SAT and Hybrid Models of the Car Sequencing Problem

Christian Artigues, Emmanuel Hébrard, Valentin Mayer-Eichberger,
Mohamed Siala, Toby Walsh

SAT and Hybrid models of the Car Sequencing problem

Christian Artigues[1,2], Emmanuel Hebrard[1,2], Valentin Mayer-Eichberger[3,4], Mohamed Siala[1,5], and Toby Walsh[3,4]

[1] CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
[2] Univ de Toulouse, LAAS, F-31400 Toulouse, France
[3] NICTA
[4] University of New South Wales
[5] Univ de Toulouse, INSA, LAAS, F-31400 Toulouse, France
{artigues, hebrard, siala}@laas.fr
{valentin.mayer-eichberger, toby.walsh}@nicta.com.au

Abstract. We compare both pure SAT and hybrid CP/SAT models for solving car sequencing problems, and close 13 out of the 23 large open instances in CSPLib. Three features of these models are crucial to improving the state of the art in this domain. For quickly finding solutions, advanced CP heuristics are important and good propagation (either by a specialized propagator or by a sophisticated SAT encoding that simulates one) is necessary. For proving infeasibility, clause learning in the SAT solver is critical. Our models contain a number of novelties. In our hybrid models, for example, we develop a linear time mechanism for explaining failure and pruning the AtMostSeqCard constraint. In our SAT models, we give powerful encodings for the same constraint. Our research demonstrates the strength and complementarity of SAT and hybrid methods for solving difficult sequencing problems.

1 Introduction

In the car sequencing problem [26], we sequence a set of vehicles along an assembly line. Each class of cars requires a set of options. However, the working station handling a given option can only deal with 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). Several global constraints have been proposed in the Constraint Programming (CP) literature to model this family of sequence constraints. At present, CP models with the AtMostSeqCard constraint [23] or its combination with the *Global Sequencing Constraint (*Gsc*)* [20] have provided the best performance. However, pure CP approaches suffer when we consider proving unsatisfiability. The goal of this paper is to show that by exploiting Boolean-Satisfiability (SAT), we can improve upon the state of the art in this domain. We are able to close 13 out the 23 large open instances in CSPLib.

We propose several approaches combining ideas from SAT and CP for solving the car sequencing problem. First, we capture CP propagation in SAT by a careful formulation of the problem into conjunctive normal form (CNF). We propose a family of pure SAT encodings for this problem and relate them to existing encoding techniques. They

are based on an extension of Sinz's encoding for the CARDINALITY constraint [24] and have similarities to the decomposition of the GEN-SEQUENCE constraint given in [2]. Second, 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 or lazy clause generating solver, where non-clausal constraints need a propagator and an explanation algorithm. In principle, the hybrid approach has access to all the advances from the SAT world, whilst benefiting from constraint propagation and dedicated branching heuristics from CP. However, our experiments reveal that in practice, SAT solvers maintain an edge when proving unsatisfiability. Due to the most up to date data structures and tuning of parameters for literal activity and clause deletion, encoding into SAT significantly outperforms the hybrid approach on hard unsatisfiable instances.

We made three observations based on these experiments: First, CP heuristics are good at quickly finding solutions. Whilst generic activity based heuristics are surprisingly robust, dedicated CP can be much faster. Second, propagation, either through finite domain propagators, or through unit propagation via a "strong" encoding, is extremely important for finding solutions reliably on harder instances. Strong propagation makes it less likely to enter an unsatisfiable subproblem during search. In conjunction with this, restarting ensures that these unlikely cases do not matter. Third, clause learning is critical for proving unsatisfiability. In this respect, the approaches that we introduce (especially the SAT encodings) greatly improve the state of the art for the car sequencing problem. Moreover, counter-intuitively, it does not seem that constraint propagation of the ATMOSTSEQCARD constraint nor the "strength" of the SAT encoding, has a significant impact on the ability of the solver to prove unsatisfiability.

The remainder of this paper is organized as follows. In Section 2, we give some background on CP, SAT and their hybridization. In Section 3, we recall the state of the art CP models for this problem and show the connection with SAT. In Section 4, we show that how to build explanations for the ATMOSTSEQCARD constraint based on its propagator. Then, we present advanced SAT encodings for this constraint in Section 5. Finally, in Section 6, we compare experimentally the approaches we introduce against pure CP and pseudo Boolean models.

2 Background

Constraint Programming. A constraint network 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, that is the restriction of its domain $\mathcal{D}(x)$ to $\{v\}$, similarly, we denote $[x \nleftarrow v]$ the pruning of the value v from $\mathcal{D}(x)$. A *partial instantiation* S is a set of assignments 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. On finite domains, we can consider a *closure* of partial instantiations with respect to initial domains. That is, if the assignment $[x \leftarrow v]$ belongs to S, we also

assume that $[x \nleftarrow v]$ for all $v \in \mathcal{D}(x) \setminus v$ belong to S. Similarly, if all but one of the values are pruned, the remaining value is added as an assignment. This is similar to *expanded* solutions in [16]. However, we shall restrict ourselves to Boolean domains in this paper. We therefore have $S \subseteq S'$ iff S' is a stronger (tighter) partial instantiation than S. Given an initial domain $\mathcal{D}$ and a partial instantiation S, we can derive a current domain taking into account the pruning and assignments of S in $\mathcal{D}$. There will not be ambiguities about the original domains, therefore we simply denote $S(x)$ the domain $\mathcal{D}(x)$ updated by the assignment or pruning associated to x in S.

