SAT Encodings for the Car Sequencing Problem

Valentin Mayer-Eichberger

NICTA
University of New South Wales
valentin.mayer-eichberger@nicta.com.au

1 Introduction

We will present several encodings of the car sequencing problem and discuss their differences. First we start with a rather simple and direct translation of the description of the problem and then gradually improve this encoding by various techniques. In the construction we will show how to use partial sums, order encodings and how to compactly translate cardinality constraints and strengthen propagation by focusing on the auxiliary variables.

2 Motivation

Our motivation comes from the recent good results of [Siala et al., 2012] on the car sequencing benchmark by using a specialized propagator of the AtMostSeqCard constraint. This constraint treats a special case of the Sequence constraint ([van Hoeve et al., 2006]) where the lower bound for the Amongs is 0. Here we will show that there is good CNF decomposition that archive compatible results on the instances of benchmarks of in the CSP_{LIB} [Gent and Walsh, 1999].

3 Two simple Encoding

1. Encoding by boolean variables c_i^k for class k in position i and a cardinality constraint on the total number of classes on the sequence and cardinality constraints on each sub window restricting the capacity on options. No further global constraints and cardinality constraints translated by standard SAT encoding. (referred to as E_1)
2. As 1) but introducing auxiliary variables o_i^l for option l in position i to encode the capacity constraints on options. (referred to as E_2).

4 Encoding of a single AtMostSeqCard Constraint

We will first show how to encode a cardinality constraint with a counter encoding (First publication?) and then integrate the AtMostSeq by reusing the auxiliary variables of the counter encoding.

Over this whole section we will work with the following notation. Given a set of consecutive positions $P = \{1 \dots n\}$ and a property that holds at a position $i \in P$ iff the boolean variable x_i is true.

4.1 Encoding of Counters

We show how to encode a cardinality constraint of the following form

$$\sum_{i \in \{1 \ldots n\}} x_i = d$$

where x_i are boolean variables and d is a fixed value. The encoding given here is named a counter encoding. The idea is to introduce auxiliary variables to denote cumulative sums with order encoding ([Tamura et al., 2009]) on these variables.

We introduce the following boolean variables (representing cumulative sums):

- $y_{i,j}$ is true iff in the positions $1 \ldots i$ the property holds at least j times.

The following formula clarifies the relationship between x and y.

$$y_{i,j} \iff (j \leq \sum_{l=0}^{i} x_l)$$

Fig. 1. The variables $y_{i,j}$ with $d = 2$ over a sequence of 10 variables. By preprocessing the variables corresponding to the cells containing $U(L)$ and above(below) are set to false (true). The question mark identifies yet unassigned variables.

The following binary clauses relate the variables y among each other:

$$\bigwedge_{i \in \{0 \ldots n-1\}} \bigwedge_{j \in \{0..d+1\}} \neg y_{i,j} \vee y_{i+1,j} \tag{1}$$

$$\bigwedge_{i \in \{1..n\}} \bigwedge_{j \in \{1..d+1\}} \neg y_{i,j} \vee y_{i-1,j-1} \tag{2}$$

These clauses restrict assignments of the auxiliary variables to represent a counter.

Now we need to relate these variables to x. First we restrict the counter not to increase if x_i is false:

$$\bigwedge_{i \in \{1 \ldots n\}} \bigwedge_{j \in \{0..d+1\}} x_i \vee \neg y_{i,j} \vee y_{i-1,j} \tag{3}$$

Second we define clauses that push the counter up if x_i is true.

$$\bigwedge_{i \in \{0 \ldots n-1\}} \bigwedge_{j \in \{0..d\}} \neg x_{i+1} \vee \neg y_{i,j} \vee y_{i+1,j+1} \tag{4}$$

Now we finally have to "initialize" the counter.

$$y_{n,d} \wedge \left(\bigwedge_{i \in \{0 \ldots n\}} y_{i,0} \right) \wedge \neg y_{0,1} \wedge \left(\bigwedge_{i \in \{0 \ldots n\}} \neg y_{i,d+1} \right) \tag{5}$$

It should be mentioned that formula (5) defines the counter to accept exactly d occurrences. However not needed for later, we can easily change the initialiser to define a constraint of at most d or at least d.

