An optimal Arc Consistency algorithm for a chain of Atmost constraints with cardinality

Mohamed Siala , Emmanuel Hebrard, and Marie-José Huguet

LAAS-CNRS

Toulouse, France

Outline

Definition

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

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

Definition

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

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

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

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 $\{v_1..v_k\}$ in a subsequence $[x_{i1}, \ldots, x_{iq}]$ is bounded between l and u.

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 $\{v_1..v_k\}$ in a subsequence $[x_{i1}, \ldots, x_{iq}]$ is bounded between l and u.
- AMONGSEQ : the conjunction of all $n - q + 1$ AMONG on q consecutive variables (i.e. $\bigwedge_{i=0}^{n-q} \text{AMONG}([x_{i+1}, \ldots, x_{i+q}])$).

Context

Sequence Constraints

- Let $[x_1, \dots, x_n]$ be a sequence of integer variables.
- The Among constraint ensures that the number of occurrences of values in $\{v_1..v_k\}$ in a subsequence $[x_{i1}, \dots, x_{iq}]$ is bounded between l and u.
- AmongSeq : the conjunction of all $n - q + 1$ Among on q consecutive variables (i.e. $\bigwedge_{i=0}^{n-q} \text{Among}([x_{i+1}, \dots, x_{i+q}])$).
- Gen-Sequence: Conjunction of (consecutive) Among

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 $\{v_1..v_k\}$ in a subsequence $[x_{i1}, \ldots, x_{iq}]$ is bounded between l and u.
- AmongSeq : the conjunction of all $n - q + 1$ Among on q consecutive variables (i.e. $\bigwedge_{i=0}^{n-q} \text{Among}([x_{i+1}, \ldots, x_{i+q}])$).
- Gen-Sequence: Conjunction of (consecutive) Among
- AtMostSeqCard $\equiv$ AmongSeq $\oplus$ a cardinality constraint

Context

Sequence Constraints

- Let $[x_1, \dots, x_n]$ be a sequence of integer variables.
- The Among constraint ensures that the number of occurrences of values in $\{v_1..v_k\}$ in a subsequence $[x_{i1}, \dots, x_{iq}]$ is bounded between l and u.
- AmongSeq : the conjunction of all $n - q + 1$ Among on q consecutive variables (i.e. $\bigwedge_{i=0}^{n-q}$ Among$([x_{i+1}, \dots, x_{i+q}])$).
- Gen-Sequence: Conjunction of (consecutive) Among
- AtMostSeqCard $\equiv$ AmongSeq $\oplus$ a cardinality constraint

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

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 $\{v_1..v_k\}$ in a subsequence $[x_{i1}, \ldots, x_{iq}]$ is bounded between l and u.
- AmongSeq : the conjunction of all $n-q+1$ Among on q consecutive variables (i.e. $\bigwedge_{i=0}^{n-q} \text{Among}([x_{i+1}, \ldots, x_{i+q}])$).
- Gen-Sequence: Conjunction of (consecutive) Among
- AtMostSeqCard $\equiv$ AmongSeq $\oplus$ 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]

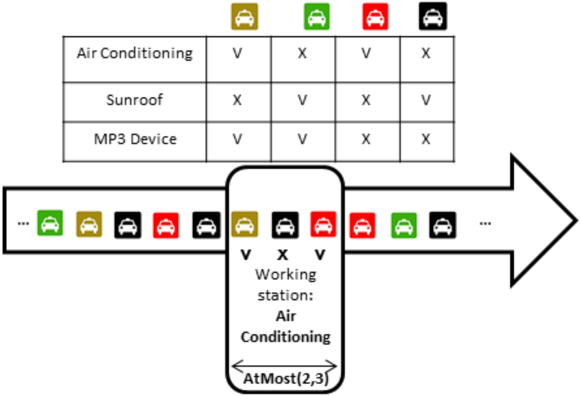


Figure: The car-sequencing problem

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 $(x_1, \dots, x_n)$ be a boolean sequence subject to $\textsc{AtMostSeqCard}(u, q, d, [x_1, \dots, x_n])$
- Our filtering algorithm is based on a greedy procedure (denoted by `leftmost`).
- `leftmost`: computes an assignment w maximizing the cardinality of the sequence with respect to the AtMost constraints.

$\overrightarrow{w}$ = 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 1	2	3	4	max
	.	—	**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					

$\overrightarrow{w}$ = 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				max
				1	2	3	4	
	.	—	**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 1	c 2	c 3	c 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**			
	.	—	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**	**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					

$\overrightarrow{w}$ = 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				
	.		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			
	.	—	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					

$\overrightarrow{w}$ = 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$([x_1, \dots, x_n], u, q, d)$: 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.
- L (resp. R): the result of leftmost_count from left to right (resp. right to left).