A constraint C defines a relation $Rel(C)$, that is, a set of instantations, over the variables in $Scope(C)$. Let S be a partial instantiation of $Scope(C)$. The constraint C is said *generalized arc consistent* (GAC) with respect to S iff, for every x in $Scope(C)$ and every $v \in S(x)$, there exists an instantiation $T \in Rel(C)$ such that $[x \nleftarrow v] \notin T$ (T is a support) and $T \subseteq S$ (T is valid). We say that a constraint is *dis-entailed* with respect to a partial instantiation S iff there is no T in $Rel(C)$ such that $S \subseteq T$.

Throughout the paper we shall associate a *propagator* with a constraint C. This is a function mapping partial instantiations to partial instantiations or to the failure $\bot$. Given a partial instantiation S, we denote $C(S)$ for the partial instantiation (or failure) obtained by applying the propagator associated to C on S, with $S \subseteq C(S)$. We say that S implies the assignment or pruning p with respect to C iff $p \notin S \ \& \ p \in C(S)$. Finally, the *level* of an assignment or a pruning p is the order of appearance of the assignment (respectively pruning) in the search tree, and we denote it $lvl(p)$. For a comprehensive introduction to CP solving in general and its techniques we refer to [21].

SAT-Solving. The Boolean Satisfiability problem (SAT) is a particular case of CSP where domains are Boolean and constraints are only clauses (disjunction of literals). A SAT solver computes a satisfying instantiation of a formula of propositional logic in conjunctive normal form (CNF) or proves that no such instantiation exists. The most widely used method to solve SAT problems is based on the DPLL algorithm ([8]), which is a depth first search with backtracking and unit propagation. Unit propagation (UP) prunes the assignment of the remaining literal in a clause when all other literals have become false. An important improvement to the DPLL algorithms is Conflict-Driven Clause Learning (CDCL). A CDCL solver records for each conflict an appropriate reason in form of a clause, and adds it to the clause database. This can potentially prune unseen parts of the search tree. Furthermore, SAT solvers are equipped with robust domain-independent branching and decision heuristics (for instance VSIDS [17]). For a comprehensive introduction to SAT solving in general and its techniques we refer to [4].

Modelling in CNF is a crucial step for the success of solving problems with SAT. A natural approach to find a good SAT model is to describe the problem with higher level constraints and then translate these constraints to CNF. In accordance with this methodology, the representation of integer domains and encodings of a variety of global constraints have been proposed and analyzed [2, 13, 28]. Similarly the notion of GAC has been adapted to SAT. UP is said to maintain GAC on the CNF encoding of a constraint if it forces all assignments to the variables representing values that must be set to avoid unsatisfiability of the constraint. The quality of an encoding into SAT is measured by both its size and its level of consistency by UP. Moreover, we must taken into

account that SAT solvers cannot distinguish between the original variables and any auxiliary variables introduced to produce a more compact encoding. Thus, when aiming for a good CNF encoding, we must consider how such auxiliary variables might increase propagation.

2.1 Hybrid CP/SAT

The notion of nogood learning in constraint programming is not new, in fact it predates [22] similar concepts in SAT. However, CDCL learns and uses nogoods in a particular way, and such methods have been reintroduced into CP. For instance Katsirelos's generalized nogoods [15] [16] enable nogood learning with arbitrary domains. Another complexity is that propagation is now not restricted to unit propagation. A given constraint can be associated with a specific propagator. However, to perform clause learning, it is necessary to *explain* either a failure or the pruning of a domain value.We say that a partial instantiation S is an *explanation* of the pruning $[x \nleftarrow v]$ with respect to a constraint C if it implies $[x \nleftarrow v]$ (that is, $[x \nleftarrow v] \in C(S) \setminus S$). Moreover, S is a valid explanation iff $lvl([x \nleftarrow v]) > max(\{lvl(p) \mid p \in S\})$.

In this paper we use a solver with an architecture similar to *Lazy Clause Generation* with backward explanations [12]. In addition to a propagator, an explanation algorithm is associated with each constraint. However, as opposed to explanation based constraint programming [6, 7], the explanations are used exactly as in CDCL, i.e., literals are replaced by their explanation until the current nogood contains a Unique Implication Point of the current level's pruning. In this sense it is very close to the way some Pseudo-Boolean CDCL solvers, such as PBS [1], PBChaff [9] or SAT4JPseudo [3] integrate unit propagations on clauses, dedicated propagators and explanations (cutting planes) for linear equations. On Boolean domains, the hybrid SAT/CP approach we use works as follows:

Propagation: The propagation is performed by a standard CP engine, except that for each pruned value we record the constraint responsible for this pruning (a simple pointer to the constraint is stored). Both original and learned clauses are handled by a dedicated propagator simulating the behavior of a clause base (i.e., using watched literals).

Learning: When a failure is raised, the standard CDCL conflict analysis algorithm is used. The constraint C responsible for the failure is asked to provide an explanation for this failure. The literals of this explanation form the base nogood Ng. Subsequently, any assignment $[x \leftarrow v]$ such that $lvl([x \leftarrow v]) \geq lvl(d)$ where d is the last decision, is removed from Ng and replaced by its explanation by the constraint marked as responsible for it. This process continues until Ng has a Unique Implication Point.

Search: Since a CP library (Mistral[1]) was used to implement this approach, it is possible to use hand made CP heuristics as well as built-in strategies such as VSIDS. However, as in CDCL algorithms, branching decisions are not refuted in a "right" branch. Instead, we backjump to the second highest level of literals in the learned clauses, and unit propagation for this clause is triggered.

