# SAT Encodings for the Car Sequencing Problem

Valentin Mayer-Eichberger and Toby Walsh

NICTA and University of New South Wales
Locked Bag 6016, Sydney NSW 1466, Australia
{valentin.mayer-eichberger,toby.walsh}@nicta.com.au

**Abstract.** Car sequencing is a well known NP-complete problem. In this paper we describe encodings of this problem into CNF and demonstrate that SAT solvers are powerful in this domain. We also compare these encodings to automatic translations of pseudo-Boolean solvers and demonstrate the advantages of our approach.

## 1 Introduction

Car sequencing is the first benchmark in the constraint programming library (prob001 in CSPLib [8]). The associated decision problem (is there a production sequence for cars on the assembly line satisfying the sliding capacity constraints?) has been shown to be NP-complete [7]. To date, however, it has not received much attention from the SAT community. This is disappointing as we show here that SAT is a very effective technology to solve such problems. We describe several CNF encodings for this problem and demonstrate the strength of SAT solvers.

The paper is organised as follows. First we formally introduce the car sequencing problem. Then in Section 2 we define the CNF encodings and discuss their properties. In Section 3 we evaluate the CNF encodings on the CSPLib benchmark.

### 1.1 Car Sequencing

Car sequencing deals with the problem of scheduling cars along an assembly line with capacity constraints for different stations (e.g. radio, sun roof, air-conditioning, etc). Cars are partitioned into classes according to their requirements. Let $C$ and $O$ be disjoint sets of classes and options. To each class $k \in C$ there is given a demand of cars $d_k$ to be scheduled. Each option $l \in O$ is limited to $u_l$ occurrences on each subsequence of length $q_l$ (denoted as a capacity constraint $u_l/q_l$), i.e. no more than $u_l$ cars with option $l$ can be sequenced among $q_l$ consecutive cars. To each class $k \in C$ we denote the set $O_k$ of options it requires and for convenience to each option $l \in O$ we denote $C_l$ the set of classes that require this options. A solution is a valid sequence of all cars.

Let $n$ be the length of the total sequence. The following pseudo Boolean model is a basis for our translation to CNF. We use 0/1 variables $c_i^k$ to denote that a car $k \in C$ is at position $i$, likewise $o_i^l$ for option $l \in O$. For the sequence it must hold:

- Demand constraints: $\forall k \in C \quad \sum_{i=1}^{n} c_i^k = d_k$
- Capacity constraints: $\forall l \in O$ with ratio $u_l/q_l \qquad \bigwedge_{i=0}^{n-q_l} (\sum_{j=1}^{q_l} o_{i+j}^l \leq u_l)$
- Link between classes and options: For all $i \in \{1 \ldots n\}$ and $k \in C$
    - $\forall l \in O_k : \;\; c_i^k - o_i^l \leq 0$
    - $\forall l \in O$ with $l \notin O_k : \;\; c_i^k + o_i^l \leq 1$
- In each position $i \in \{1 \ldots n\}$ there is exactly one car: $\sum_{k \in C} c_i^k = 1$

*Example 1.* Given classes $C = \{1, 2, 3\}$ and options $O = \{a, b\}$. The demands (number of cars) for the classes are $3, 2, 2$ and capacity constraints on options are $1/2$ and $1/5$, respectively. Class 1 has no restrictions, class 2 requires option $a$ and class 3 needs options $\{a, b\}$. Given these constraints the only legal sequence for this problem is $[3, 1, 2, 1, 2, 1, 3]$, since class 2 and 3 cannot be sequenced after another and class 3 need to be at least 5 positions apart.

Car sequencing in the CSPLib contains a selection of benchmark problems of this form ranging from 100 to 400 cars. Over the years different approaches have been used to solve these instances, among them constraint programming, local search and integer programming [9][10].