Proposition 1. *Clauses (1) to (4) encode the cardinality constraint* $\sum_i x_i = d$ *and UP enforce GAC on any partial assignment on variables* x_i*.*

Proof. We sketch the proof: Clauses (3) and (4) translate directly from the definition of the auxiliary variables representing cumulative sums. Clauses (1) and (2) represent the domain encoding of the bounds on the cumulative sums.

For GAC we need to show that given a partial assignment and assuming a variable x_i is not unit although should have been propagated. This will lead to a contradiction by case analysis on true and false of x_i and the clauses (3) and (4). In either case we are left with at least one binary clauses with index j as UP did not enforce x_i and by the definition of $y_{i,j}$ we have a contradiction.

Note that the given encoding is not the most compact for a single cardinality constraint. However, the auxiliary variables are reused in the following section to encode a sequence of AtMost constraints.

Fig. 2. Taking the counter of Figure 1 and let x_i be true for position 2 and 7. Then the resulting assignment after UP to the auxiliary variables is given in this table.

4.2 Extending to AtMostSeqCard

AtMostSeqCard is a specialized Sequence constraint: Given a sequence of boolean variables of which exactly d have to be true and each window of size q cannot contain more than u true variables. More formally:

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

To detect dis-entailment on this constraint by the CNF decomposition we propose to decompose the cardinality constraints by the counter encoding of the previous section and add the following binary clauses:

$$\bigwedge_{\substack{i \in \{1 \ldots n\} \\ i-q \geq 0}} \bigwedge_{\substack{j \in \{1 \ldots d+1\} \\ j-u \geq 0}} \neg y_{i,j} \vee y_{i-q,j-u} \tag{6}$$

Theorem 1. *The clauses of counter encoding (1) to (5) with the clauses of (6) are logically equivalent to the AtMostSeqCard . Moreover it detects dis-entailment with UP and thus GAC with failed literal test.*

Proof. Idea: We need to show that together with GAC on the cardinality constraints and clauses (6) we enforce bounds consistency on the auxiliary cumulative sums. The proof could use the construction in [Bacchus, 2007] on the sequence constraint to show dis-entailment and the idea [Brand et al., 2007] to enforce bounds consistency on the cumulative sums.

4.3 Size of Encoding

Let $s = (n \cdot d -$ upper and lower triangle), then we require s auxiliary variables and $3 \cdot (s + n)$ binary clauses and $2 \cdot (s + n)$ ternary clauses. The term $n \cdot d$ dominates and the precise number of variables can be computed by the hight d and the slope u/q and some basic algebra, which leads to $d \cdot (u - q + \frac{d \cdot q}{u})$. So the number of variables s are

$$s = d \cdot n - d \cdot (u - q + \frac{d \cdot q}{u})$$

Thus the size of this encoding lies in $O(nd)$, but can be more compact if q and u are rather strict and/or d is close to n. For example AtMostSeqCard $(u = 4, q = 8, d = 12, n = 22)$ would have

$12 \cdot 22 - 12 \cdot (4 - 8 + \frac{12 \cdot 8}{4}) = 24$ variables.

Give two references here : [Bacchus, 2007] and [Bessiere et al., 2009] on decompositions of the sequence constraint and failed literal test and discuss size of the encoding.

The power of the binary clauses are best illustrated by reusing the example [Siala et al., 2012] in Figures 3 and 4.

Fig. 3. Here we analyse the initial state of variables for a constraint with $u = 4, q = 8, d = 12, n = 22$. In this example we see that x_7, x_8, x_{15} and x_{16} should be false, this is detected by UP.

Fig. 4. State of the auxiliary variables for $u = 4, q = 8, d = 12, n = 22$ and choices x_1 and x_{13} to true and x_{12}, x_{14} and x_{21} to false. Notice the amount of propagation due to the clauses of the AtMostSeqCard constraint, also notice that variable x_1 was a redundant choice. Normal font= choice, bold = propagated, ()= propagated before, green arrow positive implication, red arrow negative implication by (6).

