A New Method to Encode the At-Most-One Constraint into SAT

Van-Hau Nguyen
Faculty of Information Technology
Hung Yen University of Technology and Education, Vietnam
nvhau66@gmail.com

Son T. Mai
Faculty of Information Technology
University of Transport, HoChiMinh City, Vietnam
mtson@hcmutrans.edu.vn

ABSTRACT

One of the most widely used constraints 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 AMO *bimander* encoding which can be easily extended to encode cardinality constraints, which are often used in constraint programming. Experimental results reveal that the new encoding is very competitive compared with all other state-of-the-art encodings. Furthermore, we will prove that the new encoding allows unit propagation to achieve arc consistency - an important technique in constraint programming. We also show that a special case of the AMO *bimander* encoding outperforms the AMO *binary* encoding, a widely used encoding, in all our experiments.

CCS Concepts

•Theory of computation → Constraint and logic programming; •Computing methodologies → Knowledge representation and reasoning;

Keywords

Boolean satisfiability, SAT encoding, at-most-one constraint, constraint programming, constraint satisfaction problem, CSP.

1. INTRODUCTION

Solving propositional satisfiability (SAT) problems is 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 [33, 13]. 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.

An increasing number of real-world applications in computer science can be expressed as constraint satisfaction problems (CSPs) [6, 38]. While CSPs can be solved directly using appropriate solvers, the generality and success of SAT solvers in recent years has let to a fruitful competition between the CSP and the SAT community. To utilize state-of-the-art SAT solvers, CSPs need to be encoded as SAT instances (see [43, 2, 42, 41, 37, 35, 9, 8, 34]). Such encodings should not only be efficiently generated, but should also be efficiently solved by SAT solvers. Currently, mapping a CSP into a SAT instance is regarded more of an art than a science ([43, 21, 25, 37]), and detailed studies of different encodings are needed in order to better understand these mappings.

Generally, different SAT encodings of CSPs yield different formula sizes and different run time behaviour of the used SAT solver. There does not seem to be general knowledge why a particular encoding performs better than others. However, before using a SAT solver a SAT instance (CNF) is mainly influenced by some of the following features:

- the number of variables (and/or literals) required (search space)
- the number of clauses (overhead when propagating variable assignments)
- the length of clauses (e.g., unit and binary)
- the strength of unit propagation (local consistency, e.g., forward checking and maintaining arc-consistency) ([43, 21])
- the characteristics of the problems (e.g., the type of CSP constraints) [8]

Although many encodings have been proposed [43, 21, 2, 42, 41, 37], the most straightforward way of mapping a CSP into a SAT instance is the *spare* encoding (see [43, 9]). The *sparse* encoding requires the translation of global constraints like the at-least-one (ALO) and at-most-one (AMO) constraints requiring that a CSP variable has at least one and at most one value assigned to it, respectively. Whereas the ALO constraint can be easily encoded by a single clause, the encoding of the AMO constraint is more complicated and has been intensively studied ([26, 17, 14, 36]). This is due to the fact that many different applications such as computer motographs [7], partial Max-SAT [3, 4], or cardinality constraints [17] contain the AMO constraint. From now on, to avoid the confusion between a SAT encoding of a finite CSP domain and a SAT encoding of the AMO constraint, this paper will use the term AMO SAT-encoding for a SAT encoding of the AMO constraint.

SoICT 2015, December 03 - 04, 2015, Hue City, Viet Nam

ISBN 978-1-4503-3843-1/15/12...$15.00

DOI: http://dx.doi.org/10.1145/2833258.2833293

In the *sparse* encoding (see [9]), 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*. Herein, this will be denoted by $\leq_1 (X_1, ..., X_n)$, where X_i, $1 \leq i \leq n$, are propositional variables.

Inspired by many interesting and recent results [26, 17, 14], especially when Prestwich used the AMO *binary* encoding [18, 19] to successfully solve many large instances with a standard SAT solver [36], we will survey several widely used encodings of the AMO constraint. Then, we will introduce a new encoding, the AMO *bimander* encoding. The new encoding requires $\lceil log_2 m \rceil (1 \leq m \leq n)$[1] auxiliary variables and $\frac{n^2}{2m} + n\lceil log_2 m \rceil - \frac{n}{2}$ binary clauses, where m is the number of disjoint subsets used by dividing the given set $\{X_1, ..., X_n\}$ of propositional variables.

