variables (less impact on the solution time)

Principle

We choose a value which has the largest probability to « succeed »

Choice of a value

- one which has the largest number of supports ;
- one which leaves the maximum number of values in the domains of other variables after propagation.

Dynamic order of values

Order of values is less important than the order of variables (less impact on the solution time)

Principle

We choose a value which has the largest probability to « succeed »

Choice of a value

- one which has the largest number of supports ;
- one which leaves the maximum number of values in the domains of other variables after propagation.

Dynamic order of values

Order of values is less important than the order of variables (less impact on the solution time)

Principle

We choose a value which has the largest probability to « succeed »

Choice of a value

- one which has the largest number of supports ;
- one which leaves the maximum number of values in the domains of other variables after propagation.

Choice of local consistency

- Local consistency should be profitable (we should spent less time to detect that a branch does not lead to any solution than to explore this branch).
- We use local consistency which has the best ratio

$$\frac{\text{cost}}{\text{propagation power}}$$

- So, for the problem in question one needs to have an idea of
  - the propagation power of different local consistencies (average number of removed values after propagation),
  - the cost of different local consistencies (practical computational complexity).
- To have an estimation of this, we often do experimental comparison.

Choice of local consistency

- Local consistency should be profitable (we should spent less time to detect that a branch does not lead to any solution than to explore this branch).
- We use local consistency which has the best ratio

$$\frac{\text{cost}}{\text{propagation power}}$$

- So, for the problem in question one needs to have an idea of
  - the propagation power of different local consistencies (average number of removed values after propagation),
  - the cost of different local consistencies (practical computational complexity).
- To have an estimation of this, we often do experimental comparison.

Choice of local consistency

- Local consistency should be profitable (we should spent less time to detect that a branch does not lead to any solution than to explore this branch).
- We use local consistency which has the best ratio

$$\frac{\text{cost}}{\text{propagation power}}$$

- So, for the problem in question one needs to have an idea of
  - the propagation power of different local consistencies (average number of removed values after propagation),
  - the cost of different local consistencies (practical computational complexity).
- To have an estimation of this, we often do experimental comparison.

Choice of local consistency

- Local consistency should be profitable (we should spent less time to detect that a branch does not lead to any solution than to explore this branch).
- We use local consistency which has the best ratio

$$\frac{\text{cost}}{\text{propagation power}}$$

- So, for the problem in question one needs to have an idea of
  - the propagation power of different local consistencies (average number of removed values after propagation),
  - the cost of different local consistencies (practical computational complexity).
- To have an estimation of this, we often do experimental comparison.