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. This paper introduces encodings of this problem into CNF based on sequential counters. We demonstrate that SAT solvers are powerful in this domain and report new lower bounds for the benchmark set in the CSPLib.

1 Introduction

Car sequencing is the first benchmark in the constraint programming library (prob001 in CSPLib [9]). 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 [12][8]. 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 introduce several CNF encodings for this problem that demonstrate the strength of SAT solvers. Furthermore, we identify new lower bounds by a preprocessing technique and by SAT solving.

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. A direct method to compute lower bounds is explained in Section 3. In Section 4 we evaluate the CNF encodings on the CSPLib benchmark and show results on the lower bounds.

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: $\forall k \in C$

$$\sum_{i=1}^{n} c_i^k = d_k$$

- Capacity constraints: $\forall l \in O$ with ratio u_l/q_l

$$\bigwedge_{i=0}^{n-q_l} (\sum_{j=1}^{q_l} o_{i+j}^l \leq u_l)$$

And in all positions $i \in \{1 \dots n\}$ of the sequence it must hold:

- Link between classes and options: for each $k \in C$ and

$$\forall l \in O_k : \; c_i^k - o_i^l \leq 0$$
$$\forall l \in O \text{ with } l \notin O_k : \; c_i^k + o_i^l \leq 1$$

- 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 [14][10][11][6][15].

Car sequencing has also been treated as an optimisation problem, although the literature does not agree on a common optimisation function and several versions have been proposed. There are several variations for the optimisation function minimising the number of violated capacity constraints. However, in this paper we use the definition of [13] which naturally transforms to a sequence of decision problem and SAT solving can be directly applied: An unsatisfiable car sequencing problem can be made solvable by adding empty slots (also called dummy cars) to the sequence. The task is then to minimise the number of dummy cars needed to generate a valid sequence.

We state the optimisation function for which we show lower bounds and for completeness provide also the optimisation function to which we compare in Section 4:

1. Let $z \in C$ be the class of additional cars that require no options with variable demand $d_z \in \mathbb{N}$. The optimisation function is then

$$\texttt{minimise}\ d_z$$

2. Let v_i^l be 1 if the capacity constraint of option l is violated in the subsequence starting at position i, otherwise 0. Then the optimisation function is

$$\texttt{minimise} \sum_{l \in O} \sum_{i=1}^{n-q_l+1} v_i^l$$

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 [16]. 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, for each position i

- x_i is true iff an object (class or option) is at position i
- $s_{i,j}$ 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: $\forall i \in \{1 \ldots n\}\ \forall j \in \{0 \ldots d+1\}$:

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

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

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

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

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

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

The variables $s_{i,j}$ represent the bounds for cumulative sums for the sequence $x_1 \dots x_i$. The encoding is best explained by visualising $s_{i,j}$ as a two dimensional grid with positions (horisontal) and cumulative sums (vertical). 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.

Example 2. The auxiliary variables $s_{i,j}$ with $d = 2$ over a sequence of 10 variables. The cells with $U(L)$ and above(below) are set to false (true) by preprocessing. The question mark identifies yet unassigned variables. Left: before variable assignments, right: after variable assignment x_2 and x_7 to true.

3			U	U	U	U	U	U	U	U	U
2		U	?	?	?	?	?	?	?	?	L
1	U	?	?	?	?	?	?	?	?	L	
0	L	L	L	L	L	L	L	L	L		
$s_{i,j}$	0	1	2	3	4	5	6	7	8	9	10

3			U	U	U	U	U	U	U	U	U
2		U	0	0	0	0	0	1	1	1	L
1	U	0	1	1	1	1	1	1	1	L	
0	L	L	L	L	L	L	L	L	L		
$s_{i,j}$	0	1	2	3	4	5	6	7	8	9	10
x_i	0	0	1	0	0	0	0	1	0	0	0

The encoding in [16] 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 [16]. 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. Our encoding 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.

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

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 global view on 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 following example will demonstrate the way the binary clauses of (6) work. It also demonstrates how much propagation is missing if the capacity constraints would be translated separately.