[1] https://github.com/ehebrard/Mistral-2.0

3 The *Car Sequencing* problem

In the car sequencing problem, n vehicles have to be produced on an assembly line. There are c classes of vehicles and m types of options. Each class $k \in \{1, \ldots, c\}$ is associated with a demand D_k^{class}, that is, the number of occurrences of this class on the line, and a set of options $\mathcal{O}_k \subseteq \{1, \ldots, m\}$. Each option is handled by a working station able to process only a fraction of the vehicles passing on the line. The capacity of an option j is defined by two integers u_j and q_j, such that no subsequence of size q_j may contain more than u_j vehicles requiring option j. A solution of the problem is then a sequence of cars satisfying both demand and capacity constraints. For convenience, we shall also define, for each option j, the corresponding set of classes of vehicles requiring this option $\mathcal{C}_j = \{k \mid j \in \mathcal{O}_k\}$, and the option's demand $D_j = \sum_{k \in \mathcal{C}_j} D_k^{class}$.

3.1 CP Modelling

As in a standard CP Model, we use two sets of variables. The first set corresponds to n integers $\{x_1, \ldots, x_n\}$ taking values in $\{1, \ldots, c\}$ and standing for the class of vehicles in each slot of the assembly line. The second set of variables corresponds to nm Boolean variables $\{o_1^1, \ldots, o_n^m\}$, where o_i^j stands for whether the vehicle in the i^{th} slot requires option j. For the constraints, we distinguish three sets:

1. *Demand constraints*: for each class $k \in \{1..c\}$, $|\{i \mid x_i = k\}| = D_k^{class}$. This constraint is usually enforced with a Global Cardinality Constraint (GCC) [19] [18].
2. *Capacity constraints*: for each option $j \in \{1..m\}$, we post the constraint ATMOSTSEQCARD$(u_j, q_j, D_j, [o_1^j..o_n^j])$ using the propagator introduced in [23].
3. *Channelling*: Finally, we channel integer and Boolean variables: $\forall j \in \{1, ..., m\}, \forall i \in \{1, ..., n\}$, $o_i^j = 1 \Leftrightarrow x_i \in \mathcal{C}_j$

3.2 Default Pseudo-Boolean and SAT Models

The above CP Model can be easily translated into a pseudo Boolean model since the majority of the constraints are sum expressions. We use the same Boolean variables o_i^j standing for whether the vehicle in the i^{th} slot requires option j. Moreover, the class variables are split into nc Boolean variables c_i^j standing for whether the ith vehicle is of class j. We have the same constraints as in the CP model, albeit expressed slightly differently. Moreover, we need to constrain variables standing for class assignment of the same slot to be mutually exclusive.

1. *Demand constraints*: $\forall j \in [1..c], \sum_i c_i^j = D_j$
2. *Capacity constraints*: $\sum_{l=i}^{i+q_j-1} o_l^j \leq u_j, \forall i \in \{1, \ldots, n - q_j + 1\}$
3. *Channelling*:
   - $\forall i \in [1..n], \forall l \in [1..c]$, we have:
     - $\forall j \in \mathcal{O}_l, \overline{c_i^l} \vee o_i^j$
     - $\forall j \notin \mathcal{O}_l, \overline{c_i^l} \vee \overline{o_i^j}$

– For better propagation, we add the following redundant clause:
$\forall i \in [1..n], j \in [1..m], \overline{o_i^j} \vee \bigvee_{l \in \mathcal{C}_j} c_i^l$

4. *Domain constraints*: a vehicle belong to only one class: $\forall i \in [1..n], \sum_j c_i^j = 1$

A SAT Encoding for this problem could translate each sum constraint (in this case only CARDINALITY constraints) into a CNF formula. We will show in Section 5 how such a translation can be improved.

4 Explaining the ATMOSTSEQCARD constraint

We present here an algorithm explaining the ATMOSTSEQCARD constraint. This algorithm is based on the propagator for this constraint, which we therefore now recall. Let $[x_1, x_2..x_n]$ be a sequence of Boolean variables, u, q and d be integer variables. The ATMOSTSEQCARD constraint is defined as follows:

Definition 1.

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

In [23], the authors proposed a $O(n)$ filtering algorithm achieving AC on this constraint. We outline the main idea of the propagator.

Let $\mathcal{X} = [x_1..x_n]$ be a sequence of variables, and S a partial instantiation over these variables. The procedure `leftmost` 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 $\overrightarrow{w}_S^i$ denote the partial instantiation $\overrightarrow{w}_S$ at the beginning of iteration i, and let $\overrightarrow{w}_S^1 = S$. The value $\max_S(i)$ denotes the maximum minimum cardinality, with respect to the current domain $\overrightarrow{w}_S^i$, of the q subsequences involving x_i. It is computed alongside $\overrightarrow{w}_S$ and will be useful to explain the subsequent pruning/failure. It is formally defined as follows (where $\min(\overrightarrow{w}_S^i(x_k)) = 0$ if $k < 1$ or $k > n$):

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

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* $\overrightarrow{w}_S^i$ *and* $\max_S(i) < u$*, it adds the assignment* $[x_i \leftarrow 0]$ *otherwise.*

Example 1. For instance, consider the execution of the procedure `leftmost` on the constraint ATMOSTSEQCARD$(2, 4, 6, [x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}])$. We suppose that we start from the partial instantiation $\{[x_2 \leftarrow 0], [x_6 \leftarrow 1], [x_8 \leftarrow 0]\}$. Initially, we have the following structures, for each i representing an iteration (and also the index of a variable):

	1	2	3	4	5	6	7	8	9	10
$\overrightarrow{w}^1_S(x_i)$	$\{0,1\}$	0	$\{0,1\}$	$\{0,1\}$	$\{0,1\}$	1	$\{0,1\}$	0	$\{0,1\}$	$\{0,1\}$
$\overrightarrow{w}^2_S(x_i)$	1	0	$\{0,1\}$	$\{0,1\}$	$\{0,1\}$	1	$\{0,1\}$	0	$\{0,1\}$	$\{0,1\}$
$\overrightarrow{w}^3_S(x_i)$	1	0	$\{0,1\}$	$\{0,1\}$	$\{0,1\}$	1	$\{0,1\}$	0	$\{0,1\}$	$\{0,1\}$
$\overrightarrow{w}^4_S(x_i)$	1	0	1	$\{0,1\}$	$\{0,1\}$	1	$\{0,1\}$	0	$\{0,1\}$	$\{0,1\}$
...										
$\overrightarrow{w}^{11}_S(x_i)$	1	0	1	0	0	1	1	0	0	1
$\max_S(i)$	0	1	1	2	2	1	1	2	2	1

