Empirical Study on SAT-Encodings of the At-Most-One Constraint

Van-Hau Nguyen
Hung Yen University of Technology and Education
Hung Yen, Vietnam
haunv@utehy.edu.vn

Van-Quyet Nguyen
Hung Yen University of Technology and Education
Hung Yen, Vietnam
quyetict@utehy.edu.vn

Kyungbaek Kim
Chonnam National University
Gwangju 61186, South Korea
kyungbaekkim@jnu.ac.kr

Pedro Barahona
New University of Lisbon
2829-516 Caparica, Portugal
pb@fct.unl.pt

ABSTRACT

Propositional satisfiability solving (SAT) has been one of the most successful automated reasoning methods in the last decade in computer science by solving a wide range of both industrial and academic problems. One of the most widely used constraint during the process of translating a practical problem into a SAT instance is the at-most-one (AMO) constraint. Many studies on SAT encodings of the AMO constraint have been reported in the literature, however thorough comparisons of these encodings are largely missing. This paper not only provide such comparisons, but also discusses the similarity of the SAT encodings of the AMO constraint and SAT encodings of more general CSPs. More specifically, the paper empirically evaluates the most well-known AMO encodings on three state-of-the-art SAT solvers of a number of benchmarks and provides some guidelines for efficient SAT encodings of the AMO constraint leading.

CCS CONCEPTS

• **Theory of computation → Automated reasoning**; Constraint and logic programming.

KEYWORDS

SAT encodings, At-most-one constraint, SAT solving

ACM Reference Format:
Van-Hau Nguyen, Van-Quyet Nguyen, Kyungbaek Kim, and Pedro Barahona. 2020. Empirical Study on SAT-Encodings of the At-Most-One Constraint. In *The 9th International Conference on Smart Media and Applications (SMA 2020), September 17–19, 2020, Jeju, Republic of Korea.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3426020.3426170

1 INTRODUCTION

SAT solving comprises two essential phases: encoding a certain problem into a SAT instance, and then finding solutions by advanced SAT solvers. Notwithstanding the steadily increasing diffusion and availability of SAT solvers, understanding of SAT encodings is still limited and challenging. To use state-of-the-art SAT solvers for solving constraint satisfaction problems (CSPs) requires these to be previously encoded as SAT instances [1, 7, 18, 29, 31, 34, 38, 39]. Such encodings should not only be efficiently generated, but also lead to efficiently solving by SAT solvers. At present, mapping a CSP into a SAT instance is still largely regarded as more of an art than a science ([17, 24, 33, 39]), and detailed studies of different encodings are needed in order to better understand their behaviour in different settings.

Among the many encodings proposed to map CSPs into SAT [17, 30, 34, 38, 39], the most straightforward are the *direct encoding*, first described by Kleer [12] and further studied by Walsh [39], and the *support encoding* by Gent [17].

Both encodings require the at-least-one (ALO) and at-most-one (AMO) constraints to enforce that a CSP variable is assigned to a single value within its domain. If the ALO constraint can be easily encoded by a single SAT clause, the encoding of the AMO constraint is more complicated and has been intensively studied [10, 14, 25, 32, 35], also because many applications such as computer motographs [5], partial Max-SAT [3], or cardinality constraints [14] contain the AMO constraint.

Generally, different AMO encodings yield different sizes and different behaviour from the SAT solver being used. Since the understanding of why a particular encoding performs better than others is still largely missing, in this paper we compare different encodings with respect to the following features:

- the number of variables required (search space),
- the number of clauses required (overhead when propagating variable assignments),
- the strength of the encoding in terms of performance of unit propagation in SAT solvers (e.g. maintaining arc-consistency),
- the length of clauses (e.g. binary),
- the runtime of a SAT solver on benchmark problems.

In addition, other issues are addressed in this paper. Firstly, we note the similarities between some AMO encodings and the more general encoding of finite CSPs to SAT and the similarities of several AMO encodings that have been proposed albeit with different names. Next we show, for all benchmarks and different encodings of the strong correlation between runtime and the number of conflicts as well as between the number of conflicts and the number

SMA 2020, September 17–19, 2020, Jeju, Republic of Korea

ACM ISBN 978-1-4503-8925-9/20/09.
https://doi.org/10.1145/3426020.3426170

of decisions. Finally there are some general conclusions that may be drawn regarding the behaviour of the encodings in the different settings (benchmarks an solvers).

The structure of the paper is as follows. In Section 2, we briefly represent well-known SAT encodings of the AMO constraint, namely AMO encodings. In Section 3, we compare among these AMO encodings, and show the experimental evaluation. Finally, we conclude and outline future research in Section 4.

2 SAT ENCODINGS OF THE AT-MOST-ONE CONSTRAINT

In the direct or support encodings, if a propositional variable is used to represent the binding of a CSP variable to a particular value, then the AMO constraint requires that at most one of n propositional variables is bound to *true*. In the following sections, generally $AMO(X)$, $ALO(X)$, and $EO(X)$ denote the at-most-one, at-least-one ,and exactly-one clauses, respectively, for the set of propositional variables X.