Example 3. See tables in Figure 1 for a visualisation of the variables. We construct the grid for capacity constraints with ratio 4/8 and a demand of $d = 12$ on a sequence of 22 variables. Unit propagation will force x_7, x_8, x_{15} and x_{16} to be false prior to search. The second table examines the variables after decisions x_1 and x_{13} to true and x_{12}, x_{14} and x_{21} to false.

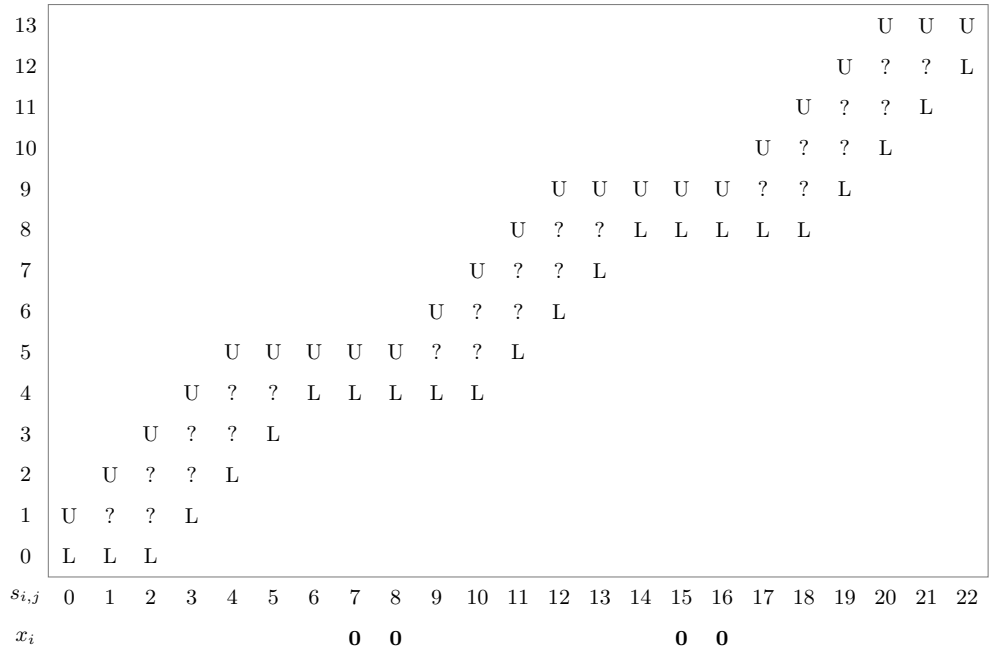
It is easy to see that the number of auxiliary variables is not more than $n \cdot d$. Since clauses (1) to (5) and (6) are generated for each auxiliary variable, the encoding consists of $3 \cdot n \cdot d$ binary clauses and $2 \cdot n \cdot d$ ternary clauses. Note that many clauses become unit in preprocessing, this number increases the stricter the capacity constraints are. E.g. in Example 3 with $n = 20$, $d = 12$ and 4/8 there are effectively only 24 unassigned variables after the first unit propagation.

