Translating the At-Most-One Constraint into SAT

Pedro Barahona[1], Steffen Hölldobler[2] and Van Hau Nguyen[2]

[1] Departamento de Informática, Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
pb@fct.unl.pt
[2] International Center for Computational Logic
Technische Universität Dresden, 01062 Dresden, Germany
hau,sh@iccl.tu-dresden.de

Abstract. One of the most widely used constraint during the process of translating a practical problem into a propositional satisfiability (SAT) instance is the at-most-one (AMO) constraint. This paper proposes a new encoding for the AMO constraint, the so-called *bimander* encoding which can be easily extended to encode cardinality constraints. Experimental results reveal that the new encoding is competitive. We will prove that the *bimander* encoding allows unit propagation to achieve arc consistency. Furthermore, we show that a special case of the *bimander* encoding outperforms the *binary* encoding, a widely used encoding, in all our experiments.

1 Introduction

An increasing number of real-world applications in computer science can be expressed as constraint satisfaction problems (CSPs) [17]. To utilize state-of-the-art SAT solvers, CSPs need to be encoded as SAT instances (see [20,19,16]). There does not seem to be general knowledge why a particular encoding performs better than others. In this study, we will compare different encodings with respect to the following features:

- the number of auxiliary variables required,
- the number of clauses required,
- the number of variables per clause,
- the strength of the encoding in terms of performance of unit propagation in SAT solvers,
- the runtime of a SAT solver on benchmark problems.

This paper propose a new way of encoding the at-most-one (AMO) constraint, which is one of the most common constraint during the process of translating a CSP into SAT ([21,10,18,14,7,9]). The new encoding, named the so-called *bimander* encoding can be easily extended to cardinality constraints. We will show that one special case of the *bimander* encoding outperforms the *binary* encoding [10] in all our experiments. The *bimander* encoding allows unit propagation (UP) to preserve arc consistency, one of the most important technique in Constraint Programming (see [6]).

2 Preliminaries

We adopt notions and notations from [9]. Let $X = \{X_i \mid 1 \leq i \leq n, n \in \mathbb{N}\}$ be a finite set of propositional variables, let A be a finite, possibly empty set of auxiliary propositional variables, and let $\phi(X, A)$ be a propositional formula in conjunctive normal form (CNF) encoding the constraint $\leq_1 \ (X_1, ..., X_n)$. The encoding $\phi(X, A)$ is *correct* if and only if:

- any (partial) assignment $\hat{x}$ that satisfies $\leq_1 \ (X_1, ..., X_n)$ can be extended to a complete assignment that satisfies $\phi(X, A)$, and
- for any (partial) assignment $\hat{x}$ for X which assigns more than one variable of X to *TRUE*, unit propagation (UP) detects a conflict, i.e., repeated applications of UP yield the empty clause.

UP plays a crucial role in SAT solving as modern SAT solvers [15], whereas arc consistency is one of the most important techniques in CSP solvers [6]. Therefore, when translating a CSP to a SAT instance one should pay much attention to determine whether UP on the resulting SAT instance enforces arc consistency. UP of a SAT encoding of the constraint $\leq_1 \ (X_1, ..., X_n)$ achieves the same pruning as arc consistency on the original CSP if the following holds [9]:

- at most one propositional variable in X is assigned to *TRUE*, and if
- any variable $X_i \in X$ is assigned to *TRUE*, then all the other variables occurring in X must be assigned to *FALSE* using UP.

For the convenience, $AMO(X)$ denotes the at-most-one clauses for the set of propositional variables X, and we illustrate the new encoding on a running example through the set consisting of 8 Boolean variables, $X = \{X_1, ..., X_8\}$.