In CSP solving, arc consistency is a quite effective technique for its tradeoff between the cost of the constraint propagation performed at each node in the search tree and pruning of the search space. Most DPLL SAT solvers [11, 21, 28] use unit propagation, also called Boolean constraint propagation, as its constraint propagation mechanism. Therefore, when translating a CSP to a SAT instance one should be concerned on whether unit propagation on the resulting SAT instance enforces arc consistency on the original CSP. For example, unit propagation of a SAT encoding of constraint $AMO(X_1, ..., X_n)$ achieves the same pruning as arc consistency on the original CSP (for the details, see [14, 32]).

2.1 The Pairwise Encoding

This encoding has several different names: the *naive encoding* [25, 37], the *pairwise encoding* [35, 36], or the *binomial encoding* [14]. In this paper, we refer to it as the *pairwise encoding*. The idea of this encoding is to express that no pair of two variables are simultaneously assigned to *true*, therefore as soon as one literal is assigned to *true*, all the others must be assigned to *false*:

$$\bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} (\bar{X}_i \vee \bar{X}_j)$$

The *pairwise encoding* is a traditional SAT encoding of the AMO constraint. Although this encoding needs no auxiliary variables, it requires a quadratic number of clauses. As a result, this encoding produces impractically large formulas on problems with large domains. The encoding allows unit propagation maintains arc consistency (see Table 1)

2.2 The Binary Encoding

Frisch et al. [15] proposed the *binary encoding*. Independently, Prestwich called it *bitwise* encoding [35] and used it to successfully solve a number of large instances of CSPs with a standard SAT solver.

The binary encoding requires new Boolean variables $B_1, ..., B_{\lceil log_2 n \rceil}$ in the following clauses:

$$\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{\lceil log_2 n \rceil} \langle \bar{X}_i \vee \phi(i,j) \rangle,$$

where $\phi(i, j)$ denotes B_j (or $\bar{B}_j$) if bit j of the binary representation of integer $i-1$ is 1 (or 0). The idea is to create the different sequences of $\lceil log_2 n \rceil$-tuples B_j such that whenever any X_i is assigned to *true* it is immediately inferred that the other variables $X_{i'}$ must be assigned to *false*, for any $1 \leq i' \neq i \leq n$.

There are two important remarks for the *binary encoding*. Firstly, unit propagation maintains arc consistency (see [14]). Secondly, the auxiliary variables, $B_1, ..., B_{\lceil log_2 n \rceil}$, used by the *binary encoding* are the exact variables in the *log encoding*, first proposed by Iwama and Miyazaki [23] and latter by Walsh [39], which encodes a finite CSP domain to SAT.

2.3 The Commander Encoding

Klieber and Kwon [25] described the *commander encoding* by dividing the set $X = \{X_1, ..., X_n\}$ of propositional variables into m, $1 \leq m \leq n$, disjoint subsets denoted by $\{G_1, ..., G_m\}$, and introducing a *commander* variable c_i for each subset G_i. The *commander encoding* is defined as follows.

(1) Exactly one variable in each set $G_i \cup \{\bar{c}_i\}$ is assigned to *true*:
$$\bigwedge_{i=1}^{m} EO(\{\bar{c}_i\} \cup G_i) = \bigwedge_{i=1}^{m} AMO(\{\bar{c}_i\} \cup G_i) \wedge \bigwedge_{i=1}^{m} ALO(\{\bar{c}_i\} \cup G_i).$$
where AMO can be encoded either by the *pairwise* or any other encoding.

(2) At most one commander variable is assigned to *true*.
$$\bigwedge_{i=1}^{m} AMO(c_i),$$
where AMO can be encoded either by any encoding including a recursive application of the *commander encoding*.

The *commander encoding* also allows unit propagation to preserve arc consistency (see [14]).

2.4 The Product Encoding

Chen [10] proposed an AMO encoding, named the *product encoding*, that uses two sets of auxiliary variables $U = \{u_1, ..., u_p\}$ and $V = \{v_1, ..., v_q$, forming a grid of $p \times q$ points (where $p \times q \geq n$, and

(1) Each variable $X_k, 1 \leq k \leq n$ is mapped onto a corresponding point (u_i, v_j), where $u_i \in U = \{u_1, ..., u_p\}$, and $v_i \in V = \{v_1, ..., v_q\}$.

(2) Then, the *product encoding* is obtained as:
$$AMO(X) = AMO(U) \wedge AMO(V) \bigwedge_{1 \leq i \leq p, 1 \leq j \leq q}^{1 \leq k \leq n, k=(i-1)q+j} ((\bar{X}_k \vee \bar{u}_i) \wedge (\bar{X}_k \vee \bar{v}_j)),$$

where $AMO(U)$ and $AMO(V)$ can be encoded by any encoding, including a recursive application of the *product encoding*.

