An Optimal Arc Consistency Algorithm for a Chain of Atmost Constraints with Cardinality

Mohamed Siala[1,2], Emmanuel Hebrard[1,3], and Marie-José Huguet[1,2]

[1] CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
[2] Univ de Toulouse, INSA, LAAS, F-31400 Toulouse, France
[3] Univ de Toulouse, LAAS, F-31400 Toulouse, France
{siala, hebrard, huguet}@laas.fr

Abstract. The ATMOSTSEQCARD constraint is the conjunction of a cardinality constraint on a sequence of n variables and of $n - q + 1$ constraints ATMOST u on each subsequence of size q.
This constraint is useful in car-sequencing and crew-rostering problems. In [18], two algorithms designed for the AMONGSEQ constraint were adapted to this constraint with a $O(2^q n)$ and $O(n^3)$ worst case time complexity, respectively. In [10], another algorithm similarly adaptable to the ATMOSTSEQCARD constraint with a $O(n^2 \log n)$ worst case time complexity was proposed.
In this paper, we introduce an algorithm for achieving Arc Consistency on the ATMOSTSEQCARD constraint with a $O(n)$ (hence optimal) worst case time complexity. We then empirically study the efficiency of our propagator on instances of the car-sequencing and crew-rostering problems.

1 Introduction

In many applications there are restrictions on the successions of decisions that can be taken. Some sequences are allowed or preferred while other are forbidden. For instance, in crew-rostering applications, it is often not recommended to have an employee work on an evening or a night shift and then again on the morning shift of the next day.

Several constraints have been proposed to deal with this type of problems. The REGULAR [12] and COST-REGULAR constraints [7] make it possible to restrict sequences in an arbitrary way. However, there might often exist more efficient algorithm for the particular case at hand. For instance, filtering algorithms have been proposed for the AMONGSEQ constraint in [6, 10, 19, 18]. This constraint ensure that all subsequences of size q have at least l but no more than u values in a set v. This constraint is often applied to car-sequencing and crew-rostering problems. However, the constraints in these two benchmarks do not correspond exactly to this definition. Indeed, there are often no lower bound restriction on the number of values ($l = 0$). Instead, the number of values in the set v is often constrained by an overall demand.

In this paper we consider the constraint ATMOSTSEQCARD. This constraint, posted on n variables $x_1, \ldots, x_n$, ensures that, in every subsequence of length q, no more than u are set to a value in a set v, and that over all the sequence, exactly d are set to values in v. In car-sequencing, this constraint allows to state that given an option, no subsequence of length q can involve more than u classes of cars requiring this option,

and that exactly d cars require it overall. In crew-rostering problems, one can state, for instance, that a worker must have at least a 16h break between two 8h shifts, or that a week should not involve more than 40h of working time, while enforcing a total number of worked hours over the scheduling period.

The rest of the paper is organized as follows: In Section 2 we give a brief state of the art of the sequence constraints. Then in Section 3, we give a linear time (hence optimal) algorithm for filtering this constraint. Last, in Section 4 we evaluate our new propagator on car-sequencing benchmarks, before concluding in Section 5.

2 CSP and SEQUENCE Constraints

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 a set of domains and $\mathcal{C}$ is a set of constraints that specify allowed combinations of values for subsets of variables. We denote by $\min(x)$ and $\max(x)$ the minimum and maximum values in $\mathcal{D}(x)$, respectively. An assignment of a set of variables $\mathcal{X}$ is a tuple w, where $w[i]$ denotes the value assigned to the variable x_i. A constraint $C \in \mathcal{C}$ on a set of variables $\mathcal{X}$ defines a relation on the domains of variables in $\mathcal{X}$. An assignment w is consistent for a constraint C iff it belongs to the corresponding relation. A constraint C is *arc consistent* (AC) iff, for every value j of every variable x_i in its scope there exists a consistent assignment w such that $w[i] = j$, i.e., a *support*.

There are several variants of the SEQUENCE constraints, we first review them and then motivate the need for the variant proposed in this paper: ATMOSTSEQCARD. In the following definitions, v is a set of integers and l, u, q are integers. Sequence constraints are conjunctions of AMONG constraints, constraining the number of occurrences of a set of values in a set of variables.

Definition 1. $\text{AMONG}(l, u, [x_1, \ldots, x_q], v) \Leftrightarrow l \leq |\{i \mid x_i \in v\}| \leq u$

Chains of AMONG Constraints: The AMONGSEQ constraint, first introduced in [2], is a chain of AMONG constraints of width q slid along a vector of n variables.

Definition 2. $\text{AMONGSEQ}(l, u, q, [x_1, \ldots, x_n], v) \Leftrightarrow \bigwedge_{i=0}^{n-q} \text{AMONG}(l, u, [x_{i+1}, \ldots, x_{i+q}], v)$

The first (incomplete) algorithm for filtering this constraint was proposed in 2001 [1]. Then in [19, 18], two complete algorithms for filtering the AMONGSEQ constraint were introduced. First, a reformulation using the REGULAR constraint using 2^{q-1} states and hence achieving AC in $O(2^q n)$ time. Second, an algorithm achieving AC with a worst case time complexity of $O(n^3)$. Moreover, this last algorithm is able to handle arbitrary sets of AMONG constraints on consecutive variables (denoted GEN-SEQUENCE), however in $O(n^4)$. Last, a flow-based algorithm achieving AC in $O(n^2)$ was introduced in [10]. Here again, the algorithm can be adapted, using more general linear programming tools to handle the GEN-SEQUENCE constraint in $O(n^2 \log n)$ in the worst case.

Chain of ATMOST Constraints: Although useful in both applications, the AMONGSEQ constraint does not model exactly the type of sequences useful in car-sequencing and crew-rostering applications.

First, there is often no lower bound for the cardinality of the sub-sequences, i.e., $l = 0$. Therefore AMONGSEQ is unnecessarily general in that respect. Moreover, the capacity constraint on subsequences is often paired with a cardinality requirement.

For instance, in car-sequencing, classes of car requiring a given option cannot be clustered together because a working station can only handle a fraction of the cars passing on the line (*at most* u times in any sequence of length q). The total number of occurrences of these classes is a requirement, and translates as an overall cardinality constraint rather than lower bounds on each sub-sequence.

In crew-rostering, allowed shift patterns can be complex, hence the REGULAR constraint is often used to model them. However, working in at most u shifts out of q is a useful particular case. If days are divided into three 8h shifts, ATMOSTSEQ with $u = 1$ and $q = 3$ makes sure that no employee work more than one shift per day *and* that there must be a 24h break when changing shifts. Moreover, similarly to car-sequencing, the lower bound on the number of worked shifts is global (monthly, for instance).