3 The Bimander Encoding

The new encoding, the so-called *bimander* encoding, is based on both the ideas of the *binary* encoding and the *commander* encoding. Similarly to the *commander* encoding, with a given positive number m, $1 \leq m \leq n$, we partition a set of propositional variables $X = \{X_1, ..., X_n\}$ into m disjoint subsets $\{G_1, ..., G_m\}$ such that each subset G_i consists of $g = \lceil \frac{n}{m} \rceil$ variables. However, we introduce a set of auxiliary propositional variables $B_1, ..., B_{\lceil log_2 m \rceil}$ like in the *binary* encoding. The variables $B_1, ..., B_{\lceil log_2 m \rceil}$ play the role of the commander variables in the *commander* encoding. The *bimander* encoding is the conjunction of the following clauses:

1. At most *one* variable in each subset can be *TRUE*. We encode this constraint for each subset $G_i, 1 \leq i \leq m$, by using the *pairwise* encoding:

$$\bigwedge_{i=1}^{m} \langle AMO(G_i) \rangle,$$

In our running example we choose $m = \lceil \sqrt{n} \rceil = 3$ to obtain:

$$AMO(X_1, X_2, X_3) \wedge AMO(X_4, X_5, X_6) \wedge AMO(X_7, X_8).$$

2. The following clauses are generated by the constraints between each variable and commander variables in a subset:

$$\bigwedge_{i=1}^{m} \bigwedge_{h=1}^{g} \bigwedge_{j=1}^{\lceil log_2 m \rceil} \bar{X}_{i,h} \vee \phi(i, j).$$

where $\phi(i, j)$ denotes B_j (or $\bar{B}_j$) if the bit j of $i-1$ represented by a unique binary string is 1 (or 0).

The following set of clauses is generated for the running example:

$$\begin{array}{cccccccc} \bar{X}_1 \vee \bar{B}_1 & \bar{X}_2 \vee \bar{B}_1 & \bar{X}_3 \vee \bar{B}_1 & \bar{X}_4 \vee B_1 & \bar{X}_5 \vee B_1 & \bar{X}_6 \vee B_1 & \bar{X}_7 \vee \bar{B}_1 & \bar{X}_8 \vee \bar{B}_1 \\ \bar{X}_1 \vee \bar{B}_2 & \bar{X}_2 \vee \bar{B}_2 & \bar{X}_3 \vee \bar{B}_2 & \bar{X}_4 \vee \bar{B}_2 & \bar{X}_5 \vee \bar{B}_2 & \bar{X}_6 \vee \bar{B}_2 & \bar{X}_7 \vee B_2 & \bar{X}_8 \vee B_2 \end{array}$$

We prove the *Correctness* and the *Complexity* in [13] where we also show that the *bimander* encoding maintains arc consistency. The *bimander* encoding can be generalized to encode the *at-most-k* constraint. Furthermore, we point out that the *pairwise* encoding and the *binary* encoding are two special cases of the *bimander* encoding (see [13]).

4 Comparison

Table 1 presents the key features of many approaches for encoding the AMO constraint (column *enc*). The columns *clauses* and *aux vars* depict the number of clauses required and auxiliary variables, respectively. The column *AC* indicates whether UP achieves arc consistency. The column *origin* refers to the original publications where the encoding had been introduced. *m* denotes the disjointed subsets by dividing the set of propositional variables $\{X_1, ..., X_n\}$ in the *bimander* encoding.

Table 1. A summary of almost all known encodings of the AMO constraint, including several encodings mainly used for cardinality constraints.

enc	*clauses*	*aux vars*	*AC*	*origin*
pairwise	$\binom{n}{2}$	0	yes	none
linear	$8n$	$2n$	no	[21]
totalizer	$O(n^2)$	$O(nlog(n))$	yes	[4]
binary	$nlog_2 n$	$\lceil log_2 n \rceil$	yes	[10]
sequential	$3n - 4$	$n - 1$	yes	[18]
sorting networks	$O(nlog_2^2 n)$	$O(nlog_2^2 n)$	yes	[8]
commander	$\sim 3n$	$\sim \frac{n}{2}$	yes	[14]
product	$2n + 4\sqrt{n} + O(\sqrt[4]{n})$	$2\sqrt{n} + O(\sqrt[4]{n})$	yes	[7]
card. networks	$6n - 9$	$4n - 6$	yes	[3]
PHFs-based	$nlog_2 n$	$\lceil log_2 n \rceil$	yes	[5]
bimander	$\frac{n^2}{2m} + nlog_2 m - \frac{n}{2}$	$log_2 m, 1 \leq m \leq n$	yes	this paper
bimander ($m = \frac{n}{2}$)	$nlog_2 n - \frac{n}{2}$	$\lceil log_2 n \rceil - 1$	yes	this paper