Two notes on the *product encoding*: a) unit propagation on the *product encoding* achieves arc consistency [10]; and b) the auxiliary variables, (u_i, v_j) are similar to the variables used to encode a finite CSP domain into SAT in the *representative-sparse encoding*, proposed by Barahona et al. [7].

2.5 The Sequential Counter Encoding

By building a count-and-compare hardware circuit and translating this circuit to an equivalent CNF formula, Sinz [37] introduced an encoding of $\leq_k (X_1, ..., X_n)$ which he called the *sequential counter*

encoding. Here, we only consider the case $k = 1$, corresponding to the $AMO(X)$ constraint.

$$(\bar{X}_1 \vee s_1) \wedge (\bar{X}_n \vee \bar{s}_{n-1}) \bigwedge_{1<i<n} ((\bar{X}_i \vee s_i) \wedge (\bar{s}_{i-1} \vee s_i) \wedge (\bar{X}_i \vee \bar{s}_{i-1})), \quad (1)$$

where $s_i, 1 \leq i \leq n-1$, are auxiliary variables.

Again, two notes are worth mentioning for this encoding: a) the *sequential counter encoding* allows unit propagation to achieve arc consistency (see [14, 36]); and b) the auxiliary variables, s_i are exactly the variables in the *order encoding*, used by Tamura et al. [38], to translate finite CSPs to SAT, as noted, if indirectly, by Argelich et al. [3]

2.6 The Bimander Encoding

Similarly to the *commander encoding*, the *bimander encoding* partitions a set of propositional variables $X = \{X_1, ..., X_n\}$ into m disjoint subsets $\{G_1, ..., G_m\}$, $(1 \leq m \leq n)$, such that each subset G_i consists of $g = \lceil \frac{n}{m} \rceil$ variables. However, instead of commander variables, the *bimander encoding* introduces a set of auxiliary propositional variables $B_1, ..., B_{\lceil log_2 m \rceil}$ (like in the *binary encoding*) that play the role of the commander variables. 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*:

$$\bigwedge_{i=1}^{m} \langle AMO(G_i) \rangle. \quad (2)$$

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

$$\bigwedge_{i=1}^{m} \bigwedge_{h=1}^{g} \bigwedge_{j=1}^{\lceil log_2 m \rceil} \bar{X}_{i,h} \vee \phi(i,j), \quad (3)$$

where $\phi(i, j)$ denotes B_j (or $\bar{B}_j$) if bit j of the binary representation of integer $i - 1$ is 1 (or 0).

Nguyen and Mai discuss the *correctness* and the *complexity* in [32], where they also show that the *bimander encoding* maintains arc consistency.

3 COMPARISON AND EXPERIMENTAL EVALUATION

As mentioned in Section 1, we compare different encodings with respect to the following features:

- the number of variables required (search space),
- the number of clauses required (overhead when propagating variable assignments),
- the strength of the encoding in terms of performance of unit propagation in SAT solvers (e.g. maintaining arc-consistency),
- the length of clauses (e.g. binary),
- the performance of a SAT solver, including:
  - the runtime,
  - the number of decisions,
  - the number of conflicts,
  - the memory consumption (MB).

Next, we will discuss the first 4 features in Section 3.1 and the last feature in Section 3.2.

3.1 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 required clauses and auxiliary variables, respectively. The column *UPaAC* indicates whether the encoding has the UPaAC property. The column *origin* refers to the original publications where the encoding had been introduced. It is worth noticing that the *totalizer* encoding proposed by Bailleux al et. [5] requires clauses of size at most *3*, and the *commander encoding* proposed by Klieber and Kwon [25] needs m (number of disjointed subsets) clauses of size $\lceil \frac{n}{m} + 1 \rceil$, whereas the *product*, *sequential*, *binary* and *bimander encoding*s require only binary clauses.

The *sequential counter encoding* and the *bimander encoding* are generalized to encode general cardinality constraints, the *at-most-k* constraint (see [37] and [32], respectively). The other encodings are also generalized by Frisch and Giannaros [14]. The authors studies SAT encodings of the at-Most-k Constraint. The study of Frisch and Giannaros is the most closely related to our work, but only one benchmark (the Pigeon-Hole problem) and only one SAT solver, MiniSat, were used.

Marques-Silva and Lynce [36] point out that conflict-driven clause learning SAT solvers can exploit the properties of the *sequential counter encoding*.

Nguyen and Mai [32] point out that the *pairwise encoding* and the *binary encoding* are two special cases of the *bimander encoding*. Furthermore, the authors also show that the *relaxed ladder encoding* is exactly the *ladder encoding* without the redundant clauses. Consequently, the *relaxed ladder encoding* and the *sequential counter encoding* are identical. Argelich et al. [3] also noticed that the *sequential counter encoding* is a reformulation of a *regular encoding* [2]. In fact, it is a simple matter to prove that the *regular encoding* and the *ladder encodings* are identical.

