# Explaining the ATMOSTSEQCARD constraint*

Mohamed Siala[1,2], Christian Artigues[1,3], and Emmanuel Hebrard[1,3]

[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, artigues, hebrard}@laas.fr

**Abstract.** We firstly propose an $O(n)$ propagation-based procedure for explaining the ATMOSTSEQCARD constraint. Then, we evaluate it against pure CP models for solving car-sequencing benchmarks. The experimental evaluation emphasizes the importance of combining CP with SAT for solving this problem.

## 1 Introduction

The ATMOSTSEQCARD constraint, recently proposed in [7], has been shown to be extremely efficient for solving Car-sequencing and Crew-rortering benchmarks. However, efficient propagators might be not enough for solving hard problems without powerful learning mechanisms. In this sense, several hybrid approaches combining the strengths of SAT and CP have been proposed. These methods are based on the notion of explanation. The idea is to keep some constraints in their implicit form (i.e. with a specific propagator) with the ability to explain themselves. In particular, whenever a *pruning/failure* is triggered by a constraint $C$, the latter should be able to generate a relevant *explanation*.

In this context, we introduce a linear time procedure for computing compact explanations for the ATMOSTSEQCARD constraint. This algorithm can be used in a hybrid CP/SAT approach such as SAT Modulo Theory, CSP Solvers with learning, or Lazy Clause Generation.

Our experimental evaluation provides good empirical evidence for the two following observations regarding the Car-sequencing problem: First, we emphasize the hypothesis that propagation, even with state-of-the-art heuristics, does not seem as important for proving unsatisability than clause learning. Second, as expected, specic CP heuristics are very useful to quickly find solutions. Moreover, they can be combined with adaptive heuristics (VSIDS) in order to keep clause learning strength for building proofs.

The remainder of this paper is organized as follows. We give in Section 2 a short background on the hybridisation of *CP* and *SAT*. In Section 3, we show that, based on the ATMOSTSEQCARD propagator, one can build a linear time explanation for this constraint. Finally, we empirically evaluate, in Section 4, the hybrid model based on this explanation against the related pure-CP one.

---

* Mohamed Siala is the student ; Dr. Artigues and Dr. Hebrard are the supervisors

## 2 Constraint Programming and its hybridisation with SAT

A *Constraint Satisfaction Problem* (*CSP*) is defined by a triplet $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ where $\mathcal{X}$ is a set of variables, $\mathcal{D}$ is a mapping of variables to finite sets of values and $\mathcal{C}$ is a set of constraints that specify allowed combinations of values for subsets of variables. We assume that $\mathcal{D}(x) \subset \mathbb{Z}$ for all $x \in \mathcal{X}$. We denote $x \leftarrow v$ the assignment of the value $v$ to the variable $x$, and $x \not\leftarrow v$ the pruning of the value $v$ from $\mathcal{D}(x)$. A *partial instantiation* $S$ is a set of assignements and/or pruning such that no variable is assigned more than one value and no value is pruned and assigned for the same variable. Let $\perp$ be a *failure* or a domain wipe-out, by convention equal to the set of all possible assignments and prunings. A constraint $C$ defines a relation $Rel(C)$, that is, a set of instantations, over the variables in $Scope(C)$. It is *generalized arc consistent* (GAC) iff, for every value $v$ of every variable $x$ in $Scope(C)$, there exists a consistent instantiation $S$ in $Rel(C)$ such that $x \leftarrow v \in S$. Throughout the paper we shall associate a constraint $C$ to a *propagator*, that is, a function mapping partial instantiations to partial instantiations or to the failure $\perp$. Given a partial instantiation $S$ we denote $C(S)$ the partial instantiation (or failure) obtained by applying the propagator associated to $C$ on $S$, and we have $S \subseteq C(S)$. Let $p$ be an assignment or a pruning. The level of $p$ (denoted by $lvl(p)$) is its order of appearance in the search tree. We say that the partial instantiation $S$ implies $p$ w.r.t $C$ iff $p \notin S$ & $p \in C(S)$. We denote $S(x)$ the domain $\mathcal{D}(x)$ updated by the assignment or pruning associated to $x$ in $S$. Moreover, we shall denote $\min(S(x))$ (resp. $\max(S(x))$) the minimum (resp. maximum) value in $S(x)$. We say that a partial instantiation $S$ is an *explanation* of the pruning $x \not\leftarrow v$ with respect to a constraint $C$ if it implies $x \not\leftarrow v$ (that is, $x \not\leftarrow v \in C(S) \setminus S$). Moreover, $S$ is a valid explanation iff $lvl(x \not\leftarrow v) > max(\{lvl(p) \mid p \in S\})$. For instance, if $C$ is the clause $p \vee \neg q \vee r$, the only possible explanation for $p \not\leftarrow 0$ with respect to $C$ is $\{q \leftarrow 1, r \leftarrow 0\}$.