Fig. 5. Example where the the failed literal test detects dis-entailment. Let $u = 1$,$q = 2$,$d = 2$,$n = 5$ and let $x_3 = 1$, then UP does not enforce $x_2 = 0$ nor $x_4 = 0$. Setting them to true will lead to a conflict by UP through clauses (1),(2) and (6).

5 Encoding of Carsequencing

Let C be the index set identifying a class and O be the index set for options. The instance for a car sequencing problem is given by a mapping $m : C \to 2^O$, relating to each class a set of options. For each class we have a cardinality constraint and for each option a AtMostSeq constraint. From this we can construct for each option and for each class a AtMostSeqCard constraint on the sequence of the plan (since all classes have to be assigned we know the exact number of options). Each such AtMostSeqCard constraint is decomposed to CNF.

The second part of the encoding relates the counter for cars and for options as in the problem specification. We will present two ways to do so.

- Relating variables for cars and options
- Relating auxiliary variables in the counter encoding of cars and options.

For clarity we introduce two types of variables. Variables c denoting variables regarding cars and variables o denoting variables regarding options. Super and subscripts identify the precise meaning of the variable. For the variables that identify weather a car $k \in C$ or a option $l \in O$ is at position i we introduce c_i^k and o_j^l, notice that these variables correspond to the single variable of x_i of the construction in the previous section. The for the counter variables y we denote by $\hat{c}_{i,j}^k$ and $\hat{o}_{i,j}^l$.

5.1 Relating Cars and Option directly

In the first encoding we relate classes and options on each position. This is done by the following clauses. Let $m' : O \to 2^C$ be the reverse mapping relating an option to its set of classes. Then

$$\bigwedge_{i\in\{1\ldots n\}} \bigwedge_{\substack{k\in C \\ l\in m(k)}} \neg c_i^k \vee o_i^l \tag{7}$$

and the support

$$\bigwedge_{i\in\{1\ldots n\}} \bigwedge_{l\in O} \left(\neg o_i^l \vee \bigvee_{k\in m'(l)} c_i^k \right) \tag{8}$$

need to hold. We refer to this encoding as E_3.

Remark: Notice in an ASP encoding this would be modelled by one rule and the completion semantics covers for the reverse case.

5.2 An Encoding only Relating Auxiliary Variables

Here we will show that omit variables c_i^k and o_i^l. The encoding builds entirely on the auxiliary variables $\hat{c}_{i,j}^k$ and $\hat{o}_{i,j}^l$ and their relationship.

This encoding will enforce more global propagation that is not covered by the previous encodings and as such has stronger properties taking the auxiliary variables into account! give example which propagation is missing .

Here the idea:

For each position in the sequence, the sum of true $\hat{c}^k$ for cars k that have option l need to be the same size as the number of true $\hat{o}^l$ for option l. As a formula, for all positions i and options $l \in O$ it has to hold:

$$\sum_j o_{i,j}^l = \sum_{k \in m'(l)} \sum_j c_{i,j}^k$$

With this we only miss the restriction that exactly one class per position. This can be enforced by introducing an option that all classes have.

The decomposition of the above equality can be done by a sorting network (as in [Eén and Sörensson, 2006], sorting networks see [Batcher, 1968]) and the size should be $n \cdot log^2(d)$ in the hight d of the counter. For pure cardinality constraints this can be improved in a CNF encoding as in [Asín et al., 2011,Codish and Zazon-Ivry, 2010], but here we face an equivalence which needs a full sorter. Maybe the recursive structure of the cumulative sum can be exploited in this construction, since in position $i-1$ we sorted the sequence with up to one difference with positions left and right of it.

We refer to this encoding as E_4.