Summarizing, the *relaxed ladder encoding* and the *sequential counter encoding* are identical. These encodings are obtained from the *ladder* or the *regular encoding* by removing redundant clauses. One should observe that Tamura et al. [38] used the *ladder structure* in the *order encoding* to translate CSPs to SAT instances in their SAT-based solving system. Bailleux et al. [6] also used this structure, in different name - *unary representation*, during their translation of cardinality constraints and pseudo-Boolean constraints to SAT formulas ([5, 6]).

Recently, Martins et. al [27] compared both encodings, the *sequential counter encoding* and the *ladder encoding*, and the experiment results obtained show very small differences between the two encodings.

In conclusion, we aim to show that the similarity of the *ladder*, *sequential*, *relaxed ladder*, *regular*, *unary representation*, and *order* encodings. We hope that this work could help the SAT community to avoid the confusion of these encodings.

3.2 Experimental Evaluation

All experiments reported in this section were performed on a 2.66 Ghz, Intel Core 2 Quad processor with 3.8 GB of memory, under Ubuntu 10.04. Runtimes reported in CPU-time are in seconds. The used solvers are *Riss* [26], *lingeling* [9], and *clasp* [16] (*clasp2.1.3-x86_64linux*) with default configurations, conflict-driven clause learning SAT solvers, which were ranked first on application and

Table 1: A summary of most well-known AMO SAT-encodings, where some encodings come from cardinality constraints noted by CAR.

Encodings	*clauses*	*aux vars*	*UPaAC*	*origin*
pairwise	$\binom{n}{2}$	0	yes	folklore
linear (CAR.)	$8n$	$2n$	no	[40]
totalizer	$O(n^2)$	$O(nlog(n))$	yes	[5]
binary	$nlog_2n$	$\lceil log_2n \rceil$	yes	[15]
sequential	$3n-4$	$n-1$	yes	[37]
sorting networks (CAR.)	$O(nlog_2^2n)$	$O(nlog_2^2n)$	yes	[13]
commander	$\sim 3n$	$\sim \frac{n}{2}$	yes	[25]
product	$2n+4\sqrt{n}+O(\sqrt[4]{n})$	$2\sqrt{n}+O(\sqrt[4]{n})$	yes	[10]
card. networks(CAR.)	$6n-9$	$4n-6$	yes	[4]
PHFs-based (CAR.)	$nlog_2n$	$\lceil log_2n \rceil$	yes	[8]
bimander	$\frac{n^2}{2m}+nlog_2m-\frac{n}{2}$	$log_2m, 1 \leq m \leq n$	yes	[22]

craft benchmarks in different categories at recent SAT competitions[1]. Due to space of limitations, we show several results among three SAT solvers since the other results are similars.

For the datasets, we choose 5 benchmarks which are well-known in Constraint Programming community: *Golomb Ruler* (CSPLIB prob006 in [19]), *Langford* (see prob024 in [19]), *Pigeon*, *Hidoku* (see [20]), *Open Shop*, and *All-interval Series* (see prob007 in [19]).

In the following section, we abbreviate *pairwise*, *sequential*, *commander*, *binary*, *product*, and *bimander* encodings as *pw*, *seq*, *cmd*, *bin*, *pro* and *bim*, respectively. For the AMO *commander*, and *bimander* encodings, the set of variables is recursively divided into 2 disjoint subsets since the encoding in that case conducted on our problems gives a best result in term of the average time.

Fig. 1 compares different AMO encodings on satisfiable Golomb ruler instances for lingeling. In terms of running times the AMO *commander* encoding is the best. The AMO *binary* and *pairwise* and encodings perform well, whereas the AMO *product*, *bimander* and *sequential* perform very poor. Regardless of different solvers, we observe that the runtime is somewhat related to the number of conflicts, whereas the number of conflicts is closely related to the number of decisions. In term of the memory consumed, a faster encoding tend to consume less than a slower encoding.

Since Lingeling and Riss3G do not offer the configuration for finding all the solution for a CNF instance, the langford problem is only performed by Clasp. A comparison of different AMO encodings performed by Clasp for finding all the number of solutions on langford instances is shown in Fig 2. It can be seen that the AMO *pairwise* encoding is worst than the others, while the AMO *sequential counter*, *product*, and *commander* encodings perform quite similarly, followed by the AMO *binary* and *bimander* encodings. Fig 2 also indicates that the runtime, the number of conflicts, and the number of decisions have a strong connection.

Fig. 3 summarizes the results of different AMO encodings performed by Lingeling on satisfiable Hidoku instances. Unlike by Lingeling, The AMO *pairwise* encoding performs worst, especially on two last instances. The five other encodings show no clear pattern. Fig. 4 shows the results of different AMO encodings performed by Lingeling on satisfiable Open Shop instances. Unlike for the previous problems, the running time has a weak relation to the number of conflicts.