As we can see in Table 1, the *bimander* encoding requires the least auxiliary variables – with the exception of the *pairwise* encoding – among known encodings. The *totalizer* encoding proposed by Bailleux al et. [4] requires clauses of size at most *3*, and the *commander* encoding proposed by Klieber and Kwon [14] needs m (number of disjointed subsets) clauses of size $\lceil \frac{n}{m} + 1 \rceil$, whereas the *product*, *sequential*, *binary* and *bimander* encodings require only binary clauses.

5 Experimental Evaluation

Our experiments use CLASP 2 [11] with default configuration on a 2.66-GHz Intel Core 2 Quad processor with 3.8 GB of memory. Bold font indicates the minimum time for each benchmark. We abbreviate *pairwise*, *sequential*, *commander*, *binary*, *product*, and *bimander* encodings as *pw*, *seq*, *cmd*, *bin*, *pro* and *bim*, respectively. For the *commander* encoding, the set of variables is recursively divided into 2 disjoint subsets since it gives best average results in our experiment. In case of the *bimander* encoding, we have considered two different values for the parameter m, viz. $m = \sqrt{n}$ and $m = \frac{n}{2}$.

Table 2. A comparison of the run times for Pigeon-Hole problems. Run times are in seconds.

enc	*pw*	*seq*	*cmd*	*bin*	*pro*	*bim*($\sqrt{n}$)	*bim*($n/2$)
10	2.16	0.73	0.56	0.80	0.22	0.33	**0.22**
11	22.15	5.79	4.46	6.59	6.13	5.10	**2.10**
12	244.59	117.83	43.27	29.52	43.21	38.19	**26.06**
13	$>$3600.00	1604.14	352.53	142.60	736.25	546.91	**64.91**
14	$>$3600.00	$>$3600.00	$>$3600.00	1271.24	$>$3600.00	$>$3600.00	**560.03**
average	$>$1493.78	$>$1065.69	$>$ 800.16	290.15	$>$ 877.16	$>$ 838.10	**130.66**

Table 3. A comparison of run times for satisfiable Quasigroup With Holes (QWH) problems [2]. Run times are in seconds.

enc	*pw*	*seq*	*cmd*	*bin*	*pro*	*bim*($\sqrt{n}$)	*bim*($n/2$)
qwh.order30.holes320	0.46	0.28	0.23	0.25	0.23	**0.20**	0.22
qwh.order35.holes405	3.62	3.51	10.35	6.51	5.73	**1.60**	2.14
1 qwh.order40.holes528	134.71	115.62	124.26	120.47	241.20	**58.90**	159.21
qwh.order40.holes544	39.26	**14.57**	47.82	123.72	46.7	70.81	154.03
qwh.order40.holes560	121.74	65.36	55.68	119.66	33.16	**21.22**	53.27
qwh.order33.holes381	58.73	435.90	174.29	94.22	108.03	**12.74**	92.30
average	358.52	635.24	412.63	464.83	435.05	**165.47**	461.17

Table 4. A comparison of run times for All-Interval Series (AIS) problems (see prob007 in [12]). Run times are in seconds. *sol* shows the number of all solutions of the corresponding instance.

