An optimal Arc Consistency algorithm for a Chain of Atmost Constraints with Cardinality

Mohamed Siala

LAAS-CNRS

Toulouse, France

Outline

CSP

A constraint satisfaction problem (CSP) is a triplet $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ where

- $\mathcal{X}$ is a set of variables.
- $\mathcal{D}$ is the related sets of values.
- $\mathcal{C}$ is a set of constraints.

A solution of a CSP is an assignment w satisfying all the constraints.

CSP

A constraint satisfaction problem (CSP) is a triplet $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ where

- $\mathcal{X}$ is a set of variables.
- $\mathcal{D}$ is the related sets of values.
- $\mathcal{C}$ is a set of constraints.

A solution of a CSP is an assignment w satisfying all the constraints.

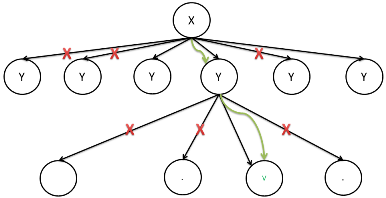
Example

- $X =< x, y >$
- $D =< \{1, 2, 3, 4\}, \{3, 4, 7, 10\} >$
- $C_1 = \{x \text{ is even}\}$
- $C_2 = \{x + y \leq 7\}$
- →A possible solution: $< x = 2, y = 4 >$

Propagation

- A propagator (or filtering algorithm) aims to remove some values that are inconsistent.
- Correctness & Checking

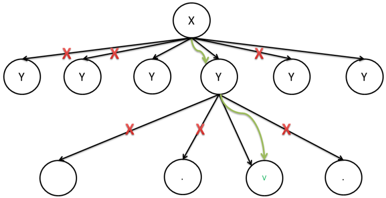
Figure: Propagation impact



Propagation

- A propagator (or filtering algorithm) aims to remove some values that are inconsistent.
- Correctness & Checking

Figure: Propagation impact



- An arc consistency algorithm is a complete filtering algorithm, i.e. when associated to a constraint C, it removes all the inconsistent values.

Global constraints

- A global constraint is constraint with unfixed arity ($n > 2$).
- For instance, the AllDifferent($x_1, x_2..x_n$) constraint ensures that the all variables, $x1$ to xn, have different values.
- More than 360 constraints in the literature (see the global constraint catalog http://www.emn.fr/z-info/sdemasse/gccat/)
- A global constraint captures a sub-problem.
- A global constraint can be used to resolve different problems.
- A global constraint $\leftrightarrow$ spatial propagator.

Propagation & Global Constraints

AllDifferent(X, Y, Z)

$X, Y, Z,\ D_X = D_Y = D_Z = \{1, 2\}$

Decomposition	Global Constraint
$C_1 : X \neq Y; C_2 : Y \neq Z; C_3 : Z \neq X;$	AllDifferent(X, Y, Z)
Propagate(C_1) : $C_1 : X \neq Y$ $D_X = D_Y\{1, 2\}$ $\rightarrow$ No propagation! Propagate(C_2),Propagate(C_3) : No propagation	$D_X = D_Y = D_Z = \{1, 2\}$ $\rightarrow$ Failure!

Definition

$\textsc{AtMostSeqCard}(u, q, d, [x_1, \dots, x_n]) \Leftrightarrow$

$$\bigwedge_{i=0}^{n-q} \left(\sum_{l=1}^{q} x_{i+l} \leq u\right) \wedge \left(\sum_{i=1}^{n} x_i = d\right)$$

Definition

AtMostSeqCard$(u, q, d, [x_1, \ldots, x_n]) \Leftrightarrow$

$$\bigwedge_{i=0}^{n-q} \left(\sum_{l=1}^{q} x_{i+l} \leq u\right) \wedge \left(\sum_{i=1}^{n} x_i = d\right)$$

Example AtMostSeqCard$(2, 4, 4, [x_1, \ldots, x_7])$

0 1 1 0 1 1 0

1 1 0 0 1 0 1

Context

Sequence Constraints

- Let $[x_1, \ldots, x_n]$ be a sequence of integer variables.

Context

Sequence Constraints

- Let $[x_1, \ldots, x_n]$ be a sequence of integer variables.
- The Among constraint ensures that the number of occurrences of values in a given set of values $v = \{v_1..v_k\}$ in a subsequence $[x_{i_1}, \ldots, x_{i_q}]$ is bounded between l and u.
  e.g. $D(x_i) = \{0, .., 9\}$, $v = \{3, 6\}$, $l = 1$, $u = 2$,
  $Among([x_3, x_4, x_5, x_6], v, 1, 2)$:

[0,3,3,4] | [6,3,6,2]

Context

Sequence Constraints

- Let $[x_1, \ldots, x_n]$ be a sequence of integer variables.
- The Among constraint ensures that the number of occurrences of values in a given set of values $v = \{v_1..v_k\}$ in a subsequence $[x_{i_1}, \ldots, x_{i_q}]$ is bounded between l and u.
  e.g. $D(x_i) = \{0,..,9\}$, $v = \{3,6\}$, $l = 1$, $u = 2$,
  $Among([x_3, x_4, x_5, x_6], v, 1, 2)$:

  [0,3,3,4] | [6,3,6,2]

- AmongSeq : the conjunction of all $n - q + 1$ Among on q consecutive variables (i.e. $\bigwedge_{i=0}^{n-q} \textsc{Among}([x_{i+1}, \ldots, x_{i+q}])$).
  e.g. $AmongSeq([x_1, x_2, x_3, x_4, x_5, x_6], 4, v, l, u) \Leftrightarrow$
  $Among([x_1, x_2, x_3, x_4], v, l, u) \wedge$
  $Among([x_2, x_3, x_4, x_5], v, l, u) \wedge$
  $Among([x_3, x_4, x_5, x_6], v, l, u)$.