[1]http://www.satcompetition.org

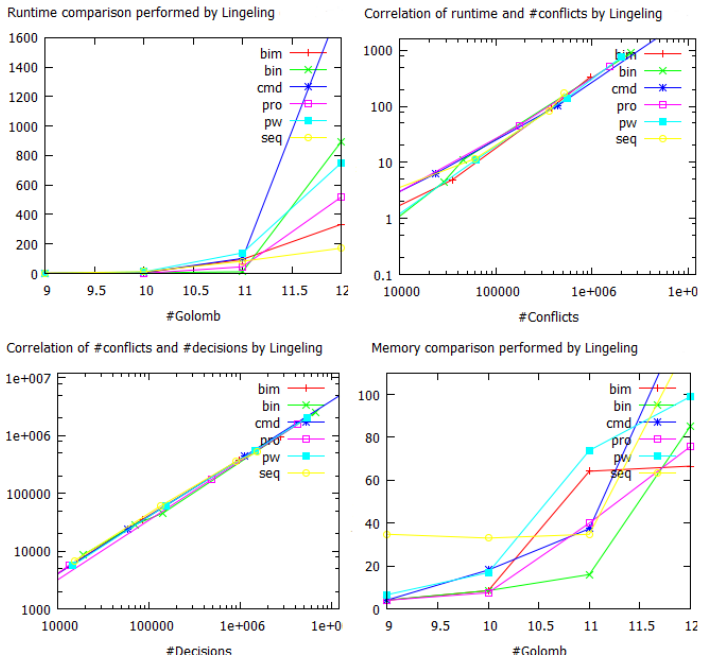

Figure 1: Lingeling on satisfiable Golomb ruler instances.

Since Lingeling and Riss3G do not offer the configuration for finding all the solution for a CNF instance, the all-interval series problem is only performed by Clasp.

Fig. 5 presents the result of different AMO encodings performed by Clasp for finding all the number of solutions on all-interval series instances. They show that, the AMO *pairwise* encoding is clear better than the others on last two instances. On the contrary, the AMO *sequential counter* encoding perform badly on last two instances. The AMO *product* and *binary* encodings are quite poor, while the others, AMO *commander* and *bimander* encodings are rather good. Like the golomb ruler problem, the runtime is rather related to the number of conflicts, whereas the number of conflicts is strongly related to the number of decisions.

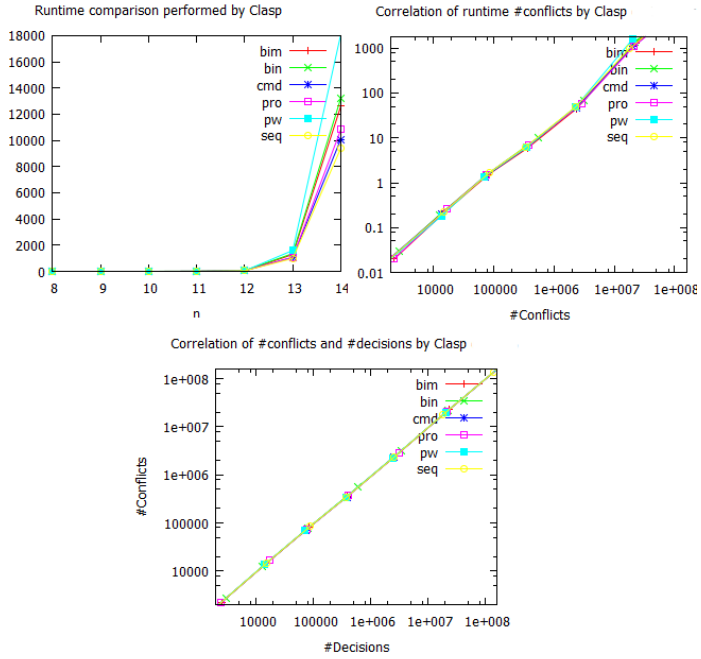

Figure 2: Clasp on Langford problem.