In other words, we often have a chain of ATMOST constraints, defined as follows:

Definition 3. $\text{ATMOST}(u, [x_1, \ldots, x_q], v) \Leftrightarrow \text{AMONG}(0, u, [x_1, \ldots, x_q], v)$

Definition 4. $\text{ATMOSTSEQ}(u, q, [x_1, \ldots, x_n], v) \Leftrightarrow \bigwedge_{i=0}^{n-q} \text{ATMOST}(u, [x_{i+1}, \ldots, x_{i+q}], v)$

However, it is easy to maintain AC on this constraint. Indeed, the ATMOST constraint is monotone, i.e., the set of supports for value 0 is a super-set of the set of supports for value 1. Hence a ATMOSTSEQ constraint is AC iff each ATMOST is AC [4].

A good tradeoff between filtering power and complexity can be achieved by reasoning about the total number of occurrences of values from the set v together with the chain of ATMOST constraints.[1] We therefore introduce the ATMOSTSEQCARD constraint, defined as the conjunction of an ATMOSTSEQ with a cardinality constraints on the total number of occurrences of values in v:

Definition 5. $\text{ATMOSTSEQCARD}(u, q, d, [x_1, \ldots, x_n], v) \Leftrightarrow$

$$\text{ATMOSTSEQ}(u, q, d, [x_1, \ldots, x_n], v) \wedge |\{i \mid x_i \in v\}| = d$$

The two AC algorithms introduced in [19] were adapted in [18] to achieve AC on the ATMOSTSEQCARD constraint. First, in the same way that AMONGSEQ can be encoded with a REGULAR constraint, ATMOSTSEQCARD can be encoded with a COST-REGULAR constraint, where the cost stands for the overall demand, and it is increased on transitions labeled with the value 1. This procedure has the same worst case time complexity, i.e., $O(2^q n)$. Second, the more general version of the polynomial algorithm (GEN-SEQUENCE) is used, to filter the following decomposition of the ATMOSTSEQCARD constraint into a conjunction of AMONG:

$$\text{ATMOSTSEQCARD}(u, q, d, [x_1, \ldots, x_n], v) \Leftrightarrow$$
$$\bigwedge_{i=0}^{n-q} \text{AMONG}(0, u, q, [x_{i+1}, \ldots, x_{i+q}], v) \ \wedge \ \text{AMONG}(d, d, [x_1, \ldots, x_n], v)$$

[1] This modeling choice is used in [18] on car-sequencing.

Since the number of AMONG constraints is $O(n)$, the algorithm of van Hoeve et al. runs in fact in $O(n^3)$ on this decomposition. Last, the algorithm of Maher et al. runs in $O(n^2 \log n)$ time and is thus the best known procedure to achieve AC on this constraint.

Global Sequencing Constraint: Finally, in the particular case of car-sequencing, not only we have an overall cardinality for the values in v, but each value corresponds to a class of car and has a required number of occurrences. Therefore, Puget and Régin proposed to consider the conjunction of a AMONGSEQ and a GCC constraint. Let c_l and c_u be two mapping on integers such that $c_l(j) \leq c_u(j)\ \forall j$, and let $\mathcal{D} = \bigcup_{i=1}^{n} \mathcal{D}(x_i)$. The Global Cardinality Constraint (GCC) is defined as follows:

Definition 6. $\text{GCC}(c_l, c_u, [x_1, \ldots, x_n]) \Leftrightarrow \bigwedge_{j \in \mathcal{D}} c_l(j) \leq |\{i \mid x_i = j\}| \leq c_u(j)$

The Global Sequencing Constraint is defined as follows:

Definition 7. $\text{GSC}(l, u, q, c_l, c_u, [x_1, \ldots, x_n], v) \Leftrightarrow$

$$\text{AMONGSEQ}(l, u, q, [x_1, .., x_n], v) \wedge \text{GCC}(c_l, c_u, [x_1, .., x_n])$$

The mappings c_l and c_u are defined so that for a value v, both $c_l(v)$ and $c_u(v)$ map to the number of occurrences of the corresponding class of car. A reformulation of this constraint into a set of GCC constraints was introduced in [15]. However, achieving AC on this constraint is NP-hard [3]. In fact, so is achieving bounds consistency.[2]

3 The ATMOSTSEQCARD Constraint

In this section we introduce a linear filtering algorithm for the ATMOSTSEQCARD constraint. We first give a simple greedy algorithm for finding a support with a $O(nq)$ time complexity. Then, we show that one can use two calls to this procedure to enforce AC. Last, we show that its worst case time complexity can be reduced to $O(n)$.

It was observed in [18] and [10] that we can consider Boolean variables and $v = \{1\}$, since the following decomposition of AMONG (or ATMOST) does not hinder propagation as it is Berge-acyclic [6]:

$$\text{AMONG}(l, u, [x_1, \ldots, x_q], v) \Leftrightarrow \bigwedge_{i=1}^{q} x_i \in v \leftrightarrow x'_i = 1 \wedge l \leq \sum_{i=1}^{q} x'_i \leq u$$

Therefore, throughout the paper, we shall consider the following restriction of the ATMOSTSEQCARD constraint, defined on Boolean variables, and with $v = \{1\}$:

Definition 8.

$$\text{ATMOSTSEQCARD}(u, q, d, [x_1, \ldots, x_n]) \Leftrightarrow \bigwedge_{i=0}^{n-q} (\sum_{l=1}^{q} x_{i+l} \leq u) \wedge (\sum_{i=1}^{n} x_i = d)$$

[2] Comics note, Nina Narodytska.

3.1 Finding a Support

Let w be a n-tuple in $\{0,1\}^n$, $w[i]$ denotes the i^{th} element of w, $|w| = \sum_{i=1}^{n} w[i]$ its cardinality, and $w[i : j]$ the $(|j - i| + 1)$-tuple equal to w on the sub-sequence $[x_i, \ldots, x_j]$.

We first show that one can find a support by greedily assigning variables in a lexicographical order to the value 1 whenever possible, that is, while taking care of not violating the AtMostSeq constraint. More precisely, doing so leads to an assignment of maximal cardinality, which may easily be transformed into an assignment of cardinality d. This greedy rule, computing an assignment w maximizing the cardinality of the sequence $(x_1, \ldots, x_n)$ subject to a AtMostSeq constraint (with parameters u and q), is shown in Algorithm 1.