Sequence Constraints[2]

- Gen-Sequence: Conjunction of (consecutive) Among

Sequence Constraints[2]

- Gen-Sequence: Conjunction of (consecutive) Among
- Global Sequencing Constraint: AmongSeq $\oplus$ Global Cardinality Constraint

Sequence Constraints[2]

- Gen-Sequence: Conjunction of (consecutive) Among
- Global Sequencing Constraint: AmongSeq $\oplus$ Global Cardinality Constraint
- AtMostSeqCard $\equiv$ AmongSeq $\oplus$ a cardinality constraint

Sequence Constraints[2]

- Gen-Sequence: Conjunction of (consecutive) Among
- Global Sequencing Constraint: AmongSeq $\oplus$ Global Cardinality Constraint
- AtMostSeqCard $\equiv$ AmongSeq $\oplus$ a cardinality constraint

$\rightarrow$ AtMostSeqCard can be encoded with a Gen-Sequence

Sequence Constraints[2]

- Gen-Sequence: Conjunction of (consecutive) Among
- Global Sequencing Constraint: AmongSeq ⊕ Global Cardinality Constraint
- AtMostSeqCard ≡ AmongSeq ⊕ a cardinality constraint

→ AtMostSeqCard can be encoded with a Gen-Sequence

→ AtMostSeqCard can be encoded with a Global Sequencing Constraint (Gsc)

Existing complexities

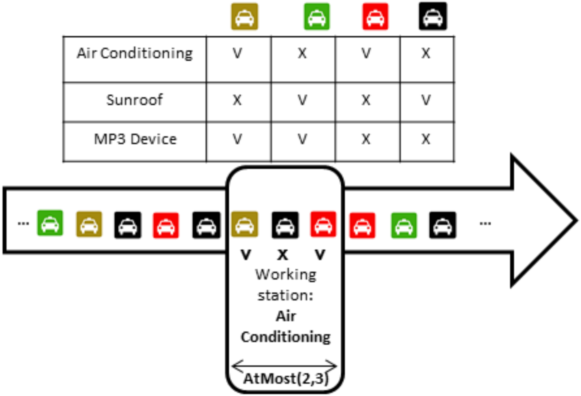
Gen-Sequence

- cost-Regular encoding: $O(2^q n)$ [Van Hoeve et al, 2009]
- Gen-Sequence: $O(n^3)$ [Van Hoeve et al, 2009]
- Flow-based Algorithm: $O(n^2)$ [Maher et al, 2008]

Gsc

- Gcc encoding, Not AC, NP-Hard [Puget and Régin, 1997]

Why the AtMostSeqCard constraint? [1]



Air Conditioning	V	X	V	X
Sunroof	X	V	X	V
MP3 Device	V	V	X	X

Figure: The car-sequencing problem

Why the AtMostSeqCard constraint? [2]

7 days, 4 employees, 3 periods, 40h per week, Atmost(1,3)

	D	E	N	D	E	N	D	E	N	D	E	N	D	E	N	D	E	N	D	E	N	d
emp_1	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	1	0	0	5
emp_2	1	0	0	1	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	0	0	5
emp_3	0	0	0	1	0	0	1	0	0	0	0	1	0	0	0	0	0	1	0	0	1	5
emp_4	0	1	0	0	0	1	0	0	1	0	0	0	1	0	0	1	0	0	0	0	0	5

Table: Crew-rostering problem

The proposed algorithm

- Let $(x_1, \ldots, x_n)$ be a boolean sequence subject to $\textsc{AtMostSeqCard}(u, q, d, [x_1, \ldots, x_n])$
- Suppose that $n = 50$, $d = 17$, and we have : $D\{x_i\} = \{0, 1\}$:

1	..	$i-1$	i	$i+1$ ..	n
0	..	1	$D\{x_i\} = \{0,1\}$	0 ..	$\{0,1\}$

- If $Max1Left_{i-1} = 7$, $Max1Right_{i+1} = 9$, $\Longrightarrow$ We have to force the variable x_i to have the value 1 in order to satisfy the cardinality.
- If $Max0Left_{i-1} = 10$, $Max0Right_{i+1} = 23$, $\Longrightarrow$ We have to force the variable x_i to have the value 0 in order to satisfy the occurrences of 0 (i.e. $n - d = 33$).

$\overrightarrow{w}$ = leftmost ($u = 2$, $q = 4$)

x_i	w	c 1	2	3	4	max
.	0					
0	0					
.	0					
1	1					
.	0					
.	0					
.	0					
0	0					
.	0					
0	0					
1	1					
.	0					
.	0					
1	1					
.	0					
.	0					

$\overrightarrow{w}$ = leftmost $(u = 2,\ q = 4)$

	x_i		w	c 1	2	3	4	max
→	.	—	0	**0**				
	0		0					
	.		0					
	1		1					
	.		0					
	.		0					
	.		0					
	0		0					
	.		0					
	0		0					
	1		1					
	.		0					
	.		0					
	1		1					
	.		0					
	.		0					

$\overrightarrow{w}$ = leftmost $(u = 2,\ q = 4)$

	x_i		w	c 1	2	3	4	max
→	.	—	0	**0**	**0**			
	0	—	0					
	.		0					
	1		1					
	.		0					
	.		0					
	.		0					
	0		0					
	.		0					
	0		0					
	1		1					
	.		0					
	.		0					
	1		1					
	.		0					
	.		0					

$\overrightarrow{w}$ = leftmost ($u = 2$, $q = 4$)

	x_i		w	c 1	2	3	4	max
→	.	—	0	**0**	**0**	**0**		
	0	—	0					
	.	—	0					
	1		1					
	.		0					
	.		0					
	.		0					
	0		0					
	.		0					
	0		0					
	1		1					
	.		0					
	.		0					
	1		1					
	.		0					
	.		0					

$\overrightarrow{w}$ = leftmost $(u = 2,\ q = 4)$

	x_i		w	c 1	2	3	4	max
→	.	—	0	**0**	**0**	**0**	**1**	
	0	—	0					
	.	—	0					
	1	—	1					
	.		0					
	.		0					
	.		0					
	0		0					
	.		0					
	0		0					
	1		1					
	.		0					
	.		0					
	1		1					
	.		0					
	.		0					

$\overrightarrow{w}$ = leftmost $(u = 2,\ q = 4)$

x_i	w	c 1	2	3	4	max
.	0	**0**	**0**	**0**	**1**	**1**
0	0					
.	0					
1	1					
.	0					
.	0					
.	0					
0	0					
.	0					
0	0					
1	1					
.	0					
.	0					
1	1					
.	0					
.	0					

$\overrightarrow{w}$ = leftmost $(u = 2,\ q = 4)$

x_i	w	c 1	2	3	4	max
.	**1**	**0**	**0**	**0**	**1**	**1**
0	0					
.	0					
1	1					
.	0					
.	0					
.	0					
0	0					
.	0					
0	0					
1	1					
.	0					
.	0					
1	1					
.	0					
.	0					