The partial solution $\overrightarrow{w}^1_S$ is equal to S. Then at each step i, `leftmost` adds the assignment $[x_i \leftarrow 1]$ or $[x_i \leftarrow 0]$ according to Definition 2. For instance, at the begining of step 4, the subsequences to consider are $[x_1, x_2, x_3, x_4]$, $[x_2, x_3, x_4, x_5]$, $[x_3, x_4, x_5, x_6]$ and $[x_4, x_5, x_6, x_7]$, of cardinality 2, 1, 2 and 1, respectively, with respect to the instantiation $\overrightarrow{w}^4_S(x_i)$. The value of $\max_S(4)$ is therefore 2.

To detect failure, we simply need to run this procedure and check that the final cardinality of $\overrightarrow{w}_S$ is greater than or equal to the demand d. We shall see that we can explain pruning by using essentially the same procedure.

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 $\overleftarrow{w}_S$. Observe that the returned instantiations $\overrightarrow{w}_S$ and $\overleftarrow{w}_S$ assign every variable in the sequence to either 0 or 1. 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 can define the propagator associated to the constraint AtMostSeqCard described in [23], and which is a conjunction of GAC on the AtMost (i.e. $\sum_{l=1}^{q} x_{i+l} \leq u$) constraints on each subsequence, of Cardinality constraint $\sum_{i=1}^{n} x_i = d$, and of the following:

$$\text{AtMostSeqCard}(S) = \begin{cases} S, & \text{if } L_S(n) > d \\ \bot, & \text{if } L_S(n) < d \\ S & \cup \{[x_i \leftarrow 0] \mid S(x_i) = \{0,1\} \\ & \quad \& \; L_S(i) + R_S(i) \leq d\} \\ & \cup \{[x_i \leftarrow 1] \mid S(x_i) = \{0,1\} \\ & \quad \& \; L_S(i-1) + R_S(i+1) < d\} \text{ otherwise} \end{cases} \tag{4.1}$$

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 4.1, then we need to specify how to generate a relevant explanation. We start by giving an algorithm explaining a failure. We show after that how to use this algorithm to explain pruning.

4.1 Explaining Failure

Suppose that the propagator detects a failure at a given level l. The original instantiation S would be a possible naive explanation expressing this failure. We propose in the following a procedure generating more compact explanations.

In example 2, the instantiation $S = \{[x_1 \leftarrow 1], [x_3 \leftarrow 0], [x_6 \leftarrow 0]\}$ is subject to AtMostSeqCard$(2, 5, 3, [x_1..x_6])$. S is unsatisfiable since $L_S(6) < d$. Consider now the sequence $S^* = \{[x_6 \leftarrow 0]\}$. The result of `leftmost` on S and S^* is identical. Therefore, S^* and S are both valid explanations for this failure, however S^* is shorter. The idea behind our algorithm for computing shorter explanations is to characterise which assignments will have no impact on the behavior of the propagator, and thus are not necessary in the explanation.

Example 2.

S	1	.	0	.	.	0
$\overrightarrow{w}(S)$	1	1	0	0	0	0
$max(S)$	1	1	2	2	2	1
$L(S)$	1	2	2	2	2	2
					$d = 3$	
					$L(6) = 2$	
					$\rightarrow$ Failure	

S^*	.	.	.	.	.	0
$\overrightarrow{w}(S^*)$	1	1	0	0	0	0
$max(S^*)$	1	1	2	2	2	1
$L(S^*)$	1	2	2	2	2	2
					$d^* = 3$	
					$L^*(6) = 2$	
					$\rightarrow$ Failure	

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 $\overrightarrow{w}^k_S$ (i.e. $card(I, \overrightarrow{w}^k_S)$) 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 $\min(\overrightarrow{w}^k_{S^*}(x_j)) = 0$.

□

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$(S) = \bot$ iff ATMOSTSEQCARD$(S^*) = \bot$. □

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 3 illustrates the explanation procedure.

Example 3.

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 here 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 explanation from 20 to 9.

Observe, however, that the generated explanation is not minimal. Take for instance the assignment $[x_1 \leftarrow 1]$. Despite it does not meet Theorem 1 conditions (i.e. $max_S(1) = u$), the set of assignments $S^* \setminus [x_1 \leftarrow 1]$ is a valid explanation since `leftmost` would return the same result between S^* and $S^* \setminus [x_1 \leftarrow 1]$.

4.2 Explaining Pruning

Suppose that a pruning $[x_i \nleftarrow v]$ was triggered by the propagator in equation 4.1 at a given level l on S (i.e. propagating ATMOSTSEQCARD(S) implies $[x_i \nleftarrow v]$). 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 $[x_i \nleftarrow v]$. 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 $[x_i \nleftarrow v]$.

5 SAT-Encoding for the ATMOSTSEQCARD constraint

In this section we present several SAT-encodings for the ATMOSTSEQCARD constraint and relate them to existing encoding techniques. First we describe a translation of Boolean cardinality constraints by a variant of the sequential counter encoding [24]. This encoding can be used to translate the decomposition of ATMOSTSEQCARD into CARDINALITY and ATMOST. Then we introduce an encoding taking advantage of the globality of ATMOSTSEQCARD by reusing the auxiliary variables for the cardinality constraint and integrating the sequence of ATMOST constraints. Finally, we combine the two encodings and prove that in this case UP maintains GAC on ATMOSTSEQCARD.