First, the tuple w is initialized to the minimum value in the domain of each variable in Line 1. Then, at each step $i \in [1, \ldots, n]$ of the main loop, $c(j)$ is the cardinality of the j^{th} subsequence involving the variable x_i, i.e. at step i, $c(j) = \sum_{l=\max(1,i-q+j)}^{\min(n,i+j-1)} w[l]$. According to the greedy rule sketched above, we set $w[i]$ to 1 iff is not yet assigned ($\mathcal{D}(x_i) = \{0,1\}$) and if this does not violate the capacity constraints, that is, there is no subsequence involving x_i of cardinality u or more. This is done by checking the maximum value of $c(j)$ for $j \in [1, \ldots, q]$ (Condition 2). In that case, the cardinality of every subsequence involving x_i is incremented (Line 3). Finally when moving to the next variable, the values of $c(j)$ are shifted (Line 4), and the value of $c(q)$ is obtained by adding the value of $w[i + q]$ and subtracting $w[i]$ to its previous value (Line 5). From now on, we shall denote $\overrightarrow{w}$ the assignment found by `leftmost` on the sequence $x_1, \ldots, x_n$. Moreover, we shall denote $\overleftarrow{w}$ the assignment found by the same algorithm, however on the sequence $x_n, \ldots, x_1$, that is, right to left. Notice that, in order to simplify the notations, $\overleftarrow{w}[i]$ shall denote the value assigned by `leftmost` to the variable x_i, and not x_{n-i+1} as it would really be if we gave the reversed sequence as input.

Lemma 1. *`leftmost` maximizes $\sum_{i=1}^{n} x_i$ subject to* AtMostSeq$(u, q, [x_1, \ldots, x_n])$.

Proof. Let $\overrightarrow{w}$ be the assignment found by `leftmost`, and suppose that there exists w another assignment (consistent for AtMostSeq$(u, q, [x_1, \ldots, x_n])$) such that $|w| > |\overrightarrow{w}|$. Let i be the smallest index such that $\overrightarrow{w}[i] \neq w[i]$. By definition of $\overrightarrow{w}$, we know that $\overrightarrow{w}[i] = 1$ hence $w[i] = 0$. Now let j be the smallest index such that $\overrightarrow{w}[j] < w[j]$ (it must exists since $|w| > |\overrightarrow{w}|$).

We first argue that the assignment w' equal to w except that $w'[i] = 1$ and $w'[j] = 0$ (as in $\overrightarrow{w}$) is consistent for AtMostSeq. Clearly, its cardinality is not affected by this swap, hence $|w'| = |w|$. Now, we consider all the sum constraints which scopes are changed by this swap, that is, the sums $\sum_{l=a}^{a+q-1} w'[l]$ on intervals $[a, a + q - 1]$ such that $a \leq i < a + q$ or $a \leq j < a + q$. There are three cases:

1. Suppose first that $a \leq i < j < a + q$: in this case, the value of the sum is the same in w and w', therefore it is lower than or equal to u.
2. Suppose now that $i < a \leq j < a+q$: in this case, the value of the sum is decreased by 1 from w to w', therefore it is lower than or equal to u.
3. Last, suppose that $a \leq i < a + q \leq j$: in this case, for any $l \in [a, a + q - 1]$, we have $w'[l] \leq \overrightarrow{w}[l]$ since j is the smallest integer such that $\overrightarrow{w}[j] < w[j]$, hence the sum is lower than or equal to u.

Algorithm 1: `leftmost`

```
   count ← 0;
1  foreach i ∈ [1, ..., n] do w[i] ← min(x_i);
   foreach i ∈ [1, ..., q] do w[n + i] ← 0;
   c(1) ← w[1];
   foreach j ∈ [2, ..., q] do c(j) ← c(j − 1) + w[l];
   foreach i ∈ [1, ..., n] do
2      if |D(x_i)| > 1 & max_{j∈[1,q]}(c(j)) < u then
           w[i] ← 1;
           count ← count + 1;
3          foreach j ∈ [1, ..., q] do c(j) ← c(j) + 1;
4      foreach j ∈ [2, ..., q] do c(j − 1) ← c(j);
5      c(q) ← c(q − 1) + w[i + q] − w[i];
   return w;
```

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	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

Fig. 1: An algorithm to compute an assignment satisfying AtMostSeq$(2, 4, [x_1, \ldots, x_n])$ with maximal cardinality (left), and and example of its execution (right). Dots in the first column stand for unassigned variables. The second column shows the computed assignement w, and the next columns show the state of the variables $c(1), c(2), c(3)$ and $c(4)$ at the start of each iteration. The last column stands for the maximum value among $c(1), c(2), c(3)$ and $c(4)$.

Therefore, given a sequence w of maximum cardinality and that differs with $\overrightarrow{w}$ at rank i, we can build one of equal cardinality and that does not differs from $\overrightarrow{w}$ until rank $i + 1$. By iteratively applying this argument, we can obtain a sequence identical to $\overrightarrow{w}$, albeit with cardinality $|w|$, therefore contradicting our hypothesis that $|w| > |\overrightarrow{w}|$. □

Corollary 1. *Let $\overrightarrow{w}$ be the assignment returned by* `leftmost`*. There exists a solution of* AtMostSeqCard$(u, q, d, [x_1, \ldots, x_n])$ *iff the three following propositions hold:*

(1) AtMostSeq$(u, q, [x_1, \ldots, x_n])$ is satisfiable

(2) $\sum_{i=1}^{n} min(x_i) \leq d$

(3) $|\overrightarrow{w}| \geq d$.

Proof. It is easy to see that these conditions are all necessary: (1) and (2) come from the definition, and (3) is a direct application of Lemma 1. Now we prove that they are sufficient by showing that if these properties hold, then a solution exists. Since AtMostSeq$(u, q, [x_1, \ldots, x_n])$ is satisfiable, $\overrightarrow{w}$ does not violate the chain of AtMost constraints as the value 1 is assigned to x_i only if all subsequences involving x_i have cardinality $u - 1$ or less. Moreover, since $\sum_{i=1}^{n} min(x_i) \leq d$ there are at least $|\overrightarrow{w}| - d$ variables such that $min(x_i) = 0$ and $\overrightarrow{w}[i] = 1$. Assigning them to 0 does not violate the AtMostSeq constraint, hence there exists a support. □

Lemma 1 and Corollary 1 give us a polynomial support-seeking procedure for AtMostSeqCard. Indeed, the worst case time complexity of Algorithm 1 is in $O(nq)$. There are n steps and on each step, Lines 2, 3 and 4 involve $O(q)$ operations. Therefore, for each variable x_i, a support for $x_i = 0$ or $x_i = 1$ can be found in $O(nq)$.

Consequently we have a naive AC procedure running in $O(n^2q)$ time.

3.2 Filtering the Domains

In this section, we show that we can filter out all the values inconsistent with respect to the AtMostSeqCard constraint within the same time complexity as Algorithm 1.