$\overrightarrow{w}$ = leftmost ($u = 2,\ q = 4$)

	x_i		w	c				max
				1	2	3	4	
	.	—	**1**	**0**	**0**	**0**	**1**	**1**
→	0	—	0	1				
	.		0					
	1		1					
	.		0					
	.		0					
	.		0					
	0		0					
	.		0					
	0		0					
	1		1					
	.		0					
	.		0					
	1		1					
	.		0					
	.		0					

$\overrightarrow{w}$ = leftmost $(u = 2,\ q = 4)$

	x_i		w	c 1	2	3	4	max
	.	—	**1**	**0**	**0**	**0**	**1**	**1**
→	0	—	0	1	1			
	.	—	0					
	1		1					
	.		0					
	.		0					
	.		0					
	0		0					
	.		0					
	0		0					
	1		1					
	.		0					
	.		0					
	1		1					
	.		0					
	.		0					

$\overrightarrow{w}$ = leftmost ($u = 2$, $q = 4$)

	x_i		w	c 1	2	3	4	max
	.	—	**1**	**0**	**0**	**0**	**1**	**1**
→	0	—	0	1	1	2		
	.	—	0					
	1	—	1					
	.		0					
	.		0					
	.		0					
	0		0					
	.		0					
	0		0					
	1		1					
	.		0					
	.		0					
	1		1					
	.		0					
	.		0					

$\overrightarrow{w}$ = leftmost ($u = 2$, $q = 4$)

	x_i		w	c 1	2	3	4	max
	.		**1**	**0**	**0**	**0**	**1**	**1**
→	0	—	0	1	1	2	1	
	.	—	0					
	1	—	1					
	.	—	0					
	.		0					
	.		0					
	0		0					
	.		0					
	0		0					
	1		1					
	.		0					
	.		0					
	1		1					
	.		0					
	.		0					

$\overrightarrow{w}$ = leftmost $(u = 2,\ q = 4)$

x_i	w	c 1	2	3	4	max
.	**1**	**0**	**0**	**0**	**1**	**1**
0	0	1	1	2	1	2
.	0					
1	1					
.	0					
.	0					
.	0					
0	0					
.	0					
0	0					
1	1					
.	0					
.	0					
1	1					
.	0					
.	0					

$\overrightarrow{w} = \texttt{leftmost}\ (u = 2,\ q = 4)$

x_i	w	c 1	2	3	4	max
.	**1**	**0**	**0**	**0**	**1**	**1**
0	0	1	1	2	1	2
.	0					
1	1					
.	0					
.	0					
.	0					
0	0					
.	0					
0	0					
1	1					
.	0					
.	0					
1	1					
.	0					
.	0					

$\overrightarrow{w}$ = leftmost ($u = 2$, $q = 4$)

	x_i		w	c 1	2	3	4	max
	.	—	**1**	**0**	**0**	**0**	**1**	**1**
	0	—	0	1	1	2	1	2
→	.	—	0	1				
	1		1					
	.		0					
	.		0					
	.		0					
	0		0					
	.		0					
	0		0					
	1		1					
	.		0					
	.		0					
	1		1					
	.		0					
	.		0					

$\overrightarrow{w}$ = leftmost $(u = 2,\ q = 4)$

	x_i		w	c 1	2	3	4	max
	.	—	**1**	**0**	**0**	**0**	**1**	**1**
	0	—	0	1	1	2	1	2
→	.	—	0	1	2			
	1	—	1					
	.		0					
	.		0					
	.		0					
	0		0					
	.		0					
	0		0					
	1		1					
	.		0					
	.		0					
	1		1					
	.		0					
	.		0					

$\overrightarrow{w}$ = leftmost ($u = 2$, $q = 4$)

	x_i		w	c 1	2	3	4	max
	.		**1**	**0**	**0**	**0**	**1**	**1**
	0	—	0	1	1	2	1	2
→	.	—	0	1	2	1		
	1	—	1					
	.	—	0					
	.		0					
	.		0					
	0		0					
	.		0					
	0		0					
	1		1					
	.		0					
	.		0					
	1		1					
	.		0					
	.		0					

$\overrightarrow{w}$ = leftmost $(u = 2,\ q = 4)$

	x_i		w	c 1	2	3	4	max
	.		**1**	**0**	**0**	**0**	**1**	**1**
	0		0	1	1	2	1	2
→	.	—	0	1	2	1	1	
	1	—	1					
	.	—	0					
	.	—	0					
	.		0					
	0		0					
	.		0					
	0		0					
	1		1					
	.		0					
	.		0					
	1		1					
	.		0					
	.		0					

$\overrightarrow{w}$ = leftmost $(u = 2,\ q = 4)$

x_i	w	c 1	2	3	4	max
.	**1**	**0**	**0**	**0**	**1**	**1**
0	0	1	1	2	1	2
.	0	1	2	1	1	2
1	1					
.	0					
.	0					
.	0					
0	0					
.	0					
0	0					
1	1					
.	0					
.	0					
1	1					
.	0					
.	0					

$\overrightarrow{w}$ = leftmost $(u = 2,\ q = 4)$

x_i	w	c 1	2	3	4	max
.	**1**	**0**	**0**	**0**	**1**	**1**
0	0	1	1	2	1	2
.	0	1	2	1	1	2
1	1					
.	0					
.	0					
.	0					
0	0					
.	0					
0	0					
1	1					
.	0					
.	0					
1	1					
.	0					
.	0					

$\overrightarrow{w}$ = leftmost ($u = 2$, $q = 4$)

	x_i		w	c 1	2	3	4	max
	.	—	**1**	**0**	**0**	**0**	**1**	**1**
	0	—	0	1	1	2	1	2
	.	—	0	1	2	1	1	2
→	1	—	1	2				
	.		0					
	.		0					
	.		0					
	0		0					
	.		0					
	0		0					
	1		1					
	.		0					
	.		0					
	1		1					
	.		0					
	.		0					

$\overrightarrow{w}$ = leftmost $(u = 2,\ q = 4)$

	x_i		w	c 1	2	3	4	max
	.		**1**	**0**	**0**	**0**	**1**	**1**
	0	—	0	1	1	2	1	2
	.	—	0	1	2	1	1	2
→	1	—	1	2	1			
	.	—	0					
	.		0					
	.		0					
	0		0					
	.		0					
	0		0					
	1		1					
	.		0					
	.		0					
	1		1					
	.		0					
	.		0					