Additionally, the AMO *bimander* encoding can be easily extended to cardinality constraints, denoted by $\leq_k (X_1, ..., X_n)$, which expresses that less than k of n propositional variables $X_i, 1 \leq i \leq n$ can be simultaneously assigned to *TRUE*. To the best of our knowledge, our encoding is the one that requires least number of auxiliary variables among known encodings except for the AMO *pairwise* encoding, which needs no auxiliary variables at all. With respect to scalability, the AMO *bimander* encoding can be adjusted by changing the parameter m. For example, by setting the parameter m to specific values, the AMO *binary* and *pairwise* encodings can be expressed as special cases of the AMO *bimander* encoding. Interestingly, the special case of the AMO *bimander* encoding where $m = \lceil \frac{n}{2} \rceil$, outperforms the AMO *binary* encoding in all our experiments. It is important to note that our encoding allows unit propagation (UP) to preserve arc consistency, one of the most important techniques in Constraint Programming (see [11]).

The structure of the paper is as follows. In Section 2, we briefly represent many known encodings of the AMO constraint. In Section 3, we describe the new *bimander* encoding and prove several important properties. In Section 4, we compare the AMO *bimander* encoding with other encodings through experiments. Finally, we conclude and outline future research in Section 5.

2. EXISTING ENCODINGS

Before giving a brief survey of AMO SAT-encodings, this section first defines several important notions and notations, mainly following Frisch and Giannoros [17].

Definition 1 (Correctness). *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 $AMO(x_1, \ldots, x_n)$. The encoding $\phi(X, A)$ is* correct *if and only if:*

- *any partial interpretation $\hat{x}$ that satisfies $AMO(x_1, \ldots, x_n)$ can be extended to a complete interpretation that satisfies $\phi(X, A)$, and*
- *for any partial interpretation $\hat{x}$ 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.*

[1] $\lceil x \rceil$ is the smallest integer not less than x.

It is well-known that one usually considers whether UP in SAT solvers achieve pruning in a similar way to CP solvers applying local consistency to the original CSP (e.g., arc consistency or forward checking).

Definition 2. *Unit propagation (UP) of a SAT encoding of the constraint $AMO(x_1, \ldots, x_n)$ achieves the same pruning as arc consistency on the original CSP, which is referred to as the UPaAC property from here on, if two following conditions hold [17]:*

- *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* 0 *by using UP.*

In the following sections, generally $AMO(X)$ and $ALO(X)$ denote the at-most-one and at-least-one clauses for the set of propositional variables $X = \{x_1, \ldots, x_n\}$, respectively, and we define $EO(X) := AMO(X) \wedge ALO(X)$, namely exactly-one clauses, for the set of propositional variables X. Our goal is to encode the constraint $AMO(X)$ into CNF. For the sake of convenience, a running example illustrates these encodings through the set consisting of eight Boolean variables, $X = \{x_1, \ldots, x_8\}$.

2.1 The AMO Pairwise Encoding

This encoding has several different names: the *naive* encoding [40, 26], the *pairwise* encoding [39, 36], or the *binomial* encoding [17]. This paper refers to it as the AMO *pairwise* encoding. The idea of this encoding is to express that all possible combinations of two variables are not simultaneously assigned to *TRUE*. Therefore as soon as one literal is assigned to *TRUE*, the all others must be assigned to *0*:

$$\bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} \neg(x_i \wedge x_j) \equiv \bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} (\neg x_i \vee \neg x_j).$$

Example 1. *In the running example, the AMO pairwise encoding produces the following clauses:*

$$\begin{array}{r}
(\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_4) \wedge \cdots \wedge (\neg x_1 \vee \neg x_8) \\
(\neg x_2 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_4) \wedge \cdots \wedge (\neg x_2 \vee \neg x_8) \\
(\neg x_3 \vee \neg x_4) \wedge \cdots \wedge (\neg x_3 \vee \neg x_8) \\
\vdots \\
(\neg x_7 \vee \neg x_8)
\end{array}$$

The AMO *pairwise* encoding is a traditional way of encoding the AMO constraint into SAT. Although this encoding does not need any auxiliary variables, it requires a quadratic number of clauses. Consequently, this method may result in large formulas on problems with large domains. Nevertheless, the AMO *pairwise* encoding is not only widely used in practice, but also able to combine with other encodings [26, 42, 14]. It is important to stress that the AMO *pairwise* encoding has the UPaAC property (see Table 1 in Section 4.1).

2.2 The AMO Binary Encoding

Frisch et al. [18, 19] proposed the AMO *binary* encoding. Independently, Prestwich introduced it as the *bitwise* encoding [36, 37]) and used it to successfully solve a number of large instances of CSPs with a standard SAT solver [36]. This paper refers to it as the AMO *binary* encoding.

The AMO *binary encoding* requires a set of auxiliary Boolean variables $\{b_1, \ldots, b_{\lceil log_2 n \rceil}\}$ with a set of clauses:

$$\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{\lceil log_2 n \rceil} x_i \rightarrow \phi(i,j) \equiv \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{\lceil log_2 n \rceil} \neg x_i \vee \phi(i,j),$$

where $\phi(i,j)$ denotes b_j (or $\neg b_j$) if the bit j of $i-1$ represented by a *binary string* is 1 (or 0).