The Boolean Satisfiability Problem (SAT) is a particular case of CSP where all the domains are $\{0, 1\}$ and constraints are clauses (i.e. disjunction over these variables and their negation). Modern SAT Solvers implement extremely efficient techniques like Conflict-Driven Clause Learning (CDCL), the two-Watched literals, activity-based heuristics, etc [1]. Several hybrid approaches trying to exploit these techniques into CP Solvers were proposed. The idea is to incorporate a SAT engine with finite domain propagators. For instance Katsirelos's generalized nogoods [3] [4] enable this type of approaches for arbitrary domains. However, to simulate the behavior of CDCL, it is necessary to *explain* either a failure or the pruning of a domain value. Lazy-clause generation [5] solvers add a clause whenever pruning is performed in order to provide an explanation for the pruning. In this paper we use a solver with a slightly different architecture, where constraints generate explanations as lazy as possible (i.e. when the learning scheme asks for explanations).

## 3 Explaining the AtMostSeqCard constraint

Let $\mathcal{X} = [x_1..x_n]$ be a be a sequence of Boolean variables, $u$, $q$ and $d$ be integer variables. The AtMostSeqCard constraint is defined as follows :

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

In [7], the authors proposed a $O(n)$ filtering algorithm achieving GAC on this constraint. We outline the main idea of the propagator. Let $S$ be a partial instantiation. The `leftmost` procedure returns an instantiation $\overrightarrow{w}_S \supseteq S$ of maximum cardinality by greedily assigning the value $1$ from left to right while respecting the AtMost constraints. Let $\overrightarrow{w}^i_S$ denotes the partial instantiation $\overrightarrow{w}_S$ at the begining of iteration $i$, and let $\overrightarrow{w}^1_S = S$. The value $\max_S(i)$ denotes the maximum minimum cardinality, with respect to the current domain $\overrightarrow{w}^i_S$, of the $q$ subsequences involving $x_i$. It is computed alongside $\overrightarrow{w}_S$ and will be useful to explain the subsequent pruning. It is formally defined as follows (where $\min(\overrightarrow{w}^i_S(x_k)) = 0$ if $k < 1$ or $k > n$):

$$\max_S(i) = \max_{j \in [1..u]} \left( \sum_{k=i-q+j}^{i+j-1} \min(\overrightarrow{w}^i_S(x_k)) \right)$$

**Definition 2.** *The outcome of the procedure* `leftmost` *can be recursively defined using* $\max_S$*: at each step i,* `leftmost` *adds the assignment* $x_i \leftarrow 1$ *iff this assignment is consistent with* $\overrightarrow{w}^i_S$ *and* $\max_S(i) < u$*, it adds the assignment* $x_i \leftarrow 0$ *otherwise.*

In order to express declaratively the full propagator, we need the following further steps: The same procedure is applied on variables in reverse order $[x_n..x_1]$, yielding the instantiation $\overleftarrow{w}_S$. We denote respectively $L_S(i)$ and $R_S(i)$ the sum of the values given by $\overrightarrow{w}_S$ (resp. $\overleftarrow{w}_S$) to the $i$ first variables (resp. $n - i + 1$ last variables). That is:

$$L_S(i) = \sum_{k=1}^{i} \min(\overrightarrow{w}_S(x_k)) \quad , \quad R_S(i) = \sum_{k=i}^{n} \min(\overleftarrow{w}_S(x_k))$$

Now we have all the tools to define the propagator associated to this constraint described in [7], and which is a conjunction of GAC on the AtMost constraints on each subsequence, of Cardinality constraint $\sum_{i=1}^{n} x_i = d$, and of the following:

$$\textsc{AtMostSeqCard}(S) = \begin{cases} S, & \text{if } \; L_S(n) > d \\ \bot, & \text{if } \; L_S(n) < d \\ S & \cup \; \{x_i \leftarrow 0 \mid S(x_i) = \{0,1\} \;\&\; L_S(i) + R_S(i) \leq d\} \\ & \cup \; \{x_i \leftarrow 1 \mid S(x_i) = \{0,1\} \;\&\; L_S(i-1) + R_S(i+1) < d\} \text{ otherwise} \end{cases} \tag{3.1}$$