First, we show that there can be inconsistent values only in the case where the cardinality $|\overrightarrow{w}|$ of the assignment returned by `leftmost` is exactly d: in any other case, the constraint is either violated (when $|\overrightarrow{w}| < d$) or AC, (when $|\overrightarrow{w}| > d$). The following proposition formalises this:

Proposition 1. *The constraint* AtMostSeqCard$(u, q, d, [x_1, \ldots, x_n])$ *is* AC *if the three following propositions hold:*
(1) AtMostSeq$(u, q, [x_1, \ldots, x_n])$ *is* AC
(2) $\sum_{i=1}^{n} min(x_i) \leq d$
(3) $|\overrightarrow{w}| > d$

Proof. By Corollary 1 we know that AtMostSeqCard$(u, q, d+1, [x_1, \ldots, x_n])$ is satisfiable. Let w be a satisfying assignment, and consider without loss of generality a variable x_i such that $|\mathcal{D}(x_i)| > 1$. Assume first that $w[i] = 1$. The solution w' equal to w except that $w'[i] = 0$ satisfies AtMostSeqCard$(u, q, d, [x_1, \ldots, x_n])$. Indeed, $|w'| = |w| - 1 = d$ and since AtMostSeq$(u, q, [x_1, \ldots, x_n])$ was satisfied by w it must be satisfied by w'. Hence, for every variable x_i, there exists a support for $x_i = 0$.

Suppose that $w[i] = 0$, and let $a < i$ (resp. $b > i$) be the largest (resp. smallest) index such that $w[a] = 1$ and $\mathcal{D}(x_a) = \{0, 1\}$ (resp. $w[b] = 1$ and $\mathcal{D}(x_b) = \{0, 1\}$). Let w' be the assignment such that $w'[i] = 1$, $w'[a] = 0$, $w'[b] = 0$, and $w = w'$ otherwise. We have $|w'| = d$, and we show that it satisfies AtMostSeq$(u, q, [x_1, \ldots, x_n])$. Consider a subsequence $x_j, \ldots, x_{j+q-1}$. If $j + q \leq i$ or $j > i$ then $\sum_{l=j}^{j+q-1} w'[l] \leq \sum_{l=j}^{j+q-1} w[l] \leq u$, so only indices j s.t. $j \leq i < j + q$ matter. There are two cases:

1. Either a or b or both are in the sub-sequence ($j \leq a < j + q$ or $j \leq b < j + q$). In that case $\sum_{l=j}^{i+q-1} w'[l] \leq \sum_{l=j}^{i+q-1} w[l]$.
2. Neither a nor b are in the sub-sequence ($a < j$ and $j + q \leq b$). In that case, since $\mathcal{D}(x_i) = \{0, 1\}$ and since condition (1) holds, we know that $\sum_{l=j}^{i+q-1} \min(x_l) < u$. Moreover, since $a < j$ and $j + q \leq b$, there is no variable x_l in that sub-sequence such that $w[l] = 1$ and $0 \in \mathcal{D}(x_l)$. Therefore, we have $\sum_{l=j}^{i+q-1} w[l] < u$, hence $\sum_{l=j}^{i+q-1} w'[l] \leq u$.

In both cases w satisfies all capacity constraints. □

Remember that achieving AC on AtMostSeq is trivial since Among is monotone. Therefore we focus of the case where AtMostSeq is AC, and $|\overrightarrow{w}| = d$. In particular, Propositions 2, 3, 4 and 5 only apply in that case. The equality $|\overrightarrow{w}| = d$ is therefore implicitly assumed in all of them.

Proposition 2. *If* $|\overrightarrow{w}[1 : i-1]| + |\overleftarrow{w}[i+1 : n]| < d$ *then* $x_i = 0$ *is not* AC.

Proof. Suppose that we have $|\overrightarrow{w}[1 : i-1]| + |\overleftarrow{w}[i+1 : n]| < d$ and suppose that there exists an assignment w such that $w[i] = 0$ and $|w| = d$.

By Lemma 1 on the sequence $x_1, \ldots, x_{i-1}$ we know that $\sum_{l=1}^{i-1} w[l] \leq |\overrightarrow{w}[1 : i-1]|$.

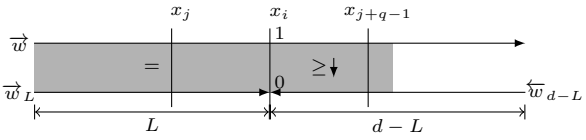

Fig. 2: Illustration of Proposition 4's proof. Horizontal arrows represent assignments.

By Lemma 1 on the sequence $x_n, \ldots, x_{i+1}$ we know that $\sum_{l=i+1}^{n} w[l] \leq |\overleftarrow{w}[i+1:n]|$.

Therefore, since $w[i] = 0$, we have $|w| = \sum_{l=1}^{n} w[l] < d$, thus contradicting the hypothesis that $|w| = d$. Hence, there is no support for $x_i = 0$. □

Proposition 3. *If* $|\overrightarrow{w}[1:i]| + |\overleftarrow{w}[i:n]| \leq d$ *then* $x_i = 1$ *is not* AC.

Proof. Suppose that we have $|\overrightarrow{w}[1:i]| + |\overleftarrow{w}[i:n]| \leq d$ and suppose that there exists an assignment w' such that $w'[i] = 1$ and $|w'| = d$.

By Lemma 1 on the sequence $x_1, \ldots, x_i$ we know that $\sum_{l=1}^{i} w'[l] \leq |\overrightarrow{w}[1:i]|$.

By Lemma 1 on the sequence $x_n, \ldots, x_i$ we know that $\sum_{l=i}^{n} w'[l] \leq |\overleftarrow{w}[i:n]|$.

Therefore, since $w'[i] = 1$, we have $|w'| = \sum_{l=1}^{i} w'[l] + \sum_{l=i}^{n} w'[l] - 1 < d$, thus contradicting the hypothesis that $|w'| = d$. Hence there is no support for $x_i = 1$. □

Propositions 2 and 3 entail a pruning rule. In a first pass, from left to right, one can use an algorithm similar to `leftmost` to compute and store the values of $|\overrightarrow{w}[1:i]|$ for all $i \in [1, \ldots, n]$. In a second pass, the values of $|\overleftarrow{w}[i:n]|$ for all $i \in [1, \ldots, n]$ are similarly computed by simply running the same procedure on the same sequence of variables, however *reversed*, i.e., from right to left. Using these values, one can then apply Proposition 2 and Proposition 3 to filter out the value 0 and 1, respectively. We detail this procedure in the next section.

We first show that these two rules are complete, that is, if AtMostSeq is AC, and the overall cardinality constraint is AC then an assignment $x_i = 0$ (resp. $x_i = 1$) is inconsistent iff Proposition 2 (resp. Proposition 3) applies. The following Lemma shows that the greedy rule maximises the density of 1s on any subsequence starting on x_1, and therefore minimises it on any subsequence finishing on x_n. Let `leftmost`(k) denote the algorithm corresponding to applying `leftmost`, however stopping whenever the cardinality of the assignment reaches a given value k.

Lemma 2. *Let* w *be a satisfying assignment of* AtMostSeq$(u, q, [x_1, \ldots, x_n])$. *If* $k \leq |w|$ *then the assignment* $\overrightarrow{w}_k$ *computed by* `leftmost`(k) *is such that, for any* $1 \leq i \leq n$: $\sum_{l=i}^{n} \overrightarrow{w}_k[l] \leq \sum_{l=i}^{n} w[l]$.