$\overrightarrow{w}$ = leftmost $(u = 2,\ q = 4)$

	x_i		w	c 1	2	3	4	max
	.		**1**	**0**	**0**	**0**	**1**	**1**
	0		0	1	1	2	1	2
	.	—	0	1	2	1	1	2
→	1	—	1	2	1	1		
	.	—	0					
	.	—	0					
	.		0					
	0		0					
	.		0					
	0		0					
	1		1					
	.		0					
	.		0					
	1		1					
	.		0					
	.		0					

$\overrightarrow{w}$ = leftmost ($u = 2$, $q = 4$)

	x_i		w	c 1	2	3	4	max
	.		**1**	**0**	**0**	**0**	**1**	**1**
	0		0	1	1	2	1	2
	.		0	1	2	1	1	2
→	1	—	1	2	1	1	1	
	.	—	0					
	.	—	0					
	.	—	0					
	0		0					
	.		0					
	0		0					
	1		1					
	.		0					
	.		0					
	1		1					
	.		0					
	.		0					

$\overrightarrow{w}$ = leftmost $(u = 2,\ q = 4)$

x_i	w	c 1	2	3	4	max
.	**1**	**0**	**0**	**0**	**1**	**1**
0	0	1	1	2	1	2
.	0	1	2	1	1	2
1	1	2	1	1	1	2
.	0					
.	0					
.	0					
0	0					
.	0					
0	0					
1	1					
.	0					
.	0					
1	1					
.	0					
.	0					

$\overrightarrow{w}$ = leftmost ($u = 2$, $q = 4$)

x_i	w	c 1	2	3	4	max
.	**1**	**0**	**0**	**0**	**1**	**1**
0	0	1	1	2	1	2
.	0	1	2	1	1	2
1	1	2	1	1	1	2
.	0					
.	0					
.	0					
0	0					
.	0					
0	0					
1	1					
.	0					
.	0					
1	1					
.	0					
.	0					

$\overrightarrow{w}$ = leftmost $(u = 2,\ q = 4)$

	x_i		w	c 1	2	3	4	max
	.		**1**	**0**	**0**	**0**	**1**	**1**
	0	—	0	1	1	2	1	2
	.	—	0	1	2	1	1	2
	1	—	1	2	1	1	1	2
→	.	—	0	**1**				
	.		0					
	.		0					
	0		0					
	.		0					
	0		0					
	1		1					
	.		0					
	.		0					
	1		1					
	.		0					
	.		0					

$\overrightarrow{w}$ = leftmost $(u = 2,\ q = 4)$

	x_i		w	c				max
				1	2	3	4	
	.		**1**	**0**	**0**	**0**	**1**	**1**
	0		0	1	1	2	1	2
	.	—	0	1	2	1	1	2
	1	—	1	2	1	1	1	2
→	.	—	0	**1**	**1**			
	.	—	0					
	.		0					
	0		0					
	.		0					
	0		0					
	1		1					
	.		0					
	.		0					
	1		1					
	.		0					
	.		0					

$\overrightarrow{w}$ = leftmost ($u = 2$, $q = 4$)

	x_i		w	c 1	2	3	4	max
	.		**1**	**0**	**0**	**0**	**1**	**1**
	0		0	1	1	2	1	2
	.		0	1	2	1	1	2
	1	—	1	2	1	1	1	2
→	.	—	0	**1**	**1**	**1**		
	.	—	0					
	.	—	0					
	0		0					
	.		0					
	0		0					
	1		1					
	.		0					
	.		0					
	1		1					
	.		0					
	.		0					

$\overrightarrow{w}$ = leftmost $(u = 2,\ q = 4)$

	x_i		w	c 1	2	3	4	max
	.		**1**	**0**	**0**	**0**	**1**	**1**
	0		0	1	1	2	1	2
	.		0	1	2	1	1	2
	1		1	2	1	1	1	2
→	.	—	0	**1**	**1**	**1**	**0**	
	.	—	0					
	.	—	0					
	0	—	0					
	.		0					
	0		0					
	1		1					
	.		0					
	.		0					
	1		1					
	.		0					
	.		0					

$\overrightarrow{w}$ = leftmost ($u = 2$, $q = 4$)

x_i	w	c 1	2	3	4	max
.	**1**	**0**	**0**	**0**	**1**	**1**
0	0	1	1	2	1	2
.	0	1	2	1	1	2
1	1	2	1	1	1	2
.	0	**1**	**1**	**1**	**0**	**1**
.	0					
.	0					
0	0					
.	0					
0	0					
1	1					
.	0					
.	0					
1	1					
.	0					
.	0					

$\overrightarrow{w}$ = leftmost ($u = 2$, $q = 4$)

x_i	w	c 1	2	3	4	max
.	**1**	**0**	**0**	**0**	**1**	**1**
0	0	1	1	2	1	2
.	0	1	2	1	1	2
1	1	2	1	1	1	2
.	**1**	**1**	**1**	**1**	**0**	**1**
.	0					
.	0					
0	0					
.	0					
0	0					
1	1					
.	0					
.	0					
1	1					
.	0					
.	0					

$\overrightarrow{w}$ = $\texttt{leftmost}$ $(u = 2,\ q = 4)$

	x_i		w	c: 1	2	3	4	max
	.		**1**	**0**	**0**	**0**	**1**	**1**
	0		0	1	1	2	1	2
	.	—	0	1	2	1	1	2
	1	—	1	2	1	1	1	2
	.	—	**1**	**1**	**1**	**1**	**0**	**1**
→	.	—	0	2				
	.		0					
	0		0					
	.		0					
	0		0					
	1		1					
	.		0					
	.		0					
	1		1					
	.		0					
	.		0					

$\overrightarrow{w} = \texttt{leftmost}\ (u = 2,\ q = 4)$

	x_i		w	c				max
				1	2	3	4	
	.		**1**	**0**	**0**	**0**	**1**	**1**
	0		0	1	1	2	1	2
	.		0	1	2	1	1	2
	1	—	1	2	1	1	1	2
	.	—	**1**	**1**	**1**	**1**	**0**	**1**
→	.	—	0	2	2			
	.	—	0					
	0		0					
	.		0					
	0		0					
	1		1					
	.		0					
	.		0					
	1		1					
	.		0					
	.		0					