If a failure/pruning is detected by the Cardinality or an AtMost constraint, then it is easy to give an explanation. However, if a failure or a pruning is due to the propagator defined in equation 3.1, then we need to specify how to generate a relevant explanation. We start by giving an algorithm explaining failure then show how to use it to explain pruning. We suppose in the following that failure/pruning was fired at level $l$.

### 3.1 Explaining Failure

The original instantiation $S$ would be a possible naive explanation expressing this failure. We propose in the flowing a procedure returning more compact explanations with no grantee regarding the optimality (see later example 1).

Let $I = [x_{k+1}..x_{k+q}]$ be a (sub)sequence of variables of size $q$ and $S$ be a partial instantiation. We denote $card(I, S)$ the minimum cardinality of $I$ under the instantiation $S$, that is: $card(I, S) = \sum_{x_i \in I} \min(S(x_i))$.

**Lemma 1.** *If* $S^* = S \setminus (\{x_i \leftarrow 0 \mid max_S(i) = u\} \cup \{x_i \leftarrow 1 \mid max_S(i) \neq u\})$ *then* $\overrightarrow{w}_S = \overrightarrow{w}_{S^*}$.

*Proof.* Suppose that there exists an index $i \in [1..n]$ s.t. $\overrightarrow{w}_S(x_i) \neq \overrightarrow{w}_{S^*}(x_i)$ and let $k$ be the smallest index verifying this property. Since the instantiation $S^*$ is a subset of $S$ (i.e., $S^*$ is weaker than $S$) and since `leftmost` is a greedy procedure assigning the value 1 whenever possible from left to right, it follows that $\overrightarrow{w}_S(x_k) = 0$ and $\overrightarrow{w}_{S^*}(x_k) = 1$. Moreover, it follows that $\max_S(k) = u$ and $\max_{S^*}(k) < u$. In other words, there exists a subsequence $I$ containing $x_k$ s.t the cardinality of $I$ in $\overrightarrow{w}^k_S$ ($card(I, \overrightarrow{w}^k_S)$) is equal to $u$, and the cardinality of $I$ in $\overrightarrow{w}^k_{S^*}$ ($card(I, \overrightarrow{w}^k_{S^*})$) is less than $u$. From this we deduce that there exists a variable $x_j \in I$ such that $\min(\overrightarrow{w}^k_S(x_j)) = 1$ and $\min(\overrightarrow{w}^k_{S^*}(x_j)) = 0$.

First, we cannot have $j < k$. Otherwise, both instantiations $\overrightarrow{w}^k_S(x_j)$ and $\overrightarrow{w}^k_{S^*}(x_j)$ contain an assignment for $x_j$, and therefore we have $\overrightarrow{w}^k_S(x_j) = \{1\}$ and $\overrightarrow{w}^k_{S^*}(x_j) = \{0\}$, which contradicts our hypothesis that $k$ is the smallest index of a discrepancy.

Second, suppose now that $j > k$. Since we have $card(I, \overrightarrow{w}^k_S) = u$, we can deduce that $card(I, \overrightarrow{w}^j_S) = u$. Indeed, when going from iteration $k$ to iteration $j$, `leftmost` only adds assignments, and therefore $card(I, \overrightarrow{w}^j_S) \geq card(I, \overrightarrow{w}^k_S)$. It follows that $\max_S(j) = u$, and by construction of $S^*$, we cannot have $x_j \leftarrow 1 \in S \setminus S^*$. However, it contradicts the fact that $\min(\overrightarrow{w}^k_S(x_j)) = 1$ and $\min(\overrightarrow{w}^k_{S^*}(x_j)) = 0$.

$\square$

**Theorem 1.** *If $S$ is a valid explanation for a failure and* $S^* = S \setminus (\{x_i \leftarrow 0 \mid max_S(i) = u\} \cup \{x_i \leftarrow 1 \mid max_S(i) \neq u\})$*, then $S^*$ is also a valid explanation.*

*Proof.* By Lemma 1, we know that the instantiations $\overrightarrow{w}_S$ and $\overrightarrow{w}_{S^*}$, computed from, respectively the instantiations $S$ and $S^*$ are equal. In particular, we have $L_S(n) = L_{S^*}(n)$ and therefore ATMOSTSEQCARD($S$) $= \perp$ iff ATMOSTSEQCARD($S^*$) $= \perp$.