Proof. Let m be the index at which `leftmost`(k) stops. We distinguish two cases. If $i > m$, for any value l in $[m+1, \ldots, n]$, $\overrightarrow{w}_k[l] \leq w[l]$ (since $\overrightarrow{w}_k[l] = \min(x_l)$), hence $\sum_{l=i}^{n} \overrightarrow{w}_k[l] \leq \sum_{l=i}^{n} w[l]$. When $i \leq m$, clearly for $i = 1$, $\sum_{l=i}^{n} \overrightarrow{w}_k[l] \leq \sum_{l=i}^{n} w[l]$ since $|\overrightarrow{w}_k| \leq |w|$. Now consider the case of $i \neq 1$. Since $|\overrightarrow{w}_k| \leq |w|$, then $\sum_{l=i}^{n} \overrightarrow{w}_k[l] \leq |w| - \sum_{l=1}^{i-1} \overrightarrow{w}_k[l]$. Thus, $\sum_{l=i}^{n} \overrightarrow{w}_k[l] \leq \sum_{l=i}^{n} w[l] + (\sum_{l=1}^{i-1} w[l] -$

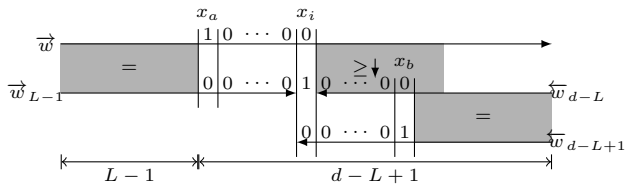

Fig. 3: Illustration of Proposition 5's proof. Horizontal arrows represent assignments.

$\sum_{l=1}^{i-1} \overrightarrow{w}_k[l]$). Moreover, by applying Lemma 1, we show that $\sum_{l=1}^{i-1} \overrightarrow{w}_k[l]$ is maximum, hence larger than or equal to $\sum_{l=1}^{i-1} w[l]$. Therefore, $\sum_{l=i}^{n} \overrightarrow{w}_k[l] \leq \sum_{l=i}^{n} w[l]$. □

Proposition 4. *If* AtMostSeq$(u, q, [x_1, \ldots, x_n])$ *is* AC*, and* $|\overrightarrow{w}[1:i-1]|+|\overleftarrow{w}[i+1:n]| \geq d$ *then* $x_i = 0$ *has a support.*

Proof. Let $\overrightarrow{w}$ be the assignment found by `leftmost`. We consider, without loss of generality, a variable x_i such that $\mathcal{D}(x_i) = \{0, 1\}$ and $|\overrightarrow{w}[1:i-1]|+|\overleftarrow{w}[i+1:n]| \geq d$ and show that one can build a support for $x_i = 0$. If $\overrightarrow{w}[i] = 0$ or $\overleftarrow{w}[i] = 0$ then there exists a support for $x_i = 0$, hence we only need to consider the case where $\overrightarrow{w}[i] = \overleftarrow{w}[i] = 1$.

Let $L = |\overrightarrow{w}[1:i-1]|$, and let $\overleftarrow{w}_{d-L}$ be the result of `leftmost`$(d-L)$ on the subsequence $x_n, \ldots, x_i$. We will show that w, defined as the concatenation of $\overrightarrow{w}[1:i-1]$ and $\overleftarrow{w}_{d-L}[i:n]$ is a support for $x_i = 0$.

First, we show that $w[i] = 0$, that is $\overleftarrow{w}_{d-L}[i] = 0$. By hypothesis, since $|\overrightarrow{w}[1:i-1]|+|\overleftarrow{w}[i+1:n]| \geq d$, we have $|\overleftarrow{w}[i+1:n]| \geq d - L$. Now consider the sequence $x_i, \ldots, x_n$, and let w' be the assignment such that $w'[i] = 0$, and $w' = \overleftarrow{w}[i+1:n]$ otherwise. Since $|w'| = |\overleftarrow{w}[i+1:n]| \geq d - L$, by Lemma 2, we know that w' has a higher cardinality than $\overleftarrow{w}_{d-L}$ on any subsequence starting in i, hence $w[i] = \overleftarrow{w}_{d-L}[i] = w'[i] = 0$.

Now we show that w does not violate the AtMostSeq constraint. Obviously, since it is the concatenation of two consistent assignments, it can violate the constraint only on the junction, i.e., on a sub-sequence $x_j, \ldots, x_{j+q-1}$ such that $j \leq i$ and $i < j + q$.

We show that the sum of any such subsequence is less or equal to u by comparing with $\overrightarrow{w}$, as illustrated in Figure 2. We have $\sum_{l=j}^{j+q-1} \overrightarrow{w}[l] \leq u$, and $\sum_{l=j}^{i-1} \overrightarrow{w}[l] = \sum_{l=j}^{i-1} w[l]$. Moreover, by Lemma 2, since $|\overrightarrow{w}[i:n]| = |\overleftarrow{w}_{d-L}| = d - L$ we have $\sum_{l=i}^{j+q-1} \overleftarrow{w}_{d-L}[l] \leq \sum_{l=i}^{j+q-1} \overrightarrow{w}[l]$ hence $\sum_{l=i}^{j+q-1} w[l] \leq \sum_{l=i}^{j+q-1} \overrightarrow{w}[l]$. Therefore, we can conclude that $\sum_{l=j}^{j+q-1} w[l] \leq u$. □

Proposition 5. *If* AtMostSeq$(u, q, [x_1, \ldots, x_n])$ *is* AC*,* $|w[1:i]| + |w[n:i]| > d$ *then* $x_i = 1$ *has a support.*

Proof. Let $\overrightarrow{w}$ and $\overleftarrow{w}$ be the assignments found by `leftmost`, on respectively $x_1, \ldots, x_n$ and $x_n, \ldots, x_1$. We consider, without loss of generality, a variable x_i such that $\mathcal{D}(x_i) = \{0, 1\}$ and $|\overrightarrow{w}[1:i]| + |\overleftarrow{w}[i:n]| > d$ and show that one can build a support for $x_i = 1$.

If $\overrightarrow{w}[i] = 1$ or $\overleftarrow{w}[i] = 1$ then there exists a support for $x_i = 1$, hence we only need to consider the case where $\overrightarrow{w}[i] = \overleftarrow{w}[i] = 0$.

Let $L = |\overrightarrow{w}[1:i]| = |\overrightarrow{w}[1:i-1]|$ (this equality holds since $\overrightarrow{w}[i] = 0$). Let $\overrightarrow{w}_{L-1}$ be the assignment obtained by using $\texttt{leftmost}(L-1)$ on the subsequence $x_1, \ldots, x_{i-1}$, and let $\overleftarrow{w}_{d-L}$ be the assignment returned by $\texttt{leftmost}(d-L)$ on the subsequence $x_n, \ldots, x_{i+1}$.