$\overrightarrow{w}$ = leftmost $(u = 2,\ q = 4)$

	x_i		w	c 1	2	3	4	max
	.		**1**	**0**	**0**	**0**	**1**	**1**
	0		0	1	1	2	1	2
	.		0	1	2	1	1	2
	1		1	2	1	1	1	2
	.	—	**1**	**1**	**1**	**1**	**0**	**1**
→	.	—	0	2	2	1		
	.	—	0					
	0	—	0					
	.		0					
	0		0					
	1		1					
	.		0					
	.		0					
	1		1					
	.		0					
	.		0					

$\overrightarrow{w} = \texttt{leftmost}\ (u = 2,\ q = 4)$

	x_i		w	c 1	2	3	4	max
	.		**1**	**0**	**0**	**0**	**1**	**1**
	0		0	1	1	2	1	2
	.		0	1	2	1	1	2
	1		1	2	1	1	1	2
	.		**1**	**1**	**1**	**1**	**0**	**1**
→	.	—	0	2	2	1	0	
	.	—	0					
	0	—	0					
	.	—	0					
	0		0					
	1		1					
	.		0					
	.		0					
	1		1					
	.		0					
	.		0					

$\overrightarrow{w}$ = leftmost ($u = 2$, $q = 4$)

x_i	w	c 1	2	3	4	max
.	**1**	**0**	**0**	**0**	**1**	**1**
0	0	1	1	2	1	2
.	0	1	2	1	1	2
1	1	2	1	1	1	2
.	**1**	**1**	**1**	**1**	**0**	**1**
.	0	2	2	1	0	2
.	0					
0	0					
.	0					
0	0					
1	1					
.	0					
.	0					
1	1					
.	0					
.	0					

$\overrightarrow{w}$ = leftmost $(u = 2,\ q = 4)$

x_i	w	c 1	2	3	4	max
.	**1**	**0**	**0**	**0**	**1**	**1**
0	0	1	1	2	1	2
.	0	1	2	1	1	2
1	1	2	1	1	1	2
.	**1**	**1**	**1**	**1**	**0**	**1**
.	0	2	2	1	0	2
.	0					
0	0					
.	0					
0	0					
1	1					
.	0					
.	0					
1	1					
.	0					
.	0					

$\overrightarrow{w}$ = leftmost ($u = 2$, $q = 4$)

x_i	w	c 1	2	3	4	max
.	**1**	**0**	**0**	**0**	**1**	**1**
0	0	1	1	2	1	2
.	0	1	2	1	1	2
1	1	2	1	1	1	2
.	**1**	**1**	**1**	**1**	**0**	**1**
.	0	2	2	1	0	2
.	0	2	1	0	0	2
0	0	1	0	0	1	1
.	**1**	**0**	**0**	**1**	**1**	**1**
0	0	0	2	2	1	2
1	1	2	2	1	2	2
.	0	2	1	2	1	2
.	0	1	2	1	1	2
1	1	2	1	1	1	2
.	**1**	**1**	**1**	**1**	**0**	**1**
.	0	2	2	1	0	2

$\overrightarrow{w}$ = leftmost $(u = 2,\ q = 4)$

x_i	w	c 1	2	3	4	max
.	**1**	**0**	**0**	**0**	**1**	**1**
0	0	1	1	2	1	2
.	0	1	2	1	1	2
1	1	2	1	1	1	2
.	**1**	**1**	**1**	**1**	**0**	**1**
.	0	2	2	1	0	2
.	0	2	1	0	0	2
0	0	1	0	0	1	1
.	**1**	**0**	**0**	**1**	**1**	**1**
0	0	0	2	2	1	2
1	1	2	2	1	2	2
.	0	2	1	2	1	2
.	0	1	2	1	1	2
1	1	2	1	1	1	2
.	**1**	**1**	**1**	**1**	**0**	**1**
.	0	2	2	1	0	2

$\rightarrow$ Complexity $= O(n.q)$

leftmost_count

- leftmost_count$([x_1, \ldots, x_n], u, q, d)$: a linear time implementation of leftmost but returning the maximum cardinality that we can add to the sequence until i.

leftmost_count

- `leftmost_count`$([x_1, \ldots, x_n], u, q, d)$: a linear time implementation of `leftmost` but returning the maximum cardinality that we can add to the sequence until i.
- Example:

$\mathcal{D}(x_i)$		.	0	.	.	.	.	.	.	0	1	0	.	.	.	.	.	.	.	.	.	.	1
leftmost$[i]$		**1**	0	**1**	**1**	**1**	0	0	0	0	1	0	**1**	**1**	**1**	0	0	0	**1**	0	**1**	**1**	1
leftmost_count$[i]$	0	1	1	2	3	4	4	4	4	4	4	4	5	6	7	7	7	7	8	8	9	10	10

leftmost_count

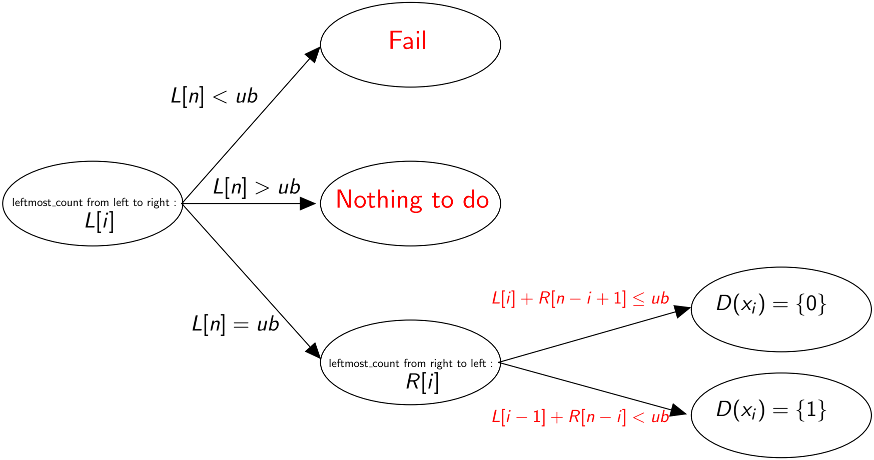
- leftmost_count$([x_1, \ldots, x_n], u, q, d)$: a linear time implementation of leftmost but returning the maximum cardinality that we can add to the sequence until i.
- Example:

$\mathcal{D}(x_i)$		.	0	.	.	.	.	.	.	0	1	0	.	.	.	.	.	.	.	.	.	.	1
leftmost[i]		**1**	0	**1**	**1**	**1**	0	0	0	0	1	0	**1**	**1**	**1**	0	0	0	**1**	0	**1**	**1**	1
leftmost_count[i]	0	1	1	2	3	4	4	4	4	4	4	4	5	6	7	7	7	7	8	8	9	10	10

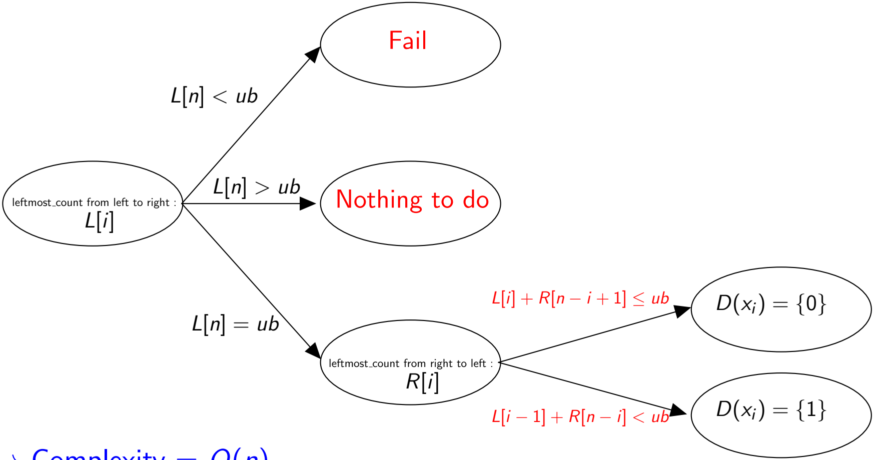
- L (resp. R): the result of leftmost_count from left to right (resp. right to left).

The Arc consistency algorithm

The Arc consistency algorithm



The Arc consistency algorithm



→ Complexity $= O(n)$

AC($u = 4, q = 8, d = 12, ub = 10$)

$\mathcal{D}(x_i)$	.	0	.	.	.	.	.	.	0	1	0	.	.	.	.	.	.	.	.	.	1

AC($u = 4, q = 8, d = 12, ub = 10$)

$\mathcal{D}(x_i)$	.	0	.	.	.	.	.	.	0	1	0	.	.	.	.	.	.	.	.	.	.	1
$\overrightarrow{w}[i]$	**1**	0	**1**	**1**	**1**	0	0	0	0	1	0	**1**	**1**	**1**	0	0	0	**1**	0	**1**	**1**	1

AC($u = 4, q = 8, d = 12, ub = 10$)

$\mathcal{D}(x_i)$	.	0	.	.	.	.	.	.	0	1	0	.	.	.	.	.	.	.	.	.	.	1
$\overrightarrow{w}[i]$	**1**	0	**1**	**1**	**1**	0	0	0	0	1	0	**1**	**1**	**1**	0	0	0	**1**	0	**1**	**1**	1
$\overleftarrow{w}[i]$	**1**	0	0	**1**	**1**	**1**	0	0	0	1	0	**1**	**1**	**1**	0	0	0	0	**1**	**1**	**1**	1

$AC(u = 4, q = 8, d = 12, ub = 10)$

$\mathcal{D}(x_i)$		.	0	.	.	.	.	.	.	0	1	0	.	.	.	.	.	.	.	.	.	.	1
$\overrightarrow{w}[i]$		**1**	0	**1**	**1**	**1**	0	0	0	0	1	0	**1**	**1**	**1**	0	0	0	**1**	0	**1**	**1**	1
$\overleftarrow{w}[i]$		**1**	0	0	**1**	**1**	**1**	0	0	0	1	0	**1**	**1**	**1**	0	0	0	0	**1**	**1**	**1**	1
$L[i]$	0	1	1	2	3	4	4	**4**	4	4	4	**4**	5	6	7	7	7	7	8	8	9	10	10

AC($u = 4, q = 8, d = 12, ub = 10$)

$\mathcal{D}(x_i)$		.	0	.	.	.	.	.	.	0	1	0	.	.	.	.	.	.	.	.	.	.	1	
$\overrightarrow{w}[i]$		**1**	0	**1**	**1**	**1**	0	0	0	0	1	0	**1**	**1**	**1**	0	0	0	**1**	0	**1**	**1**	1	
$\overleftarrow{w}[i]$		**1**	0	0	**1**	**1**	**1**	0	0	0	1	0	**1**	**1**	**1**	0	0	0	0	**1**	**1**	**1**	1	
$L[i]$	0	1	1	2	3	4	4	**4**	4	4	4	**4**	5	6	7	7	7	7	8	8	9	10	10	
$R[n-i+1]$		10	9	9	9	8	7	**6**	6	6	6	6	6	**5**	4	3	3	3	3	3	2	1	0	0

AC($u = 4, q = 8, d = 12, ub = 10$)

$\mathcal{D}(x_i)$		.	0	.	.	.	.	.	.	0	1	0	.	.	.	.	.	.	.	.	.	.	1	
$\overrightarrow{w}[i]$		**1**	0	**1**	**1**	**1**	0	0	0	0	1	0	**1**	**1**	**1**	0	0	0	**1**	0	**1**	**1**	1	
$\overleftarrow{w}[i]$		**1**	0	0	**1**	**1**	**1**	0	0	0	1	0	**1**	**1**	**1**	0	0	0	0	**1**	**1**	**1**	1	
$L[i]$	0	1	1	2	3	4	4	**4**	4	4	4	**4**	5	6	7	7	7	7	8	8	9	10	10	
$R[n-i+1]$		10	9	9	9	8	7	**6**	6	6	6	6	6	**5**	4	3	3	3	3	3	2	1	0	0
$L[i] + R[n-i+1]$		11	10	11	12	12	11	**10**	**10**	10	10	10	11	11	11	**10**	**10**	**10**	11	11	11	11	10	

AC($u = 4, q = 8, d = 12, ub = 10$)