$\square$

Theorem 1 gives us a linear time procedure to explain failure. In fact, all the values $\max_S(i)$ can be generated using one call of `leftmost`.

*Example 1.*

| | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $S$ | 1 | 0 | 1 | 0 | 0 | . | . | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| $max_S(i)$ | 2 | **2** | 2 | **2** | **2** | 1 | 2 | **2** | **2** | **2** | 2 | 2 | **2** | **2** | **2** | 1 | **1** | 1 | 1 | 1 | 1 | **1** |
| $\overrightarrow{w}_S(x_i)$ | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| $L_S(i)$ | 1 | 1 | 2 | 2 | 2 | 3 | 3 | 3 | 3 | 3 | 4 | 5 | 5 | 5 | 5 | 5 | 6 | 6 | 6 | 6 | 6 | 7 |
| $S^*$ | 1 | . | 1 | . | . | . | . | . | . | . | 1 | 1 | . | . | . | 0 | . | 0 | 0 | 0 | 0 | . |
| $max_{S^*}(i)$ | 2 | 2 | 2 | 2 | 2 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $\overrightarrow{w}_{S^*}(x_i)$ | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |

We illustrate in Example 1 the explanation of a failure on ATMOSTSEQCARD$(2, 5, 8, [x_1..x_{22}])$. The propagator returns a failure since $L_S(22) = 7 < d = 8$. The default explanation corresponds to the set of all the assignments in this sequence, whereas our procedure shall generate a more compact explanation by considering only the assignments in $S^*$. Bold face values in the $max_S(i)$ line represent the variables that will not be included in $S^*$. As a result, we reduce the size of the default explanation from 20 to 9 assignments. Note, however, that $S^*$ is not optimal (w.r.t the size) since `leftmost` returns exactly the same result on $S^*$ and $S^* \setminus x_1 \leftarrow 1$ (hence the same failure).

### 3.2 Explaining Pruning

Suppose that a pruning $x_i \not\leftarrow v$ was triggered by the propagator in equation 3.1 at a given level $l$ on $S$ (i.e. propagating ATMOSTSEQCARD(S) implies $x_i \not\leftarrow v$). Consider the partial instantiation $S_{x_i \leftarrow v}$ identical to $S$ on all assignments at level $l$ except for $x_i \leftarrow v$ instead of $x_i \not\leftarrow v$. By construction $S_{x_i \leftarrow v}$ is unsatisfiable. Let $S^*$ be the explanation expressing this failure using the previous mechanism. We have then $S^* \setminus x_i \leftarrow v$ as a valid explanation for the pruning $x_i \not\leftarrow v$.

## 4 Experimental results

To evaluate the proposed explanation, we compare it against pure CP models for solving the car-sequencing problem [9]. In the latter, a set of vehicles has to be sequenced in an assembly line. Each class of cars requires a set of options. However, the working station handling a given option can only mount it on a fraction of the cars passing on the line. Each option $j$ is thus associated with a fractional number $u_j/q_j$ standing for its capacity (at most $u_j$ cars with option $j$ occur in any sub-sequence of length $q_j$). We used the benchmarks available from the CSPLib [2]. We grouped the instances into three categories `sat[easy]` (74 instances), `sat[hard]` (7 instances) and `unsat` (28 instances). 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. We ran the following methods (all using the ATMOSTSEQCARD propagator):

- Mistral as a hybrid CP/SAT solver implementing standard CDCL features and using the proposed explanation. We tested four branching heuristics :
  1. *hybrid (VSIDS)* uses VSIDS;
  2. *hybrid (Slot)* uses a cp heuristic based on the usage rate[8].
  3. *hybrid (Slot → VSIDS)* first uses *hybrid (Slot)* then switches after 100 non-improving restarts to VSIDS.
  4. *hybrid (VSIDS → Slot)* uses VSIDS and switches after 100 non-improving restarts to *hybrid (Slot)*.
- *pure-CP*: Mistral without clause learning using the $Slot$ branching.