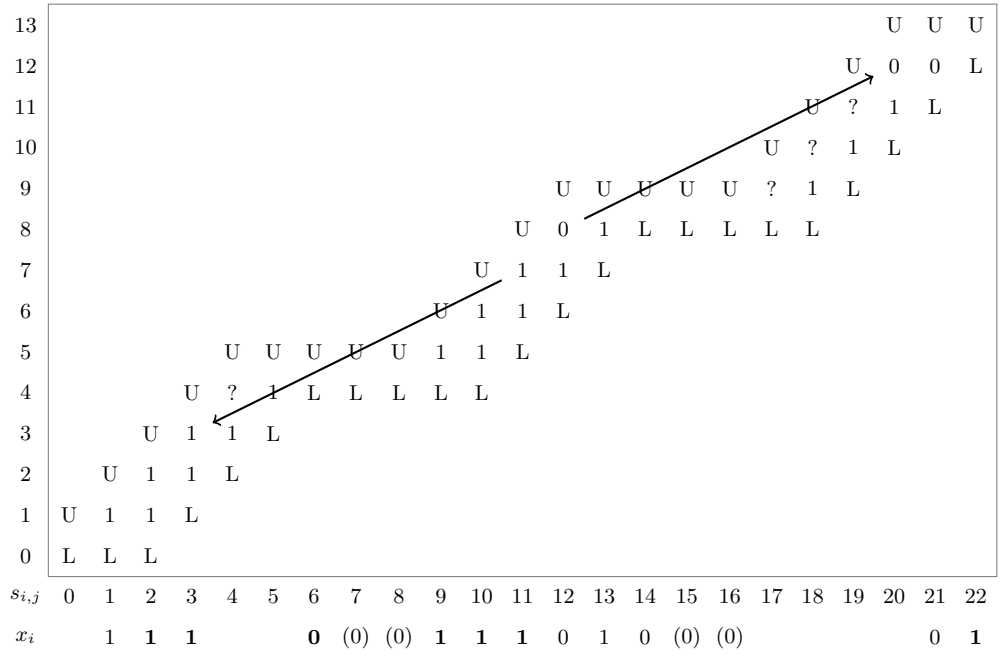
In the following example we show a situation where clauses (6) do not prune all values with unit propagation, whereas translating each capacity constraint to a counter would do so.

Example 4. Let $n = 5$, $d = 2$ with a capacity constraint of 1/2, and let x_3 be true, then unit propagation does not force x_2 nor x_4 to false. Setting them to true will lead to a conflict through clauses (4) and (6) on positions 2, 3 and 4.

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

Fig. 1. The grid of variables of Example 3. Notation for bottom row: normal font = decision variable, bold = propagated, in brackets = propagated previously. Notice the amount of propagation due to the clauses (6), the interesting two cases are shown by an arrow. These values would not have been found by unit propagation without the clauses (6). (Configuration of this example taken from [15]).





The encoding presented here is in fact similar to a special case of the encoding of the GEN-SEQUENCE constraint in [2]. 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.

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.

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

$$\bigwedge_{\substack{k \in C \\ l \in O_k}} \neg c_i^k \vee o_i^l \tag{7}$$

$$\bigwedge_{\substack{k \in C \\ l \notin O_k}} \neg c_i^k \vee \neg o_i^l \tag{8}$$

$$\bigwedge_{l \in O} \left(\neg o_i^l \vee \bigvee_{k \in C_l} c_i^k \right) \tag{9}$$

Clause (7) and (8) follow directly from the pseudo Boolean model, whereas we add (9) 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),(8) and (9). As we have seen in Example 3 and 4, unit propagation prunes different values on the translation of the capacity constraints. This motivates us to demonstrate the difference in the experimental section:

- 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 Lower Bounds by Preprocessing

The idea to this method goes back to the proof in [8] to show a lower bound of 2 for the instance 19/97. Here we will restate this technique to apply the method on all problems from the benchmark in [11].

We start with instance 300-04 as an example. The demands and options are given in Table 1.

Table 1. Overview of options and demands for instance 300-04

class	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
demand	9	4	22	2	1	62	31	4	24	4	3	36	3	25	3	8	5	2	6	21	5	7	11	2
0: 1/2	-	-	-	-	-	-	-	-	-	-	-	x	x	x	x	x	x	x	x	x	x	x	x	x
1: 2/3	-	-	-	-	-	x	x	x	x	x	x	-	-	-	-	-	-	-	x	x	x	x	x	x
2: 1/3	-	-	x	x	x	-	-	-	x	x	x	-	-	-	-	x	x	x	-	-	-	x	x	x
3: 2/5	-	x	-	-	x	-	x	x	-	-	x	-	-	x	x	-	-	x	-	x	x	-	x	x
4: 1/5	x	x	-	x	x	-	-	x	-	x	x	-	x	-	x	-	x	x	x	-	x	x	-	x

There are two classes, 21 and 23, that require options 0, 1, 2 and 4 and sum of demands is 9. First observation is that all other classes share at least one option with these two classes. Secondly cars of class 21 and 23 have to be put at least 5 apart, so they cannot share a neighbour. Furthermore, they cannot be neighbour to any of the classes that have a $1/q$ restriction. This leaves us with the classes that only share the option 1 and for each car at most one adjacent car can have restriction 2/3. Since the first and the last car in the sequence can have any neighbour with that restriction, the number of cars that share no option is at least $9-2=7$. Since there are no such cars, the lower bound for this problem is 7.