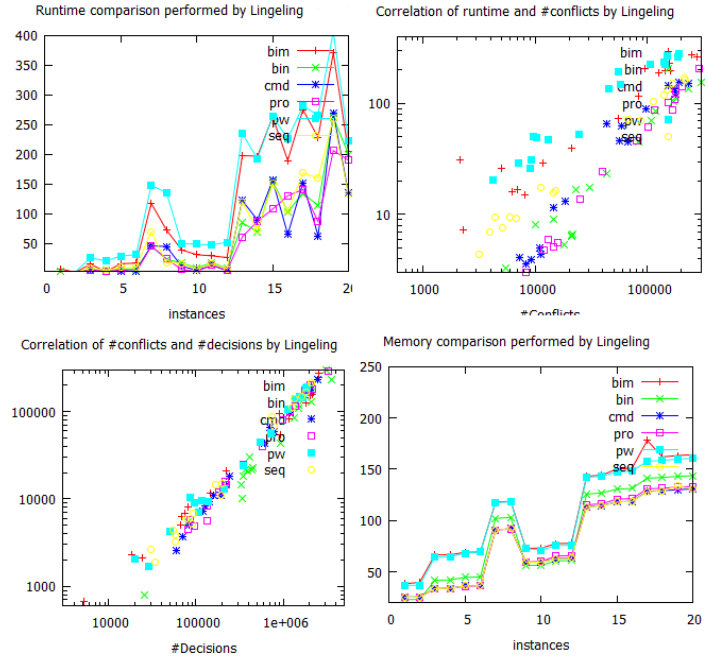

Figure 4: Lingeling on satisfiable Open Shop instances.

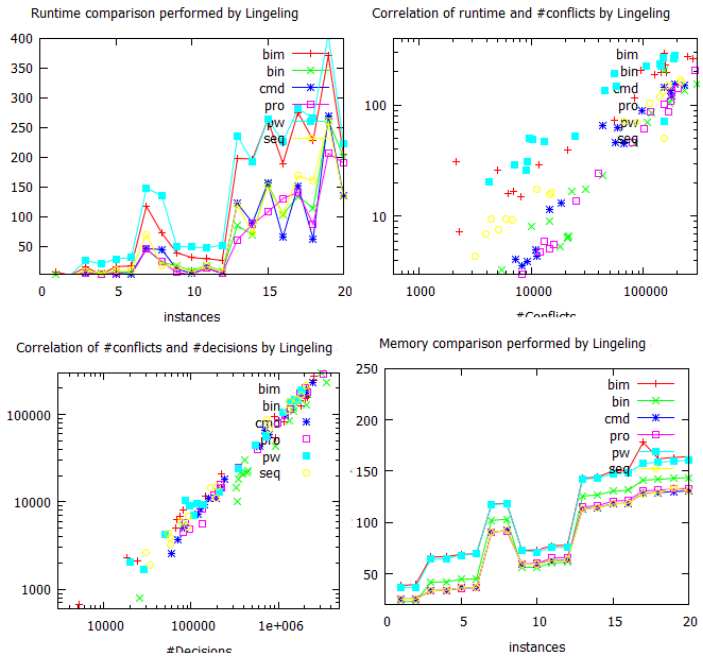

Figure 3: Riss on satisfiable Hidoku instances.

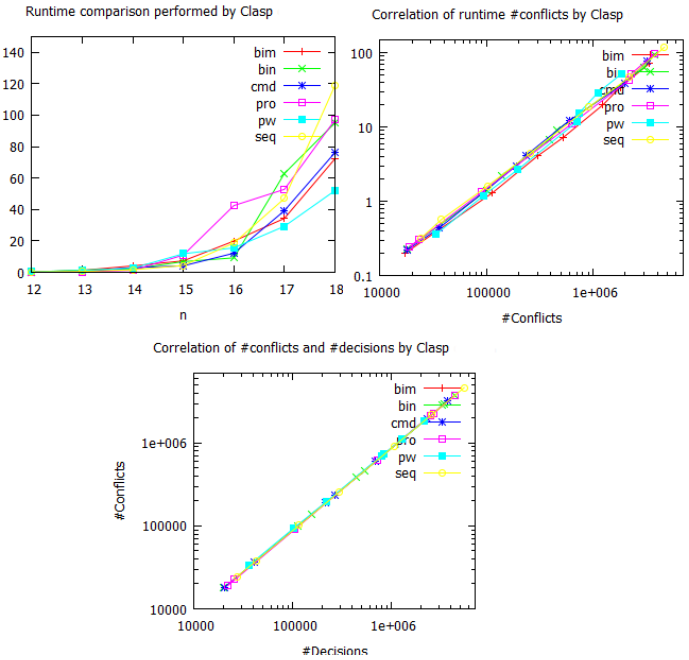

Figure 5: Clasp on all-interval series instances.

4 CONCLUSIONS

The paper has two main contributions. Firstly, a significant insight is the relationship between the auxiliary variables required by the AMO SAT-encoding and the variables used by its corresponding SAT encoding of finite CSP domains if it exists. As a result, one could use a channeling constraint for the redundant and hybrid encoding. For example, the auxiliary variables required by the AMO sequential counter encoding are exactly the variables used by its corresponding SAT encoding, the order encoding (see [38]). Consequently, these

auxiliary variables can be used for the challenging constraint, which is used by $Sp–Or_{red}$ and $Sp–Or_{hyb}$ (see [7]).

Secondly, regardless of three SAT solvers on various benchmarks, several important observations can be drawn through the empirical study on SAT encoding of the AMO constraint:

- In terms of runtime, the AMO encodings are significantly diverse. One AMO encoding may perform variously not only on different benchmark but also on the same benchmark (with different solvers) compared to the other encodings.
- In general the runtime is somewhat related to the number of conflicts, whereas the number of conflicts is usually and closely related to the number of decisions.
- Interestingly the runtime is rather related to the memory used, i.e., an AMO encoding which performs faster may consume less memory than a slower AMO encoding.