For each considered data set, we report the total number of successful runs (*#suc*).Then, we report the number of fails (*fails*) and the CPU time (*time*) in seconds both averaged over all successful random runs on every instance. We emphasize the statistics of the best method (w.r.t. *#suc*) for each data set using bold face fonts.

*Finding solutions quickly:* We observe that pure CP approaches are difficult to outperform. It must be noticed that the results reported here are significantly better than those previously reported for similar approaches. For instance, the best methods introduced in [10] take several seconds on most instances of the first category and were not able to solve two of them within a one hour time cutoff. Moreover in [7], the same solver on the same model had a similar behavior on the first category (`sat[easy]`), however was only able to solve 2 instances of the second category (`sat[hard]`). The only difference with the method we ran in this paper is that restarts according to the Luby sequence were used. However, overall, the best method on satisfiable instances is the hybrid solver using a pure CP heuristic. This study shows that propagation is very important to find solution quickly when they exist, by keeping the search “on track” and avoiding exploring large unsatisfiable subtrees. In fact, previous CP approaches that did not enforce GAC on the ATMOSTSEQCARD constraint are all dominated by *pure-CP*.

*Proving unsatisfiability:* The hybrid models are far better than pure CP approach that was not able to prove any case of unsatisfiability. To mitigate this observation, however, notice that other CP models with strong filtering, using the Global Sequencing Constraint [6], or a conjunction of this constraint and ATMOSTSEQCARD [7, 10] were able to build proofs for some of the 5 known unsatisfiable instances of the CSPLib. However, these models were not sufficient to solve any of the 23 larger unsatisfiable instances. These results clearly show that clause learning is by far the most critical factor.

Table 1: Experimental Evaluation

| Method | `sat[easy]` ($74 \times 5$) | | | `sat[hard]` ($7 \times 5$) | | | `unsat` ($28 \times 5$) | | |
|---|---|---|---|---|---|---|---|---|---|
| | #suc | avg fails | time | #suc | avg fails | time | #suc | avg fails | time |
| *hybrid (VSIDS)* | 370 | 903 | 0.23 | 16 | 207211 | 286.32 | 35 | 177806 | 224.78 |
| *hybrid (VSIDS → Slot)* | 370 | 739 | 0.23 | 35 | 76256 | 64.52 | **37** | **204858** | **248.24** |
| *hybrid (Slot → VSIDS)* | 370 | 132 | 0.04 | 34 | 4568 | 2.50 | 37 | 234800 | 287.61 |
| *hybrid (Slot)* | 370 | 132 | 0.04 | **35** | **6304** | **3.75** | 23 | 174097 | 299.24 |
| *pure-CP* | **370** | **43.06** | **0.03** | 35 | 57966 | 16.25 | 0 | - | - |

## 5 Conclusion

We proposed a linear time procedure for explaining the ATMOSTSEQCARD constraint and empirically evaluate it with car-sequencing benchmarks. Experimental results emphasize the importance of advanced propagation as well as built-in heuristics for searching feasible solutions and of clause learning for building unsatisfiability proof for this problem.

## References

1. A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.
2. I. P. Gent and T. Walsh. CSPlib: a benchmark library for constraints, 1999.
3. G. Katsirelos. *Nogood Processing in CSPs*. PhD thesis, University of Toronto, 2008.
4. G. Katsirelos and F. Bacchus. Generalized NoGoods in CSPs. In *Proceedings of AAAI*, pages 390–396, 2005.
5. Olga. Ohrimenko, P-J. Stuckey, and M. Codish. Propagation via Lazy Clause Generation. *Constraints*, 14(3):357–391, 2009.
6. J-C Régin and J-F Puget. A Filtering Algorithm for Global Sequencing Constraints. In *Proceedings of CP*, pages 32–46, 1997.
7. M. Siala, E. Hebrard, and M-J. Huguet. An Optimal Arc Consistency Algorithm for a Chain of Atmost Constraints with Cardinality. In *Proceedings of CP*, pages 55–69, 2012.
8. B. Smith. Succeed-first or Fail-first: A Case Study in Variable and Value Ordering, 1996.
9. C. Solnon, V-D. Cung, A-N, and C Artigues. The car sequencing problem: Overview of state-of-the-art methods and industrial case-study of the ROADEF'2005 challenge problem. *European Journal of Operational Research*, 191:912–927, 2008.
10. 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, 2009.