A similar argument can be made for classes 21, 22, 23 that share options 0, 1 and 2. Here the collective demand is 20 and the supply of cars that have neither

of these options is $20 - 13 = 7$. This gives a lower bound of 5, which is weaker than the first case.

We unify the two cases into a method that can then be used to compute lower bounds. Given a set of options $B \subseteq O$ that contain the following capacity constraint: at least one of the form $1/q$ where $q \geq 3$ and at most one with $2/r$ where $r \geq 3$ and arbitrary many $1/s$ for $s \geq 2$. If the total demand for this set is d_B, then there have to be at least $d_B - 2$ cars that do not require any of these options in B. The reason for this is as in the example above. Cars that have all options in B are at least 3 positions apart and thus cannot share an adjacent car. Cars that share at least one option with B can at most occupy one side since the weakest restriction is $2/r$ and consequently for a valid sequence cars that contain no option in B are needed. Edge cases (start and end of the sequence) are removed and thus we need $d_B - 2$ cars with no options in B. A lower bound is then the difference of demand and availability of such cars.

4 Evaluation

First we will compare the SAT encodings of Section 2 with pseudo Boolean solvers on the CSPLib instances. Then we show our results for lower and upper bounds. Our focus is on the 9 traditional instances plus the 30 hard instances proposed by [11] and we leave out the set of 70 easy satisfiable instances. All instances have the same set of options: 1/2,2/3,1/3,2/5 and 1/5. The largest instances is 400 cars with 25 classes and a maximal demand per class of 58.

We have written a command line tool that generates CNF in DIMACS format from a problem description as provided by the CSPLib. The tool translates the specification by different sets of clauses and computes the lower bounds from the preprocessing as defined in Section 3. It is freely available at `github.com/vale1410/car-sequencing`. For our experiments we choose the well-known SAT solver Minisat [4] of version 2.2.0, that represents a canonical implementation of state-of-the-art CDCL solvers. All experiments are done on Linux 64bit, Intel Xeon CPUs at 2.27GHs.

4.1 The Decision Problem

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 [7], 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]. In Table 2 we compare the size of the resulting formula of the different approaches. Note that ASP and PB do not differ from model E1 in terms of higher level constraint. So the difference in size is mainly caused by the encoding of the cardinality expressions. We see that the internal translation of Minisat+ prefers a more compact encoding and Clasps encoding is the largest. The sizes of the direct CNF models do not vary much since the difference is in whether to use clauses (6) and/or separate encodings of the capacity constraints. the size.

Table 2. Comparison of number of variables and clauses after translation to CNF. Values (in thousands) are average over the instances of the same length (100 to 400 cars).

Length	Variables					Clauses				
	E1	E2	E3	ASP	PB	E1	E2	E3	ASP	PB
100	27	21	27	90	14	91	83	99	243	54
200	73	60	73	335	33	256	259	291	946	127
300	136	118	136	747	48	495	524	573	2195	197
400	223	199	223	1308	63	827	907	972	3879	271

We show in Table 3 the results for the selected benchmark on models E1, E2, E3, ASP and PB with a timeout of 1800sec. The instances that can be solved by at least one method are shown and grouped into satisfiable and unsatisfiable. In total 11/39 instances cannot be solved by any approach. 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.