We show that w such that $w[i] = 1$, equal to $\overrightarrow{w}_{L-1}$ on $x_1, \ldots, x_{i-1}$ and to $\overleftarrow{w}_{d-L}$ on $x_{i+1}, \ldots, x_n$, is a support.

Clearly $|w| = d$, therefore we only have to make sure that all capacity constraints are satisfied. Moreover, since it is the concatenation of two consistent assignments, it can violate the constraint only on the junction, i.e., on a sub-sequence $x_j, \ldots, x_{j+q-1}$ such that $j \leq i$ and $i < j + q$.

We show that the sum of any such subsequence is less or equal to u by comparing with $\overrightarrow{w}$ and $\overleftarrow{w}_{d-L}$ (see Figure 3). First, observe that on the subsequence $x_1, \ldots, x_{i-1}$, $\overrightarrow{w}_{L-1} = \overrightarrow{w}$, except for the largest index a such that $\overrightarrow{w}[a] = 1$ and $w[a] = 0$. Similarly on $x_n, \ldots, x_{i+1}$, we have $\overleftarrow{w}_{d-L} = \overleftarrow{w}_{d-L+1}$, except for the smallest b such that $\overleftarrow{w}_{d-L+1}[b] = 1$. There are two cases:

Suppose first that $j > a$. In that case, $\sum_{l=j}^{j+q-1} w[l] = \sum_{l=i}^{j+q-1} \overleftarrow{w}_{d-L+1}[l]$ if $j + q - 1 \geq b$, and otherwise it is equal to 1. It is therefore alway less than or equal to u since $i \geq j$ (and we assume $u \geq 1$).

Now suppose that $j \leq a$. In that case, consider first the subsequence $x_j, \ldots, x_i$. On this interval, the cardinality of w is the same as that of $\overrightarrow{w}$, i.e., $\sum_{l=j}^{i} w[l] = \sum_{l=j}^{i-1} \overrightarrow{w}_{L-1}[l] + 1 = \sum_{l=j}^{i} \overrightarrow{w}[l]$. On the subsequence $x_{i+1}, \ldots, x_{j+q-1}$, observe that $|w[i+1:n]| = |\overrightarrow{w}[i+1:n]| = d - L$, hence by Lemma 2, we have $\sum_{l=i+1}^{j+q-1} w[l] = \sum_{l=i+1}^{j+q-1} \overleftarrow{w}_{d-L}[l] \leq \sum_{l=i+1}^{j+q-1} \overrightarrow{w}[l]$. Therefore $\sum_{l=j}^{j+q-1} w[l] \leq \sum_{l=j}^{j+q-1} \overrightarrow{w}[l] \leq u$. □

3.3 Algorithmic Complexity

Using Propositions 2, 3, 4 and 5 one can design a filtering algorithm with same worst case time complexity as `leftmost`. In this section, we introduce a linear time implementation of `leftmost`, denoted `leftmost_count`, that returns the values of $\overrightarrow{w}[1:i]$ for all values of i (Algorithm 2).

It is easy to see that `leftmost_count` has a $O(n)$ worst case time complexity. In order to prove its correctness, we will show that the assignment computed by `leftmost_count` is the same as that computed by `leftmost`.

Proof (sketch). We only sketch the proof here. The following three invariants are true at the beginning of each step i of the main loop:

- The cardinality of the j^{th} sub-sequence is given by $c[(i+j-2) \bmod q] + count[i-1]$.
- The number of sub-sequences of cardinality k is given by $occ[n - count[i-1] + k]$.
- The cardinality maximum of any sub-sequence is given by max_c.

1^{st} invariant: The value of $c[j]$ is updated in two ways in `leftmost`. First, at each step of the loop the values in $c[1]$ through to $c[q]$ are shifted to the left. Therefore,

Algorithm 2: `leftmost_count`

```
Data: u, q, d, [x_1, ..., x_n]
Result: count : [0, ..., n] ↦ [0, ..., n − 1]
foreach i ∈ [1, ..., n] do
    w[i] ← min(x_i);
    occ[i] = 0;
foreach i ∈ [0, ..., n] do count[i] ← 0;
c[0] ← w[1];
foreach i ∈ [1, ..., u + 1] do occ[n + i] = 0;
foreach i ∈ [1, ..., q] do
    w[n + i] ← 0;
    c[i] ← c[i − 1] + w[i + 1];
    occ[n + c[i − 1]] ← occ[n + c[i − 1]] + 1;
max_c ← max({c[i] | i ∈ [1, ..., q]});
foreach i ∈ [1, ..., n] do
1       if max_c < u & |D(x_i)| > 1 then
            max_c ← max_c + 1;
            count[i] ← count[i − 1] + 1;
            w[i] ← 1;
        else
            count[i] ← count[i − 1];
        prev ← c[(i − 1) mod q];
        next ← c[(i + q − 2) mod q] + w[i + q] − w[i];
        c[(i − 1) mod q] ← next;
        if prev ≠ next then
            occ[n + next] ← occ[n + next] + 1;
            if next + count > max_c then
                max_c ← max_c + 1;
            occ[n + prev] ← occ[n + prev] − 1;
            if occ[n + prev] = 0 & prev + count = max_c then
                max_c ← max_c − 1;
return count;
```

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

Fig. 4: Example of the execution of Algorithm 3 for $u = 4,\ q = 8,\ d = 12$. The first line stands for current domains, dots are unassigned variables (hence $ub = 10$). The two next lines give the assignments $\overrightarrow{w}$ and $\overleftarrow{w}$ obtained by running `leftmost_count` from left to right and from right to left, respectively. The third and fourth lines stand for the values of $L[i] = |\overrightarrow{w}[1:i]|$ and $R[n-i+1] = |\overleftarrow{w}[i:n]|$. The fifth and sixth lines correspond to the application of, respectively, Proposition 2 and 3. Last, the seventh line give the arc consistent closure of the domains.

there is only one really new value. By using the modulo operation, we can update only one of these values. Second, when $w[i]$ takes the value 1, we increment $c[1]$ up to $c[q]$. Since this happened exactly $count[i-1]$ times at the start of step i, we can simply add $count[i-1]$ to obtain the same value as in `leftmost`.

2^{nd} invariant: The data structure *occ* is a table, storing at index $n+k$ the number of subsequences involving x_i with cardinality k for the current assignment w. Therefore, by decrementing the pointer to the first element of the table we in effect shift the entire table. Here again the value of $count[i-1]$, or rather the expression $(n - count[i-1])$ points exactly to the required starting point of the table.