5.1 Sequential Counter

We describe first a translation of the cardinality expression $l \leq \sum_{i\in[1..n]} x_i \leq u$ to CNF by a sequential counter where $l, u \in \mathbb{N}$ and $x_i \in \{0, 1\}$. For technical reasons we use an additional variable x_0 s.t. $D(x_0) = \{0\}$.

- Variables:
  - $s_{i,j}$: $\forall i \in [0..n], \forall j \in [0..u+1]$, $s_{i,j}$ is *true* iff $|x_k; s.t. D(x_k) = \{1\}| \geq j$
- Encoding: $\forall i \in [1..n]$
  - Clauses for restrictions on the same level: $\forall j \in [0..u+1]$
    1. $\neg s_{i-1,j} \vee s_{i,j}$
    2. $x_i \vee \neg s_{i,j} \vee s_{i-1,j}$
  - Clauses for increasing the counter, $\forall j \in [1..u+1]$
    3. $\neg s_{i,j} \vee s_{i-1,j-1}$
    4. $\neg x_i \vee \neg s_{i-1,j-1} \vee s_{i,j}$
  - Initial values for the bounds of the counter:
    5. $s_{0,0} \wedge \neg s_{0,1} \wedge s_{n,l} \wedge \neg s_{n,u+1}$

In the rest of the section we refer to the clauses by numbers 1 to 5. The intuition is that the variables $s_{i,j}$ represent the bounds for cumulative sums of the sequence $x_1 \dots x_i$. The encoding is best explained by visualising $s_{i,j}$ as a two dimensional grid with positions (horizontal) and cumulative sums (vertical). The binary clauses 1 and 3 ensure that the counter (i.e. the variables representing the cumulative sums) is monotonically increasing. Clauses 2 and 4 control the interaction with the variables x_i. If x_i is true, then the counter has to increase at position i whereas if x_i is false an increase is prevented at position i. The conjunction 5 sets the initial values for the counter to start counting at 0 and ensures that the partial sum at position n is between to l and u.

Example 4. We illustrate the auxiliary variables: Given a sequence of 8 variables and $l = u = 2$. To the left the initial condition of the variables, followed assigning x_2 to true and then to the right x_7 to true.

3	0 0 0 0 0 0 0 0 0	3	0 0 0 0 0 0 0 0 0
2	0 0 1	2	0 0 1
1	0 . . . . . 1 1	1	0 . 1 1 1 1 1 1 1
0	1 1 1 1 1 1 1 1 1	0	1 1 1 1 1 1 1 1 1
$s_{i,j}$	0 1 2 3 4 5 6 7 8	$s_{i,j}$	0 1 2 3 4 5 6 7 8
x_i	. . . . .	x_i	. 1 . . . . .

3	0 0 0 0 0 0 0 0 0
2	0 0 0 0 0 0 0 1 1
1	0 0 1 1 1 1 1 1 1
0	1 1 1 1 1 1 1 1 1
$s_{i,j}$	0 1 2 3 4 5 6 7 8
x_i	0 1 0 0 0 0 1 0

By variants of the above set of clauses we introduce two encodings C_C and C_A, that suffice to translate ATMOSTSEQCARD:

- $\mathrm{C}_C(d, [x_1, x_2, \dots x_n])$ encodes $\sum_{i\in[1..n]} x_i = d$ using clauses 1 to 5 with $u = l = d$.
- $\mathrm{C}_A(u, q, [x_1, x_2, \dots x_n])$ encodes $\bigwedge_{i=0}^{n-q}(\sum_{l=1}^{q} x_{i+l} \leq u)$ by a set of separate translations on each $\sum_{l=1}^{q} x_{i+l} \leq u$ with $i = 1 \dots n-q$ using clauses 1 to 5 with $l = 0$ and u the upper bound.

Since each of the above encodings is a superset of the encoding introduced in [24], C_S and C_A have the following property regarding propagation:

Proposition 1. *Unit Propagation enforces* GAC *on*

1. $\sum_{i\in[1..n]} x_i = d$ *by the encoding* $\mathrm{C}_C(d, [x_1, x_2, \dots x_n])$.
2. ATMOSTSEQ$(u, q, [x_1, \dots, x_n])$ *by the encoding* $\mathrm{C}_A(u, q, [x_1, x_2, \dots x_n])$.

With these encodings at hand we can completely translate ATMOSTSEQCARD to CNF and fulfil the first two properties of a GAC propagator as characterised in the end of Section 4. However, we are missing the global propagation of Equation 4.1.

The sequential counter encoding in [24] uses only clauses 1 and 4. Indeed, they are sufficient to enforce GAC by unit propagation in case of ATMOST. However, their encoding does not necessarily force all auxiliary variables when all x_i are assigned and this effectively increases the number of models which can lead to unnecessary search. Thus, we prefer the more constrained version of the counter encoding.

The encoding in [2] of the more general AMONG constraint has similarities to a counter encoding. This encoding builds on the translation of the REGULAR constraint and introduces variables for states and transitions of a deterministic finite automaton. Regarding propagation, this encoding is equivalent to the sequential counter, but on the clausal level, it is not identical. Our encoding consists only of binary and ternary clauses whereas their encoding introduces longer clauses using two types of auxiliary variables. Another difference is that the state variables represent exact partial sums whereas the encoding presented here relate to the idea of an order encoding.

5.2 Extension to ATMOSTSEQCARD

We still need to capture the missing propagation of a GAC propagator of ATMOSTSEQCARD. To do so we introduce the following binary clauses. They are referred to by $\mathrm{C}_S(u, q, [x_1 \dots x_n])$ and reuse the auxiliary variables $s_{i,j}$ introduced by C_C. $\forall i \in [q..n], \forall j \in [u..d+1]$:

6. $\qquad \neg s_{i,j} \vee s_{i-q,j-u}$

We will show that the binary clauses capture the missing propagation for ATMOSTSEQCARD as in Equation 4.1. For this, we precisely show how the auxiliary variables $s_{i,j}$ relate to L_S and R_S.