## 2 Modelling the Car Sequencing Problem

In this section we introduce different ways to model the car sequencing problem in CNF. The basic building blocks are cardinality constraints of the form $\sum_{i \in \{1 \ldots n\}} x_i = d$ and $\sum_{i \in \{1 \ldots n\}} x_i \leq d$.

First we describe how to translate cardinality constraints as a variant of the sequential counter encoding proposed by [11]. Then we show how to enforce a global view by integrating capacity constraint into the sequential counter. We then show that this combination of the demand and the capacity constraints can be used both for classes and options and we define three complete encodings for the car sequencing problem.

### 2.1 Sequential Counter Encoding

We describe how to encode a cardinality constraint of the form $\sum_{i \in \{1 \ldots n\}} x_i = d$ where $x_i$ are $0/1$ variables and $d \in \mathbb{N}$ is a fixed demand. The key idea is to introduce auxiliary variables to represent cumulative sums.

In this section we use two types of variables. The variable $x_i$ is true iff an object (class or option) is at position $i \in \{1, \ldots n\}$. The auxiliary $s_{i,j}$ is true iff in the positions $0, 1 \ldots i$ the object exists at least $j$ times (for technical reasons $0 \leq j \leq d + 1$).

The following set of clauses (1) to (5) define the sequential counter encoding: $\forall i \in \{1 \ldots n\} \; \forall j \in \{0 \ldots d + 1\}$:

$$\neg s_{i-1,j} \vee s_{i,j} \tag{1}$$

$$x_i \vee \neg s_{i,j} \vee s_{i-1,j} \tag{2}$$

$\forall i \in \{1 \dots n\} \forall j \in \{1 \dots d+1\}$:

$$\neg s_{i,j} \vee s_{i-1,j-1} \tag{3}$$

$$\neg x_i \vee \neg s_{i-1,j-1} \vee s_{i,j} \tag{4}$$

$$s_{0,0} \wedge \neg s_{0,1} \wedge s_{n,d} \wedge \neg s_{n,d+1} \tag{5}$$

The variables $s_{i,j}$ represent the bounds for cumulative sums for the sequence $x_1 \dots x_i$. The binary clauses (1) and (3) ensures 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 total sum at position $n$ is equal to $d$.

The encoding in [11] follows a similar idea and focuses on inequalities. The encoding here can easily be adapted to represent such constraints by removing $s_{n,d}$ from the conjunction (5). Then the counter accepts all assignments to $x_1 \dots x_n$ with up to $d$ variables set to true. Comparing the resulting clauses there are small differences between the encoding proposed here and the one in [11]. In particular, we use twice as many clauses and but ensure uniqueness by means of the auxiliary variables, i.e. we ensure the same model count. The encoding here also has similarities to the translation of cardinality constraints through binary decision diagrams, see [5].

### 2.2 The Capacity Constraint

We now show how to translate the interleaving capacity constraints to CNF. Each subsequence of length $q$ can have at most $u$ true assignments. Thus, the capacity constraints are a sequence of cardinality expressions. We will translate this expression to CNF in two ways. The straight forward way is to encode a sequential counter for each subsequence separately. This will introduce independent auxiliary variables for each subsequence. The second way is more elaborate and is explained in this section.

The idea is to encode a more global view into the demand constraint by integrating the capacity of each subsequence into the counter. We intend to encode the following expression: $(\sum_{i=1}^{n} x_i = d) \wedge \bigwedge_{i=0}^{n-q} (\sum_{l=1}^{q} x_{i+l} \leq u)$.

Interestingly, we can reuse the auxiliary variables of the sequential counter and impose the following set of clauses: $\forall i \in \{q \dots n\}$, $\forall j \in \{u \dots d+1\}$:

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

The clauses restrict the internal counting not to exceed the capacities constraints and the encoding detects dis-entailment on the conjunction of the demand constraint and the capacity constraints by unit propagation. In particular, these binary clauses improve propagation when branched on auxiliary variables.