enc	*pw*	*seq*	*cmd*	*bin*	*pro*	*bim*($\sqrt{n}$)	*bim*($n/2$)	*sol*
7	0.05	0.03	0.02	0.02	0.05	**0.01**	0.02	32
8	0.56	1.07	0.63	**0.20**	0.49	0.62	0.62	40
9	5.33	8.92	0.37	0.27	5.61	0.33	**0.24**	120
10	61.72	104.02	1.72	1.58	60.71	1.95	**1.46**	296
11	972.54	1387.67	11.96	8.94	269.43	11.34	**6.72**	648
12	$>$3600.00	$>$3600.00	78.91	49.24	$>$3600.00	69.52	**43.81**	1328
13	$>$3600.00	$>$3600.00	517.72	356.64	$>$3600.00	504.61	**276.34**	3200
14	$>$3600.00	$>$3600.00	3200.21	2748.69	$>$3600.00	3537.74	**2005.18**	9912
average	$>$1480.02	$>$1537.71	476.44	395.69	$>$1392.03	515.76	**291.79**	

Table 5. A comparison of run times for satisfiable Hamiltonian Cycle (HC) instances (taken from [1]). Run times are in seconds.

enc	*pw*	*seq*	*cmd*	*bin*	*pro*	*bim*($\sqrt{n}$)	*bim*($n/2$)
miles750	135.48	25.42	**13.67**	38.19	14.18	32.55	22.92
miles1000	67.77	10.93	7.65	**7.38**	12.45	9.52	8.19
miles1500	30.01	3.30	2.60	2.95	**2.46**	3.74	3.16
queen10_10	13.87	4.16	**3.54**	3.77	3.68	4.00	3.75
queen11_11	32.34	9.75	8.32	8.43	8.41	9.23	**8.16**
queen12_12	**1.73**	22.46	20.13	21.49	18.43	20.44	21.13
queen13_13	3.10	40.99	38.43	1.58	36.30	1.45	**1.39**
queen14_14	5.17	2.47	2.53	2.42	**2.09**	2.27	2.20
queen15_15	7.75	3.64	3.42	3.76	**3.17**	3.47	3.37
queen16_16	11.26	4.80	**5.21**	5.44	5.14	5.37	5.25
average	30.84	12.79	10.55	9.54	10.63	9.20	**7.95**

Throughout above experiments, we showed that two cases of the *bimander* encoding, with certain parameters $m = \sqrt{n}$ and $m = \frac{n}{2}$, are very competitive. In particular, the encoding in case $m = \sqrt{n}$ performs clearly the best on QWH instances, and rather well on the other benchmarks, whereas the encoding in case $m = \frac{n}{2}$ is clear the best on the Pigeon-Hole, AIS, and HC problems, and acceptable on the QWH problem.

6 Conclusions and Future Works

Compared to many other well-known AMO encodings, the new encoding, *bimander* encoding, not only requires the least auxiliary variables (with the exception of the *pairwise* encoding which does not erequire any auxiliary variables at all), but also binary clauses. Although the *commander* encoding and the *bimander* encoding use the same approach, the *commander* encoding requires clauses of size $\lceil \frac{n}{m} + 1 \rceil$ (where m is the number of disjoint subsets), whereas the *bimander* encoding requires only binary clauses. We believe that this helps the *bimander* encoding to perform better than the *commander* encoding in our experimental evaluation. Moreover, the *bimander* encoding has the advantage of high scalability, and it can easily be adjusted in terms of the number of additional propositional variables to obtain particular encodings. For example, the *pairwise* or *binary* encodings are special cases of the *bimander* encoding.

The special case, by setting $m = \lceil \frac{n}{2} \rceil$ disjoint subsets, of the *bimander* encoding requires fewer auxiliary variables and clauses and shows a better performance in all our experiments than the *binary* encoding [10].

In practice, the *bimander* encoding is practical and easy to implement. Our results reveal that two particular cases of the *bimander* encoding are very competitive in a comparison with other well-known encodings.

A future research is to study how the number of disjoint subsets could affect the *bimander* encoding in realistic problems. It would be particularly useful to extend our findings to the at-most-k constraint.