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

- O(n) implementation

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

- O(n) implementation

c	c[1]	c[2]	c[3]	..	c[q]
ptr	↑				

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

- O(n) implementation

c	c[1]	c[2]	c[3]	..	c[q]
ptr		↑			

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

- O(n) implementation

c	c[1]	c[2]	c[3]	..	c[q]
ptr			↑	..	

leftmost_count

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

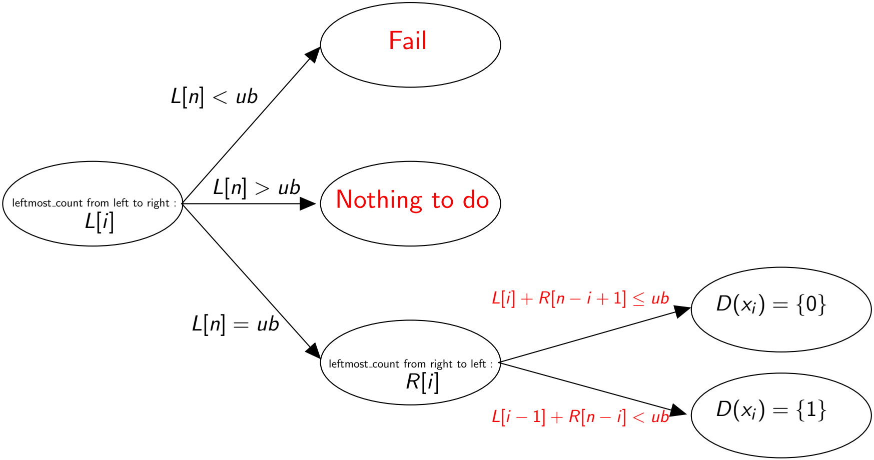
$\mathcal{D}(x_i)$		.	0	.	.	.	.	.	.	0	1	0	.	.	.	.	.	.	.	.	.	.	1
leftmost[i]		**1**	0	**1**	**1**	**1**	0	0	0	0	1	0	**1**	**1**	**1**	0	0	0	**1**	0	**1**	**1**	1
leftmost_count[i]	0	1	1	2	3	4	4	4	4	4	4	4	5	6	7	7	7	7	8	8	9	10	10

- O(n) implementation

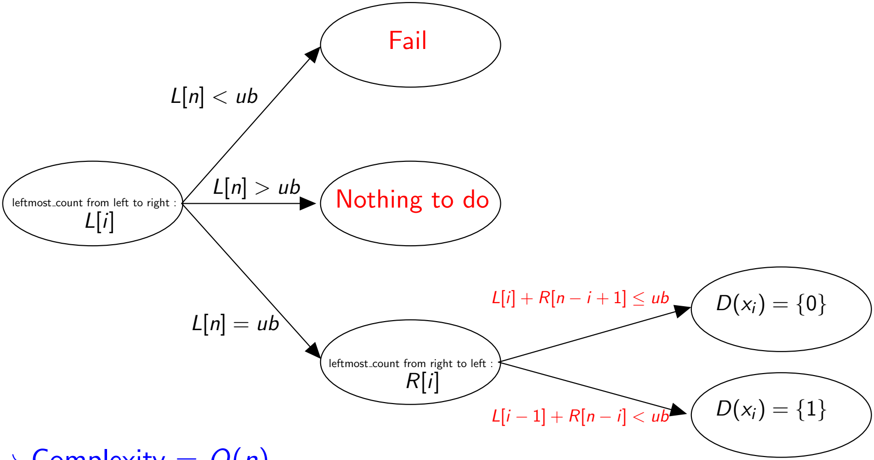
c ptr	c[1]	c[2]	c[3]	..	c[q]

The Arc consistency algorithm

The Arc consistency algorithm



The Arc consistency algorithm



$\rightarrow$ 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	

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

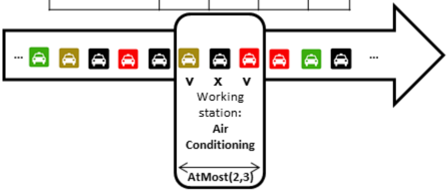
$\mathcal{D}(x_i)$		.	0	.	.	.	.	.	.	0	1	0	.	.	.	.	.	.	.	.	.	.	1	
$\overrightarrow{w}[i]$		**1**	0	**1**	**1**	**1**	0	0	0	0	1	0	**1**	**1**	**1**	0	0	0	**1**	0	**1**	**1**	1	
$\overleftarrow{w}[i]$		**1**	0	0	**1**	**1**	**1**	0	0	0	1	0	**1**	**1**	**1**	0	0	0	0	**1**	**1**	**1**	1	
$L[i]$	0	1	1	2	3	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	
$\text{AC}(\mathcal{D}(x_i))$		**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

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



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 ($70 \times 34 \times 5$) 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

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

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 × 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**

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 × 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.

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.
- 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

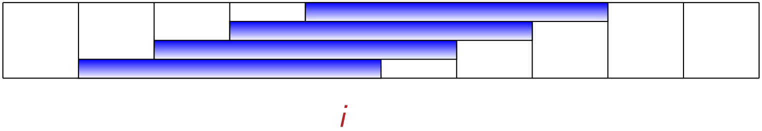
- Adapt the filtering rule with more general sequence constraints.
- Using the AtMostSeqCard algorithm and more generally filtering algorithms in a CP-based SMT-Solver.