For future work, a possible research is to study how the number of disjoint subsets in some AMO encodings, consisting of the *product*, *bimander* and *commander* encodings, could affect these encodings in realistic problems. Based on the guidelines, the AMO encodings should be more thoroughly tested to obtain some certain guidelines. In particular, one may know which an AMO encoding should be used for a particular type of problems, consequently the encoding can achieve a high performance.

ACKNOWLEDGMENTS

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (NRF-2017R1A2 B4012559).

REFERENCES

[1] Anbulagan and Alban Grastien. 2009. Importance of Variables Semantic in CNF Encoding of Cardinality Constraints. In *SARA 2009, Lake Arrowhead, California, USA, 8-10 August 2009*, Vadim Bulitko and J. Christopher Beck (Eds.). AAAI.

[2] Carlos Ansótegui and Felip Manyà. 2004. Mapping problems with finite-domain variables into problems with boolean variables. In *SAT 2004, 10-13 May 2004, Vancouver, BC, Canada*. Springer LNCS, 1–15.

[3] Josep Argelich, Alba Cabiscol, Inês Lynce, and Felip Manyà. 2010. New Insights into Encodings from MaxCSP into Partial MaxSAT. In *40th IEEE International Symposium on Multiple-Valued Logic, ISMVL 2010, Barcelona, Spain, 26-28 May 2010*. IEEE Computer Society, 46–52.

[4] Roberto Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. 2011. Cardinality Networks: a Theoretical and Empirical Study. *Constraints* 16, 2 (2011), 195–221.

[5] Olivier Bailleux and Yacine Boufkhad. 2003. Efficient CNF Encoding of Boolean Cardinality Constraints. In *CP 2003 (LNCS)*, Francesca Rossi (Ed.), Vol. 2833. Springer, 108–122.

[6] Olivier Bailleux, Yacine Boufkhad, and Olivier Roussel. 2009. New Encodings of Pseudo-Boolean Constraints into CNF. In *SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings (Lecture Notes in Computer Science)*, Oliver Kullmann (Ed.), Vol. 5584. Springer, 181–194.

[7] Pedro Barahona, Steffen Hölldobler, and Van-Hau Nguyen. 2014. Representative Encodings to Translate Finite CSPs into SAT. In *11th CPAIOR 2014, Cork, Ireland, May 19-23, 2014*.

[8] Yael Ben-Haim, Alexander Ivrii, Oded Margalit, and Arie Matsliah. 2012. Perfect Hashing and CNF Encodings of Cardinality Constraints. In *SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings (LNCS)*, Alessandro Cimatti and Roberto Sebastiani (Eds.), Vol. 7317. Springer, 397–409.

[9] Armin Biere. 2013. Lingeling, Plingeling and Treengeling Entering the SAT Competition 2013. In *Proceedings of SAT Competition 2013*, Anton Belov Adrian Balint, Marjin Heule, and Matti Järvisalo (Eds.). 51–52.

[10] Jing-Chao Chen. 2010. A new SAT Encoding of the At-Most-One Constraint. In *Proc. of the Tenth Int. Workshop of Constraint Modelling and Reformulation*.

[11] Martin Davis, George Logemann, and Donald Loveland. 1962. A Machine Program for Theorem-proving. *Commun. ACM* 5, 7 (July 1962), 394–397.

[12] Johan de Kleer. 1989. A Comparison of ATMS and CSP Techniques. In *IJCAI*. 290–296.

[13] Niklas Eén and Niklas Sörensson. 2006. Translating Pseudo-Boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation* 2 (2006), 1–26.

[14] Alan M. Frisch and Paul A. Giannoros. 2010. 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*.

[15] Alan M. Frisch, Timothy J. Peugniez, Anthony J. Doggett, and Peter W. Nightingale. 2005. Solving Non-Boolean Satisfiability Problems with Stochastic Local Search: A Comparison of Encodings. *J. Autom. Reason.* 35 (October 2005), 143–179. Issue 1-3.

[16] Martin Gebser, Benjamin Kaufmann, and Torsten Schaub. 2009. The Conflict-Driven Answer Set Solver clasp: Progress Report. In *LPNMR*. 509–514.

[17] Ian P. Gent. 2002. Arc Consistency in SAT. In *Proceedings of the 15th Eureopean Conference on Artificial Intelligence, ECAI'2002*, Frank van Harmelen (Ed.). IOS Press, 121–125.

[18] Gael Glorian, Jean-Marie Lagniez, Valentin Montmirail, and Nicolas Szczepanski. 2019. An Incremental SAT-Based Approach to the Graph Colouring Problem. In *CP 2009*, Thomas Schiex and Simon de Givry (Eds.). Springer, 213–231.