6 CP Model: Cumulative Sums

This encoding uses the idea behind sequence constraint in [Brand et al., 2007] and we adapt it here for the AtMostSeqCard with redundant constraints. Let S_i be an integer variable encoding the partial sum of positions $1 \dots n$. Then we post the following linear constraints and enforce bounds consistency on all. For all positions i and demands d, and the mapping between cars k and options l:

$$S_{i-1} \leq S_i \tag{9}$$
$$S_i \leq S_{i-1} + 1 \tag{10}$$
$$S_i = S_{i-1} + x_i \tag{11}$$
$$S_i \leq S_{i-q} + u \tag{12}$$
$$S_n = d \tag{13}$$
$$S_i^l = \sum_{k \in m'(l)} S_i^k \tag{14}$$

With this set of constraints we can simulate the SAT encodings E_3 and E_4, depending on the choice of constraints and variables.

6.1 Symmetry Breaking Constraints

A simple symmetry breaking constraint is restricting the class of the first car in the sequence to be of a lower or equal class than the last car in sequence. This is correct we can read the sequence from the beginning to the end or the other way around and the capacity constraints are equally satisfied. There is one interesting point to be made, formulating this constraint with boolean variables. We can either multiply all combinations of violated combinations of cars. The size is n^2 binary clauses, n is the number of classes. However, we can reuse the auxiliary variables from the constraint that enforces exactly one car in each position. Then we introduce no new variables and express this constraint in size $n - 1$ binary clauses.

7 Evaluation

We take the classification of the instances in the benchmark introduced by [Siala et al., 2012]. Set1 consists of 70 simple instances, Set2 of 4 SAT, Set3 of 4 UNSAT and Set4 of 30 new hard SAT and UNSAT instances.

We focus in our analysis on encoding E_3. Include comparison with E_1 till E_4 (still ongoing). See Table 3

The results given in this section can robustly be reproduced by current sat solvers. We compared newest version of minisat, lingeling, cryptominisat, glucose and clasp and they all consistently find solutions with the default strategy within 30 min runtime. In the following Table 1 we take the solved instances by lingeling:

Table 1.

	set1	**set2**	**set3**	**set4**
sat	70	4	0	7
unsat	0	0	4	13
unknown	0	0	1	10

This is comparable - sometimes considerable better - than the experimental sections of most papers evaluating the car sequencing problem on special constraints (CP) or specialized heuristics (e.g. branch and bound) or optimisation (IP) [van Hoeve et al., 2009,Siala et al., 2012,Gravel et al., 2005].

For the set 4 a more detailed view is interesting as the benchmark introduces new hard instances of the car sequencing problem, see Table 2.

Within 1h we can solve 7 satisfiable instances and prove 13/23 instances to be unsatisfiable. There is to the best of our knowledge no paper that treats these instances as a decisions problem and proves UNSAT on any instances.

8 Further Work

– Optimisations: there are two definitions of the cost function for the car sequencing problem. First extend the plan by class without options and solve

Table 2. Solutions to the benchmark proposed in [Gravel et al., 2005] with minimum violations found on the target function in their experiments (violated capacity of options per window) by a local search method and compared to solutions on the decision version SAT encoding with lingeling (LING).

name	**min**	**LING**	**sec**
200-01	0	SAT	189.9
300-01	0	SAT	315.7
400-01	1	?	-
200-02	2	?	-
300-02	12	?	-
400-02	16	?	-
200-03	4	UNSAT	70.2
300-03	13	UNSAT	873.0
400-03	9	UNSAT	88.1
200-04	7	UNSAT	19.7
300-04	7	UNSAT	33.6
400-04	19	UNSAT	83.2
200-05	6	UNSAT	543.7
300-05	29	UNSAT	9.7
400-05	0	SAT	2146.1
200-06	6	?	-
300-06	2	?	-
400-06	0	SAT	605.4
200-07	0	SAT	30.2
300-07	0	SAT	122.6
400-07	4	?	-
200-08	8	?	-
300-08	8	UNSAT	65.9
400-08	4	?	-
200-09	10	UNSAT	350.4
300-09	7	?	-
400-09	5	UNSAT	220.9
200-10	19	UNSAT	9.9
300-10	21	UNSAT	18.3
400-10	0	SAT	468.1