References

1. A computational symposium at cornell university, ithaca, ny, usa, 2002. http://mat.gsia.cmu.edu/COLOR03/

2. Achlioptas, D., Gomes, C.P., Kautz, H.A., Selman, B.: Generating satisfiable problem instances. In: Kautz, H.A., Porter, B.W. (eds.) Proceedings of the Seventeenth National Conference on Artificial Intelligence and 12th Conference on Innovative Applications of Artificial Intelligence, 2000, Austin, Texas, USA. pp. 256–261. AAAI Press / The MIT Press (2000)
3. Asín, R., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E.: Cardinality networks: a theoretical and empirical study. Constraints 16(2), 195–221 (2011)
4. Bailleux, O., Boufkhad, Y.: Efficient CNF encoding of boolean cardinality constraints. Principles and Practice of Constraint Programming 9th International Conference CP-2003 pp. 108–122 (2003)
5. Ben-Haim, Y., Ivrii, A., Margalit, O., Matsliah, A.: Perfect hashing and CNF encodings of cardinality constraints. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7317, pp. 397–409. Springer (2012)
6. Bessiere, C.: Chapter 3 Constraint Propagation, vol. 2, pp. 27 – 81. Elsevier (2006)
7. Chen, J.C.: A new SAT encoding of the at-most-one constraint. In: Proc. of the Tenth Int. Workshop of Constraint Modelling and Reformulation (2010)
8. Eén, N., Sörensson, N.: Translating pseudo-boolean constraints into sat. Journal on Satisfiability, Boolean Modeling and Computation 2, 1–26 (2006)
9. Frisch, A.M., Giannoros, P.A.: SAT encodings of the at-most-k constraint. some old, some new, some fast, some slow. In: Proc. of the Tenth Int. Workshop of Constraint Modelling and Reformulation (2010)
10. Frisch, A.M., Peugniez, T.J., Doggett, A.J., Nightingale, P.W.: Solving non-boolean satisfiability problems with stochastic local search: A comparison of encodings. J. Autom. Reason. 35, 143–179 (October 2005)
11. Gebser, M., Kaufmann, B., Schaub, T.: The conflict-driven answer set solver clasp: Progress report. In: Erdem, E., Lin, F., Schaub, T. (eds.) LPNMR 2009, Potsdam, Germany, September 14-18, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5753, pp. 509–514. Springer (2009)
12. Hnich, B., Miguel, I., Gent, I.P., , Walsh, T.: Csplib is a library of test problems for constraint solvers. `http://www.csplib.org/`, [Online; accessed 24-August-2012]
13. Hölldobler, S., Nguyen, V.H.: An efficient encoding of the at-most-one constraint. Tech. rep., Knowledge Representation and Reasoning Group, Technische Universität Dresden, 01062 Dresden, Germany (2013)
14. Klieber, W., Kwon, G.: Efficient CNF encoding for selecting 1 from n objects. In: the Fourth Workshop on Constraint in Formal Verification(CFV) (2007)
15. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001. pp. 530–535. ACM (2001)
16. Prestwich, S.D.: CNF Encodings, pp. 75–98. IOS Press (2009)
17. Rossi, F., Beek, P.v., Walsh, T.: Handbook of Constraint Programming (Foundations of Artificial Intelligence). Elsevier Science Inc., New York, NY, USA (2006)
18. Sinz, C.: Towards an optimal CNF encoding of boolean cardinality constraints. In: CP 2005, Spain, October 2005, Proceedings. Lecture Notes in Computer, vol. 3709, pp. 827–831. Springer (2005)
19. Tamura, N., Taga, A., Kitagawa, S., Banbara, M.: Compiling finite linear CSP into SAT. Constraints 14(2), 254–272 (2009)
20. Walsh, T.: SAT v CSP. In: Principles and Practice of Constraint Programming - CP2000. Lecture Notes in Computer Science, vol. 1894, pp. 441–456. Springer (2000)
21. Warners, J.P.: A linear-time transformation of linear inequalities into conjunctive normal form. Information Processing Letters 68(2), 63–69 (1998)