$\mathcal{D}(x_i)$		.	0	.	.	.	.	.	.	0	1	0	.	.	.	.	.	.	.	.	.	.	1	
$\overrightarrow{w}[i]$		**1**	0	**1**	**1**	**1**	0	0	0	0	1	0	**1**	**1**	**1**	0	0	0	**1**	0	**1**	**1**	1	
$\overleftarrow{w}[i]$		**1**	0	0	**1**	**1**	**1**	0	0	0	1	0	**1**	**1**	**1**	0	0	0	0	**1**	**1**	**1**	1	
$L[i]$	0	1	1	2	3	4	4	**4**	4	4	4	**4**	5	6	7	7	7	7	8	8	9	10	10	
$R[n-i+1]$		10	9	9	9	8	7	**6**	6	6	6	6	6	**5**	4	3	3	3	3	3	2	1	0	0
$L[i] + R[n-i+1]$		11	10	11	12	12	11	**10**	**10**	10	10	10	11	11	11	**10**	**10**	**10**	11	11	11	11	10	
$L[i-1] + R[n-i]$		**9**	10	10	10	10	10	10	10	10	10	10	**9**	**9**	**9**	10	10	10	10	10	**9**	**9**	10	

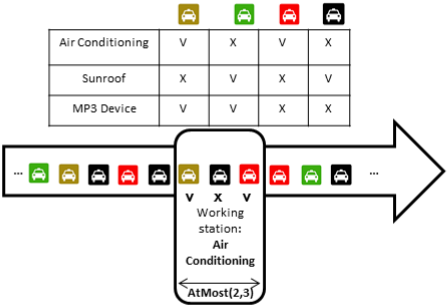
$\mathrm{AC}(u = 4, q = 8, d = 12, ub = 10)$

$\mathcal{D}(x_i)$		.	0	.	.	.	.	.	.	0	1	0	.	.	.	.	.	.	.	.	.	.	1	
$\overrightarrow{w}[i]$		**1**	0	**1**	**1**	**1**	0	0	0	0	1	0	**1**	**1**	**1**	0	0	0	**1**	0	**1**	**1**	1	
$\overleftarrow{w}[i]$		**1**	0	0	**1**	**1**	**1**	0	0	0	1	0	**1**	**1**	**1**	0	0	0	0	**1**	**1**	**1**	1	
$L[i]$	0	1	1	2	3	4	4	**4**	4	4	4	**4**	5	6	7	7	7	7	8	8	9	10	10	
$R[n-i+1]$		10	9	9	9	8	7	**6**	6	6	6	6	6	**5**	4	3	3	3	3	3	2	1	0	0
$L[i]+R[n-i+1]$		11	10	11	12	12	11	**10**	**10**	10	10	10	11	11	11	**10**	**10**	**10**	11	11	11	11	10	
$L[i-1]+R[n-i]$		**9**	10	10	10	10	10	10	10	10	10	10	**9**	**9**	**9**	10	10	10	10	10	**9**	**9**	10	
$\mathrm{AC}(\mathcal{D}(x_i))$		**1**	0	.	.	.	.	**0**	**0**	0	1	0	**1**	**1**	**1**	**0**	**0**	**0**	.	.	**1**	**1**	1	

$AC(u = 4, q = 8, d = 12, ub = 10)$

$\mathcal{D}(x_i)$		.	0	.	.	.	.	.	.	0	1	0	.	.	.	.	.	.	.	.	.	.	1	
$\overrightarrow{w}[i]$		**1**	0	**1**	**1**	**1**	0	0	0	0	1	0	**1**	**1**	**1**	0	0	0	**1**	0	**1**	**1**	1	
$\overleftarrow{w}[i]$		**1**	0	0	**1**	**1**	**1**	0	0	0	1	0	**1**	**1**	**1**	0	0	0	0	**1**	**1**	**1**	1	
$L[i]$	0	1	1	2	3	4	4	**4**	4	4	4	**4**	5	6	7	7	7	7	8	8	9	10	10	
$R[n-i+1]$		10	9	9	9	8	7	**6**	6	6	6	6	6	**5**	4	3	3	3	3	3	2	1	0	0
$L[i] + R[n-i+1]$		11	10	11	12	12	11	**10**	**10**	10	10	10	11	11	11	**10**	**10**	**10**	11	11	11	11	10	
$L[i-1] + R[n-i]$		**9**	10	10	10	10	10	10	10	10	10	10	**9**	**9**	**9**	10	10	10	10	10	**9**	**9**	10	
$AC(\mathcal{D}(x_i))$		**1**	0	.	.	.	.	**0**	**0**	0	1	0	**1**	**1**	**1**	**0**	**0**	**0**	.	.	**1**	**1**	1	

Car-sequencing



Constraints

- Each class c is associated with a demand D_c.
- For each option j, each sub-sequence of size q_j must contain at most u_j cars requiring the option j.

Models

1. sum
2. gsc
3. amsc
4. amcs + gsc

Heuristics

$\langle\{lex, mid\}, \{class, opt\}, \{1, q/u, d, \delta, n-\sigma, \rho\}, \{\leq_{\sum}, \leq_{Euc}, \leq_{lex}\}\rangle$.
$\rightarrow$ 34 heuristics x 5 randomized tests.

Benchmarks (CSP Lib)

- Groupe 1: 70 satisfiable instances
- Groupe 2: 4 satisfiable instances
- Groupe 3: 5 unsatisfiable instances
- Groupe 4: 7 satisfiable instances

Experimental results

Table: Experimental results : Car-sequencing

Models	G1 (70 × 34 × 5) 11900		G2 (4 × 34 × 5) 680		G3 (5 × 34 × 5) 850		G4 (7 × 34 × 5) 1190	
	#sol	time	#sol	time	#sol	time	#sol	time
sum	8480	13.93	95	76.60	0	> 1200	64	43.81
gsc	11218	3.60	325	110.99	31	276.06	140	56.61
amsc	10702	4.43	**360**	**72.00**	16	8.62	**153**	**33.56**
amsc+gsc	**11243**	**3.43**	339	106.53	**32**	**285.43**	147	66.45

Experimental results

Table: Experimental results : Car-sequencing

Models	G1 ($70 \times 34 \times 5$) 11900		G2 ($4 \times 34 \times 5$) 680		G3 ($5 \times 34 \times 5$) 850		G4 ($7 \times 34 \times 5$) 1190	
	#sol	time	#sol	time	#sol	time	#sol	time
sum	8480	13.93	95	76.60	0	> 1200	64	43.81
gsc	11218	3.60	325	110.99	31	276.06	140	56.61
amsc	10702	4.43	**360**	**72.00**	16	8.62	**153**	**33.56**
amsc+gsc	**11243**	**3.43**	339	106.53	**32**	**285.43**	147	66.45