Table 3. Comparison of the three encodings (E_4 on the way). Times are given in seconds, time out 1800 seconds, solver lingeling, default configuration.

inst	E_1	E_2	E_3	inst	E_1	E_2	E_3
p00	-	16	5	p54	-	58.5	5.7
p01	-	15.9	8.6	p55	622.3	90.2	11.1
p02	-	17.9	11.4	p56	756.1	65.7	13.5
p03	-	19.4	7.8	p57	981.2	86.5	8.6
p04	-	-	-	p58	-	76.8	11.1
p05	-	50.5	17.7	p59	-	73.4	11.4
p06	-	19.6	9.1	p60	-	122.1	6
p07	-	8.9	3.5	p61	118.7	54.7	5.7
p08	-	18	8.5	p62	-	67.3	5.9
p09	92.5	40.3	6.3	p63	1042.3	140.9	7.1
p10	53.4	35.6	5.2	p64	-	106	18.9
p11	104.6	75.3	6	p65	-	87.6	10.7
p12	60.6	66	5.7	p66	-	125.3	7
p13	66.5	89.9	3	p67	-	131.8	17
p14	115.1	93	6.3	p68	508.3	64.8	13.3
p15	88.8	57.2	5.9	p69	-	54.9	10.8
p16	76.7	62.7	5.6	p70	-	111.4	12.6
p17	93.1	69.3	6	p71	-	119.8	10.8
p18	91.5	70.8	5.4	p72	-	140.8	11.4
p19	79.2	87.4	6.4	p73	-	108	10.8
p20	47	69	5.5	p74	122.8	54.1	5.7
p21	208.4	72.2	6.6	p75	-	76.1	22.6
p22	76.3	88.9	5.9	p76	-	101.4	11.4
p23	95.2	41.8	6.8	p77	-	49.9	11
p24	142.1	35.6	6.1	p78	-	92.4	12.6
p25	176.5	40.2	10.2	p200_01	-	202.4	213.5
p26	149.4	131.9	5.5	p200_02	-	-	-
p27	70.1	85.2	7.7	p200_03	-	159.8	79.7
p28	63.8	61.8	7.6	p200_04	95	42.9	20.9
p29	91	41.4	10.8	p200_05	-	449.3	391.1
p30	241.2	122.9	9.6	p200_06	-	-	-
p31	532	106.7	6.5	p200_07	-	390.6	35.4
p32	97	80.2	5.9	p200_08	-	-	-
p33	158.9	130.2	6.5	p200_09	76.6	125.3	697
p34	116.9	82.8	6.1	p200_10	33.6	77.4	10.6
p35	94.2	76.3	6.7	p300_01	-	-	194.7
p36	319	48.2	5.5	p300_02	-	-	-
p37	361.2	104.3	6.7	p300_03	615	653.3	553.8
p38	98.3	102.9	6	p300_04	81.6	73.3	30.6
p39	58.4	56	6	p300_05	60.7	752.7	11.7
p40	347.4	76.5	6.2	p300_06	-	-	-
p41	226.3	60	6.9	p300_07	-	-	177.5
p42	191.2	61.8	6.2	p300_08	275.8	280.6	54.9
p43	154.7	69	6.9	p300_09	-	-	-
p44	1287.8	102.1	7.1	p300_10	213.7	405.6	22
p45	123	101.5	8.8	p400_01	-	-	-
p46	282.7	71.3	11.6	p400_02	-	-	-
p47	900.7	85	7	p400_03	-	926.3	103.6
p48	86.9	129.3	6.7	p400_04	790.9	205.6	73.8
p49	174.4	61.5	10.1	p400_05	-	-	709.7
p50	238.6	98.3	10.8	p400_06	-	-	647.9
p51	507	109	7	p400_07	-	-	-
p52	718.5	90.1	6.1	p400_08	-	-	-
p53	117.6	61.6	7	p400_09	535.4	640.4	247.6

the corresponding decision problem. The task is then to minimize the number demand for such a class in order to solve the problem. . And the second version is minimize the number of windows that exceed the capacity constraint on their options. It would be interesting to compare both definition and to evaluate against published results in the literature. There are still gaps between known upper and lower bounds.