Proposition 2. *Let* C_C *and* C_S *be the decomposition of* ATMOSTSEQCARD$(u, q, d, [x_1 \dots x_n])$. *Given a partial assignment* S *on* $\{x_1, x_2 \dots x_n\}$ *and assuming that* $L_S(n) \leq d$ *and* $R_S(0) \leq d$, *for all* $i \in \{0 \dots n\}$ *UP forces*

1. $s_{i,L_S(i)+1}$ *to false and*
2. $s_{i-1,d-R_S(i)}$ *to true.*

Proof. We concentrate on 1) since 2) is analogous. The proof follows an inductive argument on i. For $i = 0$ it holds from unit $s_{0,1}$ in the clauses 5 of C_C. For the inductive step we have to show, assuming $s_{i,L_S(i)+1}$ is set to false, that $s_{i+1,L_S(i+1)+1}$ is enforced to false by UP. There are two cases to analyse: a) $L_S(i+1) = L_S(i) + 1$, and

b) $L_S(i+1) = L_S(i)$. The first case follows from clauses 3 in C_C. The second case involves a complicated step, essentially showing that with the induction hypothesis, clauses 6 in C_S and GAC on C_C, UP enforces $s_{i+1,L_S(i)+1}$ to false.

For case b) there are two situations to consider using the definition of L_S: x_{i+1} is assigned to false in S or x_{i+1} is unassigned. The first situation is covered by clauses 2. In the second situation it holds that $\overrightarrow{w}(x_{i+1}) = 0$. This is caused by `leftmost` not evaluating x_{i+1} to true. Hence, there exists a window $k+1...k+q$ that includes position $i+1$ and the maximal number of variables in this windows assigned to true by $\overrightarrow{w}$ is equal to u. Let there be α true assignements in $\overrightarrow{w}$ before $i+1$ and β after. We have $L_S(k) = L_S(i) - \alpha$. However, since $k \leq i$, we know by induction that $\neg s_{k,L_S(k)+1}$, that is, $\neg s_{k,L_S(i)-\alpha+1}$ holds.

Now, clauses 6 of C_S, instantiated to $\neg s_{k+q,L_S(i)-\alpha+u+1} \vee s_{k,L_S(i)-\alpha+1}$ infers by UP $\neg s_{k+q,L_S(i)+\beta+1}$ (recall that $\alpha + \beta = u$).

Finally, observe that when leftmost computed $L_S(i)$, no assignment were made on the interval $i+2...k+q$. Hence we have $\sum_{j=i+2}^{k+q} \min(x_j) = \beta$. Standard cardinality reasoning (clauses 1 and 4) is thus sufficient to show that $\neg s_{k+q,L_S(i)+\beta+1}$ implies $\neg s_{i+1,L_S(i)+1}$. Since we are in the case where $L_S(i+1) = L_S(i)$ we have shown that UP infers $\neg s_{i+1,L_S(i+1)+1}$. This concludes the inductive proof and it demonstrates that UP maintains the values for $L_S(i)$ for all positions i. The case for R_S follows a dual argument. □

The key idea of the binary clauses 6 can also be found behind the decomposition of GEN-SEQUENCE into cumulative sums as in [5]. Furthermore, there is a strong similarity between the combination of C_C with C_S and the encoding of GEN-SEQUENCE in [2] and it is possible to show that also here in fact it detects dis-entailment on ATMOSTSEQCARD similarly to Theorem 3 of [2]. The following case exemplifies what kind of propagation is missing with the combination of C_C and C_S.

Example 5. Consider the encodings C_C and C_S on $u = 1$, $q = 2$, $d = 2$, $n = 5$ and let x_3 be true, then UP does not enforce x_2 nor x_4 to false. Setting them to true will lead to a conflict by UP through clauses 4 and 6 on positions 2, 3 and 4.

3	0 0 0 0 0 0
2	0 0 0 . . 1
1	0 . . 1 1 1
0	1 1 1 1 1 1
$s_{i,j}$	0 1 2 3 4 5
x_i	. . 1 . .

We see that the encoding C_A would propagate in the case of the previous example. If we combine all three encodings we can provide a CNF encoding that maintains the desired property of GAC on ATMOSTSEQCARD:

Theorem 2. *UP on* C_C+C_A+C_S *enforces* GAC *on the* ATMOSTSEQCARD *constraint.*

Proof. The proof follows from Proposition 1 and 2 showing that this encoding fulfils all sufficient properties of a GAC propagator as described in Section 4.

In particular, UP maintains GAC on $\bigwedge_{i=0}^{n-q}(\sum_{l=1}^{q} x_{i+l} \leq u)$ and $\sum_{i\in[1..n]} x_i = d$ by C_C and C_A. We elaborate on the interesting cases of Equation 4.1:

1. Let $L_S(i)+R_S(i) \leq d$. By Proposition 2 , UP forces $\neg s_{i,L_S(i)+1}$ and $s_{i-1,d-R_S(i)}$ on S, and by assumption we know that $d - R_S(i)$ is greater or equal to $L_S(i)$, so UP forces also $s_{i-1,L_S(i)}$ to true. By clauses 4 instantiated to $\neg x_i \vee \neg s_{i-1,L_S(i)} \vee s_{i,L(i)+1}$ UP forces x_i to false. Hence if $L_S(i) + R_S(i) \leq d$ holds then UP forces x_i to false.