[19] Brahim Hnich, Ian Miguel, Ian P. Gent, , and Toby Walsh. [n. d.]. CSPLib is a Library of Test Problems for Constraint Solvers. http://www.csplib.org/. ([n. d.]). [Online; accessed 24-April-2020].

[20] S. Hölldobler, N. Manthey, VH. Nguyen, and P. Steinke. 2012. Solving Hidokus Using SAT Solvers. In *Proc. INFOCOM-5*. 208–212. ISSN 2219-293X.

[21] Steffen Hölldobler, Norbert Manthey, Van Hau Nguyen, Julian Stecklina, and Peter Steinke. 2011. *Modern Parallel SAT-Solvers*. Technical Report. TU Dresden, Knowledge Representation and Reasoning.

[22] Steffen Hölldobler and Van-Hau Nguyen. 2013. *An Efficient Encoding of the At-Most-One Constraint*. Technical Report. Knowledge Representation and Reasoning Group 2013-04, Technische Universität Dresden, 01062 Dresden, Germany.

[23] Kazuo Iwama and Shuichi Miyazaki. 1994. SAT-Variable Complexity of Hard Combinatorial Problems. In *In Proceedings of the World Computer Congress of the IFIP, pages 253–258 (Volume 1). Elsevier Science B.V.* 253–258.

[24] Henry Kautz and Bart Selman. 2007. The State of SAT. *Discrete Appl. Math.* 155 (June 2007), 1514–1524.

[25] Will Klieber and Gihwon Kwon. 2007. Efficient CNF Encoding for Selecting 1 from N Objects . In *the Fourth Workshop on Constraint in Formal Verification(CFV)*.

[26] Norbert Manthey. 2013. The SAT Solver RISS3G at SC 2013 *(Department of Computer Science Series of Publications B)*, A. Balint, A. Belov, M. J.H. Heule, and M. Järvisalo (Eds.), Vol. B-2013-1. University of Helsinki, Helsinki, Finland, 72–73.

[27] Ruben Martins, Vasco M. Manquinho, and Inês Lynce. 2011. Exploiting Cardinality Encodings in Parallel Maximum Satisfiability. In *ICTAI 2011, Boca Raton, FL, USA, November 7-9, 2011*. 313–320.

[28] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. 2001. Chaff: Engineering an Efficient SAT Solver. In *DAC 2001, June 18-22, 2001*. ACM, 530–535.

[29] Saeed Nejati, Jan Horáček, Catherine Gebotys, and Vijay Ganesh. 2018. Algebraic Fault Attack on SHA Hash Functions Using Programmatic SAT Solvers. In *CP2018*, John Hooker (Ed.). Springer, Cham, 737–754.

[30] Van-Hau Nguyen. 2017. SAT Encodings of Finite-CSP Domains: A Survey. In *SoICT2017, Nha Trang City, Viet Nam, December 7-8, 2017*. ACM, 84–91.

[31] Van-Hau Nguyen and Son Thai Mai. 2014. Solving the all-interval series problem: SAT vs CP. In *SoICT '14, Hanoi, Vietnam, December 4-5, 2014*. ACM, 65–74.

[32] Van-Hau Nguyen and Son Thai Mai. 2015. A New Method to Encode the At-Most-One Constraint into SAT. In *SoICT2015, Hue City, Vietnam, December 3-4, 2015*. ACM, 46–53.

[33] Steve David Prestwich. 2003. SAT problems with chains of dependent variables. *Discrete Applied Mathematics* 130, 2 (2003), 329–350.

[34] Steve David Prestwich. 2004. Full Dynamic Substitutability by SAT Encoding. In *CP 2004 (LNCS)*, Mark Wallace (Ed.), Vol. 3258. Springer, 512–526.

[35] Steve David Prestwich. 2007. Finding Large Cliques using SAT Local Search, B. OSullivan F. Benhamou, N. Jussien (Ed.), Vol. Trends in Constraint Programming. ISTE, 273–278.

[36] João Marques Silva and Inês Lynce. 2007. Towards Robust CNF Encodings of Cardinality Constraints . In *CP 2007 (LNCS)*, Vol. 4741. Springer, 483–497.

[37] Carsten Sinz. 2005. Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In *CP 2005 (LNCS)*, Vol. 3709. Springer, 827–831.

[38] Naoyuki Tamura, Akiko Taga, Satoshi Kitagawa, and Mutsunori Banbara. 2009. Compiling Finite Linear CSP into SAT. *Constraints* 14, 2 (2009), 254–272.

[39] Toby Walsh. 2000. SAT v CSP. In *Principles and Practice of Constraint Programming - CP2000 (Lecture Notes in Computer Science)*, Vol. 1894. Springer, 441–456.

[40] Joost P. Warners. 1998. A Linear-Time Transformation of Linear Inequalities into Conjunctive Normal Form. *Inform. Process. Lett.* 68, 2 (1998), 63–69.