4.2 The Optimisation Problem

Moving to the optimisation version of car sequencing we show in Table 4 the best known lower bounds (LB) and upper bounds (UB) found by the preprocessing

Table 3. Comparison of the SAT, ASP and PB model. The upper table gives the satisfiable instances and the lower gives the unsatisfiable instances, that were solved by at least one of the models. (Average of three runs with different seeds, * indicates that not all runs solved the instance).

Inst(SAT)	E1	E2	E3	ASP	PB
4/72	0.14	0.17	**0.05**	6.78	-
16/81	0.38	**0.08**	0.14	13.25	-
41/66	0.06	**0.04**	0.04	2.55	123.41
26/82	0.95	**0.16**	0.21	80.06	654.80
200-01	30.43	47.70	**15.35**	1141.87	-
200-07	6.32	2.39	**1.46**	1478.01	-
300-01	143.76	10.62	**5.98**	810.23	-
300-07	59.86	28.39	**8.24**	-	-
400-05	623.30*	768.97	846.41	-	-
400-06	445.36	24.79	**16.29**	-	-
400-10	884.91	18.99	**13.50**	-	-

Inst(UNSAT)	E1	E2	E3	ASP	PBO
6/76	72.55	117.28	**57.55**	929.87	289.68
10/93	11.40	**6.48**	11.08	331.31	-
21/90	119.83	**74.01**	95.18	-	-
36/92	**16.97**	18.67	41.34	277.63	-
200-03	137.21	**24.02**	30.81	-	-
200-04	69.64	475.76	16.83	-	**1.84**
200-05	**254.03**	1337.39*	1172.38*	-	-
200-09	358.81	-	504.26	-	**3.77**
200-10	**2.10**	3.36	2.53	3.91	2.32
300-03	**99.03**	-	214.41	949.55	-
300-04	3.17	30.57	**2.03**	46.60	4.40
300-08	18.52	799.16	50.98	123.22	**13.54**
300-05	**0.37**	2.73	0.62	-	-
300-10	1.08	25.15	**0.96**	1282.09	904.31
400-03	37.05	**30.31**	31.47	-	-
400-04	13.03	185.32	**6.33**	130.33	14.47
400-09	25.75	470.60	32.90	557.60	**5.19**

and SAT solving. The column LB(pre) contains the lower bounds determined by the preprocessing as explained in Section 3. The next four columns show the lower bounds and upper bounds and the time to compute the last instance. For the bounds the number of additional empty slots ranges from 0 to the best known upper bound in literature. Each run was limited to 1800 seconds and we picked the best results among model E1 to E3. In the last two columns we compare the bounds that have been published previously to the best of our knowledge.

Note that lower and upper bounds (LB*,UB*) in the literature are subjected to different definitions of the optimisation function and cannot directly be compared. Our result show that in many cases the different definitions force the same upper and lower bound, in others we find that they are incomparable. See 400-03 for conflicting lower and upper bound between the definitions, this was also reported in [6]. Instance 300-5 indicates that our version of the optimisation function allows smaller upper bounds due to a large gap between UB and UB*. For some instances there is still a large gap between lower and upper bound, and room for improvement. The method for computing lower bound from Section 3 is powerful and reports for many of the instances higher lower bounds than the SAT approach. Combining the two methods 21/30 of the new instances are solved to optimality.

5 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. Furthermore, for one type of the optimisation problem of car sequencing we have shown new lower and upper bounds and provide the SAT community with a set of hard benchmarks. 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][1]. We will also contrast our work in more depth with [2]. We have pointed out some properties of the presented encodings and we plan to lift 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.

We believe there is a generalisation to the method in Section 3 to arbitrary sets of options. Our results show evidence that such a structure can be beneficial in determining lower bounds. The analysis of subsets of options and their collective demand might not only help in preprocessing but also lead to interesting redundant constraints that can heavily improve early pruning in the search tree. The generation of such constraints is based on an exponential method in the

Table 4. Lower and upper bounds found by preprocessing (pre), by the SAT solving and the best known bounds from the CSPLib.