2. Let $L_S(i-1) + R_S(i+1) < d$. By Proposition 2, UP forces $\neg s_{i-1,L_S(i-1)+1}$ and $s_{i,d-R_S(i+1)}$. Since $L_S(i-1) < d - R_S(i+1)$ UP enforces by clauses 1 and 3 that $s_{i,L_S(i-1)+1}$ is true. Now clauses 2 trigger by $x_i \vee s_{i-1,L_S(i)+1} \vee \neg s_{i,L_S(i)+1}$ and set x_i to true by UP. □

6 Experimental results

We tested the different approaches on the three data sets available at CSPLib [14]. All experiments ran on Intel Xeon CPUs 2.67GHz under Linux. For each instance, we performed 5 randomized runs with a 20 minutes time cutoff. The first set contains 5 unsatisfiable and 4 satisfiable instances of relatively small size (100 cars). The second set contains 70 instances generated with varying usage rate. All instances in this set are satisfiable and involve 200 cars. The third set, proposed by Gagné, features larger instances divided into three sets of ten each, involving respectively 200, 300 and 400 cars. Seven of these instances were solved using local search algorithms. To the best of our knowledge the remaining 23 instances have never been proved unsatisfiable. To facilitate the analysis, we grouped the instances into three categories:

In the first category (`sat[easy]`), we consider the 70 satisfiable instances of the second set as well as the 4 satisfiable instances of the first set. All these instances are extremely easy for all the methods we introduce in this paper;

In the second category (`sat[hard]`), we consider the 7 known satisfiable instances of the second set. These instances are challenging and were often out of reach of previous systematic approaches;

In the third category (`unsat`*), we consider the remaining 5 unsatisfiable instances of the first set as well as the 23 unknown instances form the third set. Those instances are challenging and indeed open for 23 of them.

We ran the following methods:

SAT Encoding. We use Minisat (version 2.2.0) with default parameter settings on three variants of the SAT encoding. Links between classes and options as well as the constraint for exactly one class of vehicle per position are translated as in the basic model. For each option we encode one AtMostSeqCard. The following three models differ only in how this translation is performed (w.r.t Section 5):

1. *SAT (1)* encodes the basic model by using C_C+C_A for each AtMostSeqCard.
2. *SAT (2)* uses C_C+C_S for each AtMostSeqCard.
3. *SAT (3)* combines all of three encodings C_C+C_A+C_S.

Hybrid CP/SAT. We use Mistral as a hybrid CP/SAT solver (Section 2) using our explanation for the AtMostSeqCard constraint. We tested four branching heuristics:

1. *hybrid (VSIDS)* uses VSIDS;
2. *hybrid (Slot)* uses the following heuristic (denoted $Slot$): we branch on $option$ variables from the middle of the sequence and towards the extremities following the first unassigned $Slot$. The options are firstly evaluated by their dynamic usage rate[25] then lexicographically compared.

3. *hybrid (Slot/VSIDS)* first uses the heuristic $Slot$, then switches after 100 non-improving restarts to VSIDS.
4. *hybrid (VSIDS/Slot)* reverse of above.

Baseline methods. We also use three "control" approaches run in the same setting:

1. *CP*: A pure CP approach, implemented using Mistral without clause learning on the model described in Section 3 using the $Slot$ branching.
2. *PBO-clauses*: A pseudo-Boolean method relying on SAT encoding. We used MiniSat+ [11] on the pseudo-Boolean encoding described in Section 3 except that the AtMostSeqCard constraint is decomposed into Cardinality and AtMost.
3. *PBO-cutting planes*: A pseudo-Boolean method with dedicated propagation and learning based on cutting planes [10]. We used SAT4J [3] on the same model, with the "CuttingPlanes" algorithm.

For each considered data set, we report the total number of successful runs (*#suc*).[2] Then, we report the number of fail nodes (*fails*) and the CPU time (*time*) in seconds both averaged over all successful runs. We emphasize the statistics of the best method (w.r.t. *#suc*, ties broken by CPU time) for each data set using bold face fonts.

Method	sat[easy] (74 × 5)			sat[hard] (7 × 5)			unsat* (28 × 5)		
	#suc	avg fails	time	#suc	avg fails	time	#suc	avg fails	time
SAT (1)	370	2073	1.71	28	337194	282.35	**85**	**249301**	**105.07**
SAT (2)	370	1114	0.87	31	60956	56.49	65	220658	197.03
SAT (3)	370	612	0.91	34	32711	36.52	77	190915	128.09
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
CP	**370**	**43.06**	**0.03**	35	57966	16.25	0	-	-
PBO-clauses	277	538743	236.94	0	-	-	43	175990	106.92
PBO-cutting planes	272	2149	52.62	0	-	-	1	5031	53.38

Table 1: Evaluation of the models

We first observe that most of the approaches we introduce in this paper significantly improve the state of the art, at least for systematic methods. For instance, in the experiments reported in [23] several instances of the set sat[hard] were not solved within a 20 minutes cutoff. Moreover we are not aware of other systematic approaches being able to solve these instances. More importantly, we are able to close 13 out of the 23 large open instances proposed by Gagné. The set of open instances is now reduced to pb_200_02/06/08, pb_300_02/06/09, and pb_400_01/02/07/08.

[2] They all correspond to solutions found for the two first categories, and unsatisfiability proofs for the last.

Second, on satisfiable instances, we observe that pure CP approaches are difficult to outperform. It must be noticed that the results reported for *CP* are significantly better than those previously reported for similar approaches. For instance, the best methods introduced in [27] 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 [23], the same solver on the same model had a similar behavior on the category `sat[easy]`, but was only able to solve 2 instances of the category `sat[hard]` due to not using restarts.

However, the best method on sat instances is the hybrid solver using a CP heuristic. Moreover, we can see that even with a "blind" heuristic, MiniSat on the strongest encodings has extremely good results (all sat instances were solved with a larger cutoff).