The encoding presented here is in fact similar to a special case of the encoding of the GEN-SEQUENCE constraint in [1]. One difference lies in their auxiliary variables $q_i^j$ that encode the equality $\sum_{l=1}^{i} x_l = j$. This also changes the size and shape of the clauses and will have different behaviour when branching on auxiliary variables. Our encoding also has similarities to the decomposition of the SEQUENCE constraint into cumulative sums as definied in [2].

### 2.3 Link Cars and Options

For a complete CNF translation of car sequencing we need to link classes and options. This is done by the following clauses.

$\forall i \in \{1 \ldots n\}$:

$$\bigwedge_{\substack{k \in C \\ l \in O_k}} (\neg c_i^k \vee o_i^l) \quad \wedge \bigwedge_{\substack{k \in C \\ l \notin O_k}} (\neg c_i^k \vee \neg o_i^l) \quad \wedge \bigwedge_{l \in O} \left( \left( \neg o_i^l \vee \bigvee_{k \in C_l} c_i^k \right) \right) \tag{7}$$

The first two set of Clauses follow directly from the pseudo Boolean model. We also add the third to ensure more propagation when e.g. branched negatively on a set of variables of classes such that a support for an option is lost and its variable should be false. Furthermore, for each position an additional sequential counter is used to restrict the number of cars to exactly one.

### 2.4 The Complete Model

The demand constraints for classes are translated through cardinality constraints. In fact, we can identify for each option $l \in O$ the implicit demand by adding up the demand of all classes $C_l$ that require this option. $d_l = \sum_{i=1}^{n} o_i^l = \sum_{k \in C_l} d_k$.

Moreover, for each class we may use one capacity constraint of its options as this restriction applies also to the class. To maximise pruning we choose the strictest capacity constraint with minimal (u/q), e.g. 1/5 restricts more than 2/3. With this setting the translation of classes and options can be uniformly done by the same type of constraints - a demand constraint and consecutive overlapping capacity constraints. In the following we refer to capacity constraints both for options and classes.

We define three CNF encodings E1, E2 and E3. All three encode the demand constraint by a sequential counter with clauses (1) to (5). The link between classes and options for all three models is encoded by clauses (7). In the experimental evaluation we want to identify the impact clauses (6) have in practice and we alternate the translation of the capacity constraint in the following ways:

- E1 translates each capacity constraint separately by the clauses (1) to (5) with a fresh set of auxiliary variables.
- E2 translates the capacity constraints by clauses (6) and thus reuses the variables of the sequential counter on the demand constraint.
- E3 uses both E1 and E2.

## 3 Evaluation

We will compare the SAT encodings of Section 2 with pseudo Boolean solvers on the CSPLib instances. Our focus is on the 9 traditional instances plus the 30 hard instances proposed by [9] and we leave out the set of 70 easy satisfiable instances.

All encodings and results are available at `github.com/vale1410/car-sequencing`. For our experiments we choose the well-known SAT solver Minisat [4] of version 2.2.0. All experiments are done on Linux 64bit, Intel Xeon CPUs at 2.27GHs.

We compare to the pseudo Boolean solver (PB) Minisat+ version 1.0-2 [5] and to the answer set programming solver Clasp 2.1.3 [6], referred to as PB and ASP respectively. Both approaches offer native cardinality constraints. We use for PB and ASP the basic model of Section 2 and add the redundant constraint for the implicit demand on options. Apart from the encoding of cardinality constraints this should be equivalent to model E1.

Clasp solves hardly any instance with the standard configuration. Instructing Clasp to ground all cardinality constraints to normal rules improves the number of solved instances (using the switch `--trans-ext=all`), so we will report here with this switch turned on. Apart from that we use the standard configuration for all solvers. Clasp and Minisat+ apply different translations for cardinality constraints. Clasp applies an encoding based on counters and Minisat+ uses encodings through adders, sorting networks and binary decision diagrams [5].