	LB (pre)	LB (SAT)	sec	UB (SAT)	sec	LB* (known)	UB* (known)
4/72		0	0.07	0	0.07	0	0
6/76		6	209.77	6	0.10	1	6
10/93		1	18.93	3	0.53	1	3
16/81		0	-	0	0.07	0	0
19/71	2	-	-	2	1.50	2	2
21/90	2	1	93.80	2	0.11	1	2
36/92		1	38.55	1	0.07	1	2
41/66		0	-	0	0.06	0	0
26/82		0	-	0	0.14	0	0
200-01		0	-	0	7.46		0
200-02	2	-	-	2	0.86		2
200-03		3	1323.05	3	110.33		3
200-04	7	1	17.59	7	1.04		7
200-05		1	639.42	3	39.63		6
200-06	6	-	-	6	0.69		6
200-07		0	-	0	0.69		0
200-08	8	-	-	8	20.17		8
200-09	10	1	189.14	10	2.32		10
200-10	17	16	213.88	17	3.51		19
300-01		0	-	0	24.83		0
300-02		-	-	6	39.89		12
300-03	13	2	872.06	13	77.76		13
300-04	7	6	795.68	7	12.20		7
300-05	2	12	1145.39	16	1247.82		27
300-06	2	-	-	2	1559.76		2
300-07		0	-	0	6.33		0
300-08	8	1	102.26	8	1.01		8
300-09	7	-	-	7	141.56		7
300-10	3	9	863.15	13	115.67		21
400-01		-	-	-	-		1
400-02	15	-	-	15	112.36		15
400-03		10	1531.53				9
400-04	19	5	25.85	19	222.68		19
400-05		0	-	0	302.19		0
400-06		0	-	0	2.76		0
400-07		-	-	-	-		4
400-08	4	-	-	-	-		4
400-09		4	1253.59	5	53.49		5
400-10		0	-	0	27.05		0

number of options, but typically this numbers is small compared to the length of the sequence.

Acknowledgement

We want to thank the reviewers for their comments that helped us improving the paper. We also want to thank Mohamed Siala, Emmanuel Hebrard and Nina Narodytska for fruitful discussions on the car sequencing problem.

References

1. Roberto Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. Cardinality Networks: a theoretical and empirical study. *Constraints*, 16(2):195–221, 2011.
2. Fahiem Bacchus. GAC Via Unit Propagation. In *CP*, pages 133–147, 2007.
3. Michael Codish and Moshe Zazon-Ivry. Pairwise Cardinality Networks. In *LPAR (Dakar)*, pages 154–172, 2010.
4. Niklas Eén and Niklas 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. Bertrand Estellon, Frédéric Gardi, and Karim Nouioua. Large neighborhood improvements for solving car sequencing problems. *RAIRO - Operations Research*, 40(4):355–379, 2006.
7. Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub. *clasp* : A conflict-driven answer set solver. In *LPNMR*, pages 260–265, 2007.
8. Ian P. Gent. Two Results on Car-sequencing Problems. In *Report APES-02-1998*, 1998.
9. Ian P. Gent and Toby Walsh. CSP_{LIB}: A Benchmark Library for Constraints. In *CP*, pages 480–481, 1999.
10. Jens Gottlieb, Markus Puchta, and Christine Solnon. A Study of Greedy, Local Search, and Ant Colony Optimization Approaches for Car Sequencing Problems. In *EvoWorkshops*, pages 246–257, 2003.
11. 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.
12. Tamás Kis. On the complexity of the car sequencing problem. *Operations Research Letters*, 32(4):331 – 335, 2004.
13. Laurent Perron and Paul Shaw. Combining Forces to Solve the Car Sequencing Problem. In *CPAIOR*, pages 225–239, 2004.
14. Jean-Charles Régin and Jean-Francois Puget. A Filtering Algorithm for Global Sequencing Constraints. In *CP*, pages 32–46, 1997.
15. Mohamed Siala, Emmanuel Hebrard, and Marie-José Huguet. An Optimal Arc Consistency Algorithm for a Chain of Atmost Constraints with Cardinality. In *CP*, pages 55–69, 2012.
16. Carsten Sinz. Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In *CP*, pages 827–831, 2005.