This study shows that propagation is very important to find solutions quickly, by keeping the search "on track" and avoiding exploring large unsatisfiable subtrees. There is multiple evidence for these claims: First, the pseudo Boolean models (*PBO-clauses* and *PBO-cutting planes*) perform limited propagation and are consequently very poor even on `sat[easy]`. Second, the best SAT models for `sat[easy]` and `sat[hard]` are those providing the tightest propagation. Last, previous CP approaches that did not enforce GAC on the AtMostSeqCard constraint are all dominated by *CP*.

For proving unsatisfiability, our results clearly show that clause learning is by far the most critical factor. Surprisingly, stronger propagation is not always beneficial when building a proof using clause learning, as shown by the results of the different encodings. One could even argue for a negative correlation, since the "lightest" encodings are able to build more proofs than stronger ones. Similarly, the pure pseudo Boolean model performs much better comparatively to the satisfiable case. The hybrid models are slightly worse than pseudo Boolean but far better than the 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 [20], or a conjunction of this constraint and AtMostSeqCard [23, 27] were able to build proofs for some of the 5 unsatisfiable instances of the CSPLib. However, these models were not able to solve any of the 23 larger unsatisfiable instances.

7 Conclusion

We proposed and compared hybrid CP/SAT models for the car sequencing problem against several SAT-encodings. Both approaches exploit the AtMostSeqCard constraint. In particular, we proposed a linear time procedure for explaining failure and pruning as well as advanced SAT-encodings for this constraint. Experimental results emphasize the importance of advanced propagation for searching feasible solutions and of clause learning for building unsatisfiability proofs. Our models advance the state of the art in this domain, and close 13 out of the 23 large open instances in CSPLib.

References

1. Fadi A. Aloul, Arathi Ramani, Igor L. Markov, and Karem A. Sakallah. Generic ILP versus specialized 0-1 ILP: an update. In *Proceedings of ICCAD*, pages 450–457, 2002.
2. Fahiem Bacchus. GAC Via Unit Propagation. In *Proceedings of CP*, pages 133–147, 2007.

3. Daniel Le Berre and Anne Parrain. The Sat4j library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*, 7:59–64, 2010.
4. Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.
5. Sebastian Brand, Nina Narodytska, Claude-Guy Quimper, Peter J. Stuckey, and Toby Walsh. Encodings of the Sequence Constraint. In *Proceedings of CP*, pages 210–224, 2007.
6. Hadrien Cambazard. *Résolution de problmes combinatoires par des approches fondées sur la notion dexplication*. PhD thesis, Ecole des mines de Nantes, 2006.
7. Hadrien Cambazard and Narendra Jussien. Identifying and exploiting problem structures using explanation-based constraint programming. *Constraints*, 11(4):295–313, 2006.
8. Martin Davis and Hilary Putnam. A Computing Procedure for Quantification Theory. *Journal of the ACM*, 7(3):201–215, 1960.
9. Heidi Dixon. *Automating Pseudo-Boolean Inference within a DPLL Framework*. PhD thesis, University of Oregon, 2004.
10. Heidi E. Dixon and Matthew L. Ginsberg. Inference Methods for a Pseudo-Boolean Satisability Solver. In *Proceedings of AAAI*, pages 635–640, 2002.
11. Niklas Eén and Niklas Sörensson. Translating Pseudo-Boolean Constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006.
12. Thibaut Feydy, Andreas Schutt, and Peter Stuckey. Semantic Learning for Lazy Clause Generation. In *TRICS workshop, held alongside CP*, 2013.
13. Ian P. Gent. Arc Consistency in SAT. In *Proceedings of ECAI*, pages 121–125, 2002.
14. Ian P. Gent and Toby Walsh. CSPlib: a benchmark library for constraints, 1999.
15. George Katsirelos. *Nogood Processing in CSPs*. PhD thesis, University of Toronto, 2008.
16. George Katsirelos and Fahiem Bacchus. Generalized NoGoods in CSPs. In *Proceedings of AAAI*, pages 390–396, 2005.
17. Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *DAC*, pages 530–535, 2001.
18. Claude-Guy Quimper, Alexander Golynski, Alejandro López-Ortiz, and Peter van Beek. An Efficient Bounds Consistency Algorithm for the Global Cardinality Constraint. *Constraints*, 10(2):115–135, 2005.
19. Jean Charles Régin. Generalized Arc Consistency for Global Cardinality Constraint. In *Proceedings of AAAI*, volume 2, pages 209–215, 1996.
20. Jean-Charles Régin and Jean-François Puget. A Filtering Algorithm for Global Sequencing Constraints. In *Proceedings of CP*, pages 32–46, 1997.
21. Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming*. Elsevier, 2006.
22. Thomas Schiex and Gérard Verfaillie. Nogood Recording for Static and Dynamic CSP. In *Proceeding of ICTAI*, pages 48–55, 1993.
23. Mohamed Siala, Emmanuel Hebrard, and Marie-José Huguet. An optimal arc consistency algorithm for a particular case of sequence constraint. *Constraints*, 19(1):30–56, 2014.
24. Carsten Sinz. Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In *Proceedings of CP*, pages 827–831, 2005.
25. Barbara M. Smith. Succeed-first or Fail-first: A Case Study in Variable and Value Ordering, 1996.
26. Christine Solnon, Van Dat Cung, Alain Nguyen, and Christian 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.

27. Willem J. van Hoeve, Gilles Pesant, Louis-Martin Rousseau, and Ashish Sabharwal. New Filtering Algorithms for Combinations of Among Constraints. *Constraints*, 14(2):273–292, 2009.
28. Toby Walsh. SAT v CSP. In *Proceedings of CP*, pages 441–456, 2000.