- There is a natural extension of the AtMostSeqCard constraint that to a cyclic version and in the same and natural way we can extend the encoding given above. Todo: find benchmarks that could require cyclic constraints.
- The Sequence constraint consists of a sequence of among constraints and we should compare this encoding to the known CNF encodings and filtering algorithms in the literature.
- Analyzing the quality of SAT encodings our results suggests to evaluate the consistency archived on the auxiliary variable introduced. E.g. in the encoding E_4 we had a stronger notion of propagation, as also on all partial assignment on $y_{i,j,k}$ we archive GAC.
- Another interesting aspect of encoded equality relating option and class counter is its relation to the UTVPI constraint (see [Seshia et al., 2007] for a recent treatment). Sorting Networks with order encoded variables give an interesting application for decomposing a more similar type of constraints. Let x_i, y be integer variables and d a fixed integer. Then there is a decomposition into CNF of the constraint

$$\sum_i x_i = y + d$$

such that by an order encoding on x_i and y we enforce bounds consistency on the equality. This construction uses the theory of cardinality networks. To our knowledge this has not been analysed in the literature.

References

[Asín et al., 2011] Asín, R., Nieuwenhuis, R., Oliveras, A., and Rodríguez-Carbonell, E. (2011). Cardinality Networks: a theoretical and empirical study. *Constraints*, 16(2):195–221.

[Bacchus, 2007] Bacchus, F. (2007). GAC Via Unit Propagation. In *CP*, pages 133–147.

[Batcher, 1968] Batcher, K. E. (1968). Sorting Networks and Their Applications. In *AFIPS Spring Joint Computing Conference*, pages 307–314.

[Bessiere et al., 2009] Bessiere, C., Katsirelos, G., Narodytska, N., and Walsh, T. (2009). Circuit Complexity and Decompositions of Global Constraints. In *IJCAI*, pages 412–418.

[Brand et al., 2007] Brand, S., Narodytska, N., Quimper, C.-G., Stuckey, P. J., and Walsh, T. (2007). Encodings of the Sequence Constraint. In *CP*, pages 210–224.

[Codish and Zazon-Ivry, 2010] Codish, M. and Zazon-Ivry, M. (2010). Pairwise Cardinality Networks. In *LPAR (Dakar)*, pages 154–172.

[Eén and Sörensson, 2006] Eén, N. and Sörensson, N. (2006). Translating Pseudo-Boolean Constraints into SAT. *JSAT*, 2(1-4):1–26.

[Gent and Walsh, 1999] Gent, I. P. and Walsh, T. (1999). CSP_{LIB}: A Benchmark Library for Constraints. In *CP*, pages 480–481.

[Gravel et al., 2005] Gravel, M., Gagné, C., and Price, W. L. (2005). 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.

[Seshia et al., 2007] Seshia, S. A., Subramani, K., and Bryant, R. E. (2007). On Solving Boolean Combinations of UTVPI Constraints. *JSAT*, 3(1-2):67–90.

[Siala et al., 2012] Siala, M., Hebrard, E., and Huguet, M.-J. (2012). An Optimal Arc Consistency Algorithm for a Chain of Atmost Constraints with Cardinality. In *CP*, pages 55–69.

[Tamura et al., 2009] Tamura, N., Taga, A., Kitagawa, S., and Banbara, M. (2009). Compiling finite linear CSP into SAT. *Constraints*, 14(2):254–272.

[van Hoeve et al., 2006] van Hoeve, W. J., Pesant, G., Rousseau, L.-M., and Sabharwal, A. (2006). Revisiting the Sequence Constraint. In *CP*, pages 620–634.

[van Hoeve et al., 2009] van Hoeve, W. J., Pesant, G., Rousseau, L.-M., and Sabharwal, A. (2009). New filtering algorithms for combinations of among constraints. *Constraints*, 14(2):273–292.