We show the results for the selected benchmark on models E1, E2, E3, ASP and PB with a timeout of 1800sec. In total 11/39 instances cannot be solved by any approach.

| | E1 | E2 | E3 | ASP | PB |
|---|---|---|---|---|---|
| #solved UNSAT | **17** | 15 | **17** | 10 | 8 |
| #fastest UNSAT | **5** | 4 | 4 | 0 | 4 |
| #solved SAT | **11** | **11** | **11** | 7 | 2 |
| #fastest SAT | 0 | 4 | **7** | 0 | 0 |

We see that the models E1 to E3 perform considerably better than the one generated through Minisat+ or Clasp (Running Clasp as a SAT solver on our models is comparable to Minisat). This indicates that the internal treatment of cardinality expressions of both tools is inferior compared with Minisat on our encodings.

There is a tendency of model E3 to solve satisfiable instances faster than the other two models, whereas model E1 is stronger in finding UNSAT proofs. Since E3 is a superset of E1, the additional clauses cost some performance in finding unsatisfiability proofs. On the other hand the increased propagation due to the global view is a factor for faster solving times for satisfiable instances, since E2 and E3 dominate here clearly. Interestingly Minisat+ which in general performs poor, does in some cases find UNSAT proofs faster than all other methods. Overall, the results show clearly that it is crucial how cardinality constraints are translated and automatic translations should be handled with care.

## 4 Conclusion and Future Work

We have introduced CNF encodings for the car sequencing problem based on sequential counters and demonstrated that SAT solvers perform well on the instances of the CSPLib. This specialised translation has advantages over automatic translations of cardinality constraints as done in Minisat+ and Clasp. Our approach is still work in progress and in the following we identify our next steps and future work.

To identify the precise advantages of our encodings we will extend the comparison to other CNF encodings for cardinality constraints, like parallel counters and various types of sorting networks [3]. We will also contrast our work in more depth with [1]. We have pointed out properties of the presented encodings and we plan to extend this analysis to a theoretical level and prove consistency properties.

Our experimental analysis still lacks comparison to related paradigms as constraint programming, local search and integer programming. We plan to extend the evaluation by comparing these approaches.

## Acknowledgement

We want to thank Mohamed Siala, Emmanuel Hebrard and Nina Narodytska for fruitful discussions on modelling the car sequencing problem.

## References

1. F. Bacchus. GAC Via Unit Propagation. In *CP*, pages 133–147, 2007.
2. S. Brand, N. Narodytska, C. Quimper, P. Stuckey, and T. Walsh. Encodings of the Sequence Constraint. In *CP*, pages 210–224, 2007.
3. M. Codish and M. Zazon-Ivry. Pairwise Cardinality Networks. In *LPAR (Dakar)*, pages 154–172, 2010.
4. N. Eén and N. Sörensson. An Extensible SAT-solver. In *SAT*, pages 502–518, 2003.
5. Niklas Eén and Niklas Sörensson. Translating Pseudo-Boolean Constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):1–26, 2006.
6. M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. *clasp* : A conflict-driven answer set solver. In *LPNMR*, pages 260–265, 2007.
7. I. Gent. Two Results on Car-sequencing Problems. In *Report APES-02-1998*, 1998.
8. I. Gent and T. Walsh. $\text{CSP}_{\text{LIB}}$: A Benchmark Library for Constraints. In *CP*, pages 480–481, 1999.
9. M. Gravel, C. Gagné, and W. L. Price. Review and Comparison of Three Methods for the Solution of the Car Sequencing Problem. *The Journal of the Operational Research Society*, 56(11):1287–1295, 2005.
10. M. Siala, E. Hebrard, and M. Huguet. An Optimal Arc Consistency Algorithm for a Chain of Atmost Constraints with Cardinality. In *CP*, pages 55–69, 2012.
11. C. Sinz. Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In *CP*, pages 827–831, 2005.