3^{rd} invariant: The maximum (or minimum) cardinality (of subsequences involving x_i) can change by a unit at the most from one step to the next. Therefore, when the variable max_c needs to change, it can only be incremented (when $occ[n - count[i -$

$1] + max\_c + 1]$ goes from the value 0 to 1) or decremented (when $occ[n - count[i - 1] + max\_c]$ goes from the value 1 to 0). □

Algorithm 3: AC(ATMOSTSEQCARD)

Data: $[x_1, \ldots, x_n], u, q, d$
Result: AC on ATMOSTSEQCARD$(u, q, d, [x_1, .., x_n])$
AC(ATMOSTSEQ)$(u, q, [x_1, \ldots, x_n])$;
$ub \leftarrow d - \sum_{i=1}^{n} \min(x_i)$;
$L \leftarrow$ `leftmost_count`$([x_1, \ldots, x_n], u, q, d)$;
if $L[n] = ub$ **then**
  $R \leftarrow$ `leftmost_count`$([x_n, \ldots, x_1], u, q, d)$;
  foreach $i \in [1, \ldots, n]$ *such that* $\mathcal{D}(x_i) = \{0, 1\}$ **do**
    if $L[i] + R[n - i + 1] \leq ub$ **then** $\mathcal{D}(x_i) \leftarrow \{0\}$;
    if $L[i - 1] + R[n - i] \leq ub$ **then** $\mathcal{D}(x_i) \leftarrow \{1\}$;

Algorithm 3 computes the AC closure of a constraint ATMOSTSEQCARD$(u, q, d, [x_1, \ldots, x_n])$. In the first line, the AC closure of ATMOSTSEQ$(u, q, [x_1, \ldots, x_n])$ is computed. It ensures that the filtering rules introduced in this paper hold. For lack of space, we do not give a pseudo-code for achieving AC on ATMOSTSEQ. However, it can be done in linear time using a procedure similar to `leftmost_count`. We want to compute, for the assignment corresponding to the lower bound of each domain, if an unassigned variable is covered by a subsequence of cardinality u for the lower bounds of the domains. We do it using a truncated version of `leftmost_count`: the values of $w[i]$ are never updated, i.e., they are set to the minimum value in the domain and we never enter the if-then-else block starting at condition 1 in Algorithm 2. Moreover, we store the value of max_c for each value of i in a table that we can subsequently use to achieve AC on ATMOSTSEQ, by going through it and assigning 0 to any unassigned variable covered by at least one subsequence of cardinality u.

The remainder is a straight application of Propositions 2, 3, 4 and 5. We give an example of its execution in Figure 4. The worst case time complexity of Algorithm 3 is therefore $O(n)$, hence optimal.

4 Experimental Results

We tested our filtering algorithm on two benchmarks: car-sequencing and crew-rostering. All experiments ran on Intel Xeon CPUs 2.67GHz under Linux. For each instance, we launched 5 randomized runs with a 20 minutes time cutoff.[3] All models are implemented using Ilog-Solver.

Since we compare propagators, we averaged the results across several branching heuristics to reduce the bias that these can have on the outcome. For each considered data set, we report the total number of successful runs (*#sol*). Then, we report the CPU time (*time*) in seconds and number of backtracks (*avg bts*) both averaged over all successful runs, instances and branching heuristics. Moreover, we report the maximum

[3] For a total of approximately 200 days of CPU time.

number of backtracks (*max bts*) in the same scope. We emphasize the statistics of the best method (w.r.t. *#sol*) for each data set using bold face fonts.

4.1 Car-sequencing

In the car-sequencing problem [8, 17], n vehicles have to be produced on an assembly line, subject to capacity and demand constraints.

We use a standard model, implemented in Ilog-Solver. We have n integer variables standing for the class of vehicles in each slot of the assembly line and nm boolean variables y_i^j standing for whether the vehicle in the i^{th} slot requires option j. The demand for each class is enforced with a GCC [14]. We compare four models for the capacity constraints coupled with the demand on each option (derived from the demand on classes). In the first model (*sum*) a simple decomposition into a chain of sum constraints plus an extra sum for the demand is used. In the second (*gsc*), we use one GSC constraint per option. In the third, (*amsc*), we use the AC procedure introduced in this paper for the ATMOSTSEQCARD constraint. Finally, in the fourth (*amsc+gsc*) we combine the GSC constraint with our filtering algorithm.

We use 34 different heuristics, obtained by combining different ways of *exploring* the assembly line either in lexicographic order or from the middle to the sides); of *branching* on affectation of a class to a slot or of an option to a slot; of *selecting* the best class or option among a number of natural criteria (such as maximum demand, minimum u/q ratio, as well as other criteria described in or derived from [5, 16]).

We use benchmarks available in the CSPLib [9] divided into four sets of respectively 70, 4, 5 and 7 instances ranging from 100 to 400 cars. Instances of the third set are all unsatisfiable all other are satisfiable.

The results are given in Table 1. In all cases, the best number of solved problems is obtained either by *amsc+gsc* (for small or unsatisfiable instances), or by *amsc* alone (for larger instances of set2 and set4). Overall, we observe that the GSC constraint allows to prune much more values than ATMOSTSEQCARD. However, it slows down the search by a substantial amount (we observed a factor 12.5 on the number of nodes explored per second). Moreover, the amounts of filtering obtained by these two methods are incomparable. Therefore combining them is always better than using GSC alone.

In [18] the authors applied their method to set1, set2 and set3 only. For their experiments, they considered the best result provided by 2 heuristics (σ and min domain). When using COST-REGULAR or GEN-SEQUENCE filtering alone, 50.7% of problems are solved and when combining either COST-REGULAR or GEN-SEQUENCE with GSC, 65.2% of problems are solved (with a time out of 1 hour). In our experiments, in average over the 34 heuristics and the 5 re-starts, ATMOSTSEQCARD and GSC solve respectively 82.5% and 86.11% of instances and combining ATMOSTSEQCARD with GSC solves 86.36% instances in a time out of 20 minutes.

4.2 Crew-rostering

In the crew-rostering problem, working shifts have to be attributed to employees over a period, so that there is enough employee to ensure the required service at any time

Table 1: Evaluation of the filtering methods (averaged over all heuristics)

Method	set1 ($70 \times 34 \times 5$)			set2 ($4 \times 34 \times 5$)			set3 ($5 \times 34 \times 5$)			set4 ($7 \times 34 \times 5$)		
	#sol	avg bts	time	#sol	avg bts	time	#sol	avg bts	time	#sol	avg bts	time
sum	8480	231.2K	13.93	95	1.4M	76.60	0	-	> 1200	64	543.3K	43.81
gsc	11218	1.7K	3.60	325	131.7K	110.99	31	55.3K	276.06	140	25.2K	56.61
amsc	10702	39.1K	4.43	**360**	**690.8K**	**72.00**	16	40.3K	8.62	**153**	**201.4K**	**33.56**
amsc+gsc	**11243**	**1.2K**	**3.43**	339	118.4K	106.53	**32**	**57.7K**	**285.43**	147	23.8K	66.45

and working regulations are respected. The latter condition can entail a wide variety of constraints. Previous works [11] [13] used allowed (or forbidden) patterns to express successive shift constraints. For example, with 3 shifts of 8 hours per day: D (day), E (evening) and N (night), ND can be forbidden since employees need some rest after night shifts. In this paper we consider a simple case involving 20 employees with 3 shifts of 8 hours per days where no employee can work more than one 8h shift per day, no more than 5 days a week, and the break between two worked shifts must be at least 16h. The planning horizon is of 28 days, and each employee must work 34 hours per week in average (17 shifts over the 4 weeks period).