- The level of filtering obtained by enforcing AC on the AtMostSeqCard constraint is incomparable with that of the Gcc encoding of the Gsc constraint

Experimental results

Table: Experimental results : Car-sequencing

Models	G1 ($70 \times 34 \times 5$) 11900		G2 ($4 \times 34 \times 5$) 680		G3 ($5 \times 34 \times 5$) 850		G4 ($7 \times 34 \times 5$) 1190	
	#sol	time	#sol	time	#sol	time	#sol	time
sum	8480	13.93	95	76.60	0	$>$ 1200	64	43.81
gsc	11218	3.60	325	110.99	31	276.06	140	56.61
amsc	10702	4.43	**360**	**72.00**	16	8.62	**153**	**33.56**
amsc+gsc	**11243**	**3.43**	339	106.53	**32**	**285.43**	147	66.45

- The level of filtering obtained by enforcing AC on the AtMostSeqCard constraint is incomparable with that of the Gcc encoding of the Gsc constraint
- The Gsc propagator seems to save more backtracks than AtMostSeqCard.

Experimental results

Table: Experimental results : Car-sequencing

Models	G1 ($70 \times 34 \times 5$) 11900		G2 ($4 \times 34 \times 5$) 680		G3 ($5 \times 34 \times 5$) 850		G4 ($7 \times 34 \times 5$) 1190	
	#sol	time	#sol	time	#sol	time	#sol	time
sum	8480	13.93	95	76.60	0	>1200	64	43.81
gsc	11218	3.60	325	110.99	31	276.06	140	56.61
amsc	10702	4.43	**360**	**72.00**	16	8.62	**153**	**33.56**
amsc+gsc	**11243**	**3.43**	339	106.53	**32**	**285.43**	147	66.45

- The level of filtering obtained by enforcing AC on the AtMostSeqCard constraint is incomparable with that of the Gcc encoding of the Gsc constraint
- The Gsc propagator seems to save more backtracks than AtMostSeqCard.
- However, it's much slower than AtMostSeqCard (overall a factor of 12.5 on the number of nodes explored per second!)

Crew-rostering

	Week 1							W 2	W 3	W 4	d
emp_1	- - -	- - -	- - -	- - -	- - -	- - -	- - -				17
emp_2	- - -	- - -	- - -	- - -	- - -	- - -	- - -	..	..	..	17
..	- - -	- - -	- - -	- - -	- - -	- - -	- - -	..	..	..	17
emp_{20}	- - -	- - -	- - -	- - -	- - -	- - -	- - -	..	..	..	17
demande:	6;6;3	6;6;3	6;6;3	6;6;3	6;6;3	2;2;1	2;2;1	..	..	..	17*20

Constraints

- A required demand for each period.
- Each employee has to work 34 hours per week (17 shifts overall).
- Atmost 8h working shift per day.
- Atmost 5 days per week.

Models

- *sum*
- *gsc*
- *amsc*

Heuristics

- *worst employee*: $MIN(\sigma_i = n_i - \frac{21d_i}{5})$, $MIN(\sigma'_j = m_j - d^s_j)$.
- *worst shift*: $MIN(\sigma'_j = m_j - d^s_j)$, $MIN(\sigma_i = n_i - \frac{21d_i}{5})$

Benchmarks

- 281 instances with different employee unavailabilities (ranging from from 18% to 46% by increment of 0.1).
- Set 1: 126 sat instances.
- Set 2: 111 instances (mostly sat).
- Set 3: 44 instances (mostly unsat).

Experimental results

Table: Experimental results: Crew-Rostering

Benchmarks	G1 ($5 \times 2 \times 126$) 1260		G2 ($5 \times 2 \times 111$) 1110		G3 ($5 \times 2 \times 44$) 440	
	#sol	time	#sol	time	#sol	time
sum	1229	12.72	574	38.45	272	5.56
gsc	1210	29.19	579	77.78	276	24.14
amsc	**1237**	**5.82**	**670**	**31.01**	**284**	**6.22**

Experimental results

Table: Experimental results: Crew-Rostering

Benchmarks	G1 (5 × 2 × 126) 1260		G2 (5 × 2 × 111) 1110		G3 (5 × 2 × 44) 440	
	#sol	time	#sol	time	#sol	time
sum	1229	12.72	574	38.45	272	5.56
gsc	1210	29.19	579	77.78	276	24.14
amsc	**1237**	**5.82**	**670**	**31.01**	**284**	**6.22**

- By analogy with the car-sequencing, there is one class with one option for each employee since we treat boolean variables.

Experimental results

Table: Experimental results: Crew-Rostering

Benchmarks	G1 ($5 \times 2 \times 126$) 1260		G2 ($5 \times 2 \times 111$) 1110		G3 ($5 \times 2 \times 44$) 440	
	#sol	time	#sol	time	#sol	time
sum	1229	12.72	574	38.45	272	5.56
gsc	1210	29.19	579	77.78	276	24.14
amsc	**1237**	**5.82**	**670**	**31.01**	**284**	**6.22**

- By analogy with the car-sequencing, there is one class with one option for each employee since we treat boolean variables.
- The Gsc constraint here is equivalent to the AtMostSeqCard hence can not do better that our propagator.

Experimental results

Table: Experimental results: Crew-Rostering

Benchmarks	G1 (5 × 2 × 126) 1260		G2 (5 × 2 × 111) 1110		G3 (5 × 2 × 44) 440	
	#sol	time	#sol	time	#sol	time
sum	1229	12.72	574	38.45	272	5.56
gsc	1210	29.19	579	77.78	276	24.14
amsc	**1237**	**5.82**	**670**	**31.01**	**284**	**6.22**

- By analogy with the car-sequencing, there is one class with one option for each employee since we treat boolean variables.
- The Gsc constraint here is equivalent to the AtMostSeqCard hence can not do better that our propagator.
- AtMostSeqCard is much faster than the Gsc : a factor 20.4 in terms of explored nodes per second!

Contributions

- Best existing complexity: $O(n^2)$ [Maher et al, 2008].
- A complete filtering algorithm with a linear time complexity O(n).
    - Car-sequencing
    - Crew-Rostering

Future work

- Adapt the filtering rule with more general sequence constraints.
- Building a Propagator-based nogood generator for the AtMostSeqCard algorithm in a Pseudo-Boolean Solver.

Thank you!

Questions?