Thank you!

Questions?

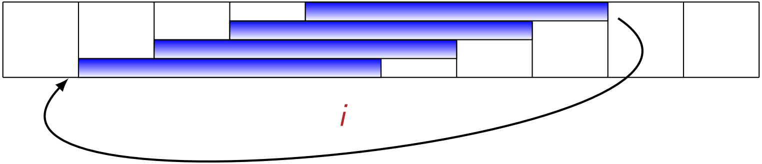
Computing the Cardinality of Subsequences

- When moving one step forward, we get *one* new subsequence (and lose another *one*)



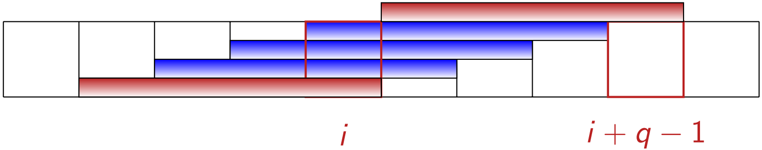
Computing the Cardinality of Subsequences

- When moving one step forward, we get *one* new subsequence (and lose another *one*)
- $i - 1 \mod q$ points to the first subsequence at step i



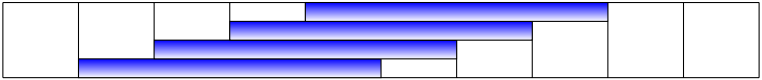
Computing the Cardinality of Subsequences

- When moving one step forward, we get *one* new subsequence (and lose another *one*)
- $i - 1 \mod q$ points to the first subsequence at step i
  - Replace $c(i-1 \mod q)$ by $c(i+q-1 \mod q) + w[i+q] - w[i]$



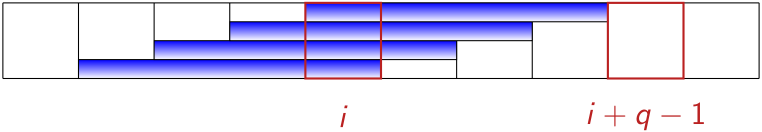
Computing the Cardinality of Subsequences

- When assigning $w[i]$ to 1, we should increment *all* subsequences
  - $O(q)$ operations



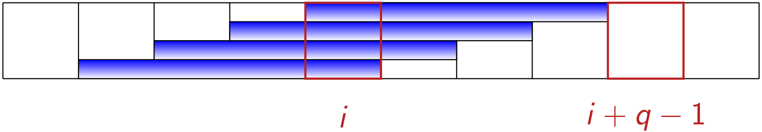
Computing the Cardinality of Subsequences

- When assigning $w[i]$ to 1, we should increment *all* subsequences
  - $O(q)$ operations
- In the previous formula: only the *negative* delta
  - $w[i+q-1]$ is equal to the minimum value in $D(x_{i+q-1})$
  - $w[i]$ might be equal to 1 because of an assignment



Computing the Cardinality of Subsequences

- When assigning $w[i]$ to 1, we should increment *all* subsequences
  - $O(q)$ operations
- In the previous formula: only the *negative* delta
  - $w[i+q-1]$ is equal to the minimum value in $D(x_{i+q-1})$
  - $w[i]$ might be equal to 1 because of an assignment
- However, the positive delta is the same for all: $\sum_{l=1}^{i}(w[l] - min(x_l))$
  - Cardinality of the j^{th} subsequence: $c[(i+j-2) \mod q] + \sum_{l=1}^{i}(w[l] - min(x_l))$

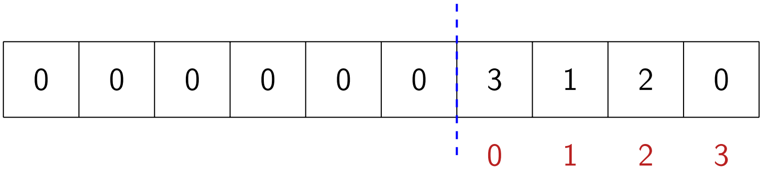


Computing the Maximum Cardinality of any Subsequence

- Computing the max, or keeping the cardinalities sorted?
  - $O(q)$ operations

Computing the Maximum Cardinality of any Subsequence

- Computing the max, or keeping the cardinalities sorted?
  - $O(q)$ operations
- We keep the *number* of subsequences of each cardinality
  - Increment all subsequences in $O(1)$



Computing the Maximum Cardinality of any Subsequence

- Computing the max, or keeping the cardinalities sorted?
  - $O(q)$ operations
- We keep the *number* of subsequences of each cardinality
  - Increment all subsequences in $O(1)$
- The maximum cardinality of any subsequence can only change by 1
  - If the number of subsequences of card $MAX(c)$ becomes 0, then $MAX(c) - 1$ is the new maximum

0	0	0	0	0	0	3	1	2	0
					0	1	2	3	