Table 2: Evaluation of the filtering methods (averaged over all heuristics)

Benchmarks	underconstrained ($5 \times 2 \times 126$)				hard ($5 \times 2 \times 111$)				overconstrained ($5 \times 2 \times 44$)			
	#sol	avg bts	max bts	time	#sol	avg bts	max bts	time	#sol	avg bts	max bts	time
sum	1229	110.5K	10.1M	12.72	574	370.7K	13.4M	38.45	272	52.1K	5.7M	5.56
gsc	1210	6.2K	297.2K	29.19	579	23.5K	433.5K	77.78	276	7.7K	378.9	24.14
amsc	**1237**	**34.2K**	**7.5M**	**5.82**	**670**	**213.4K**	**7.5M**	**31.01**	**284**	**51.3**	**7.5M**	**6.22**

We use a model with one Boolean variable e_{ij} for each of the m employees and each of the n shifts stating if employee i works on shift j. The demand d_j^s on each shift j is enforced through a sum constraint $\sum_{i=1}^{m} e_{ij} = d_j^s$.The other constraints are stated using two AtMostSeqCard constraints per employee, one with ratio $u/q = 1/3$, another with ratio $5/21$, and both with the same demand $d = 17$ corresponding to 34 hours of work per week. We compare three models. In the first (*sum*), we use a decomposition in a chain of sum constraints. In the second (*gsc*), we use the Gsc constraint to encode it. Observe that in this case, since the domains are Boolean, the Gcc within the Gsc constraint is in fact nothing more than a sum. Therefore, it cannot prune more than AtMostSeqCard (however it is stronger than the simple sum decomposition). In the third model (*amsc*) we use the algorithm presented in this paper.

281 instances were generated, with employee unavailability ranging from from 18% to 46% by increment of 0.1. We partition the instances into three sets, with in the first 126 instances with lowest unavailability (all satisfiable), in the third, the 44 instances with highest unavailability (mostly unsatisfiable), and the rest in the second group.

We used two branching heuristic. In the first we chose the employee with minimum slack, and assign it to its possible shift of maximum demand. In the second we use the same criteria, but select the shift first and then the employee.

We report the results in Table 2. The AC algorithm achieves more filtering than the sum decomposition and the GSC decomposition. However, depending on the heuristic choices and the random seed, the size of the search tree is not always smaller. We observe that our propagator is only marginally slower in terms of nodes explored per second than the sum decomposition and much faster (by a factor 20.4 overall) than GSC. It is able to solve 15.7% and 16.7% more problems within the 20 minutes cutoff in the "hard" set than the GSC and sum decompositions, respectively.

5 Conclusion

We introduced a linear optimal algorithm for achieving arc consistency on the constraint ATMOSTSEQCARD. The empirical evaluation on car-sequencing and crew-rostering benchmarks shows that this propagator is useful on these applications.

References

1. N. Beldiceanu and M. Carlsson. Revisiting the Cardinality Operator and Introducing the Cardinality-Path Constraint Family. In *ICLP*, pages 59–73, 2001.
2. N. Beldiceanu and E. Contejean. Introducing Global Constraints in CHIP. *Mathematical Computation Modelling*, 20(12):97–123, 1994.
3. C. Bessiere, E. Hebrard, B. Hnich, Z. Kiziltan, and T. Walsh. The Slide Meta-Constraint. In *CPAI Workshop, held alongside CP*, 2006.
4. C. Bessiere, E. Hebrard, B. Hnich, Z. Kiziltan, and T. Walsh. Slide: A Useful Special Case of the Cardpath Constraint. In *ECAI*, pages 475–479, 2008.
5. S. Boivin, M. Gravel, M. Krajecki, and C. Gagné. Résolution du Problème de Car-sequencing à l'Aide d'une Approche de Type FC. In *JFPC*, 2005.
6. S. Brand, N. Narodytska, C.-G. Quimper, P. J. Stuckey, and T. Walsh. Encodings of the Sequence Constraint. In *CP*, pages 210–224, 2007.
7. S. Demassey, G. Pesant, and L.-M. Rousseau. A Cost-Regular Based Hybrid Column Generation Approach. *Constraints*, 11(4):315–333, 2006.
8. M. Dincbas, H. Simonis, and P. Van Hentenryck. Solving the Car-Sequencing Problem in Constraint Logic Programming. In *ECAI*, pages 290–295, 1988.
9. I. P. Gent and T. Walsh. Csplib: a benchmark library for constraints, 1999.
10. M. J. Maher, N. Narodytska, C.-G. Quimper, and T. Walsh. Flow-Based Propagators for the SEQUENCE and Related Global Constraints. In *CP*, pages 159–174, 2008.
11. J. Menana and S. Demassey. Sequencing and Counting with the multicost-regular Constraint. In *CPAIOR*, pages 178–192, 2009.
12. G. Pesant. A Regular Language Membership Constraint for Finite Sequences of Variables. In *CP*, pages 482–495, 2004.
13. G. Pesant. Constraint-Based Rostering. In *PATAT*, 2008.
14. J.-C. Régin. Generalized Arc Consistency for Global Cardinality Constraint. In *AAAI*, pages 209–215(2), 1996.
15. J.-C. Régin and J.-F. Puget. A Filtering Algorithm for Global Sequencing Constraints. In *CP*, pages 32–46, 1997.
16. B.M. Smith. Succeed-first or Fail-first: A Case Study in Variable and Value Ordering, 1996.
17. C. Solnon, V. Cung, A. Nguyen, and C. Artigues. The car sequencing problem : Overview of state-of-the-art methods and industrial case-study of the ROADEF'2005 challenge problem. *EJOR*, 191:912–927, 2008.
18. W. J. van Hoeve, G. Pesant, L.-M.Rousseau, and A. Sabharwal. New Filtering Algorithms for Combinations of Among Constraints. *Constraints*, 14(2):273–292, June 2009.
19. W. J. van Hoeve, G. Pesant, L.-M. Rousseau, and Ashish Sabharwal. Revisiting the Sequence Constraint. In *CP*, pages 620–634, 2006.