The idea is to create the different sequences of $\lceil log_2 n \rceil$-tuples $b_j, 1 \leq j \leq \lceil log_2 n \rceil$, such that whenever any x_i is assigned to $TRUE$ for all i, then one immediately infers that the other variables $x_{i'}$ must be assigned to 0, for any $1 \leq i' \neq i \leq n$.

Example 2. *The running example is represented by the AMO binary encoding as follows:*

$$\begin{array}{l} x_1 \rightarrow \neg b_1 \wedge x_2 \rightarrow b_1 \wedge x_3 \rightarrow \neg b_1 \wedge \ldots \wedge x_8 \rightarrow b_1 \wedge \\ x_1 \rightarrow \neg b_2 \wedge x_2 \rightarrow \neg b_2 \wedge x_3 \rightarrow b_2 \wedge \ldots \wedge x_8 \rightarrow b_2 \wedge \\ x_1 \rightarrow \neg b_3 \wedge x_2 \rightarrow \neg b_3 \wedge x_3 \rightarrow \neg b_3 \wedge \ldots \wedge x_8 \rightarrow b_3 \end{array}$$

which is semantically equivalent to

$$\begin{array}{l} (\neg x_1 \vee \neg b_1) \wedge (\neg x_2 \vee b_1) \wedge (\neg x_3 \vee \neg b_1) \wedge \ldots \wedge (\neg x_8 \vee b_1) \wedge \\ (\neg x_1 \vee \neg b_2) \wedge (\neg x_2 \vee \neg b_2) \wedge (\neg x_3 \vee b_2) \wedge \ldots \wedge (\neg x_8 \vee b_2) \wedge \\ (\neg x_1 \vee \neg b_3) \wedge (\neg x_2 \vee \neg b_3) \wedge (\neg x_3 \vee \neg b_3) \wedge \ldots \wedge (\neg x_8 \vee b_3) \end{array}$$

2.3 The AMO Commander Encoding

Klieber and Kwon [26] described the AMO *commander* encoding by dividing the set of propositional variables $X = \{x_1, \ldots, x_n\}$ into m (between 1 and n) disjoint subsets denoted by $\{G_1, \ldots, G_m\}$, and introducing a *commander* variable c_i for each subset G_i, $1 \leq i \leq m$. The AMO *commander encoding* is defined as follows.

1. Exactly one variable in each set $G_i \cup \{\neg c_i\}$ is assigned to $TRUE$:

$$\bigwedge_{i=1}^{m} EO(\{\neg c_i\} \cup G_i) = \bigwedge_{i=1}^{m} AMO(\{\neg c_i\} \cup G_i) \wedge \bigwedge_{i=1}^{m} ALO(\{\neg c_i\} \cup G_i),$$

   whereas the ALO constraint is easily translated into a single clause, AMO can be encoded either by the AMO pairwise or *commander* encoding.

2. At most one commander variable is assigned to $TRUE$. This constraint can be encoded either by the AMO *pairwise* encoding or by another encoding (even by a recursive application of the AMO *commander* encoding):

$$\bigwedge_{i=1}^{m} AMO(c_i).$$

Example 3. *In the running example, by selecting $m = 4$, dividing the set $X = \{x_1, \ldots, x_8\}$ into the disjoint subsets $G_1 = \{x_1, x_2\}$, $G_2 = \{x_3, x_4\}$, $G_3 = \{x_5, x_6\}$, and $G_4 = \{x_7, x_8\}$, and adding four* commander variables c_1, c_2, c_3, *and* c_4 *we obtains:*

$$\begin{array}{lll} AMO(\neg c_1, x_1, x_2) & \wedge & (\neg c_1 \vee x_1 \vee x_2) \quad \wedge \\ AMO(\neg c_2, x_3, x_4) & \wedge & (\neg c_2 \vee x_3 \vee x_4) \quad \wedge \\ AMO(\neg c_3, x_5, x_6) & \wedge & (\neg c_3 \vee x_5 \vee x_6) \quad \wedge \\ AMO(\neg c_4, x_7, x_8) & \wedge & (\neg c_4 \vee x_7 \vee x_8). \end{array}$$

By using the AMO pairwise encoding, the above formula is further encoded:

$$\begin{array}{l} (c_1 \vee \neg x_1) \wedge (c_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_2) \wedge (\neg c_1 \vee x_1 \vee x_2) \wedge \\ (c_2 \vee \neg x_3) \wedge (c_2 \vee \neg x_4) \wedge (\neg x_3 \vee \neg x_4) \wedge (\neg c_2 \vee x_3 \vee x_4) \wedge \\ (c_3 \vee \neg x_5) \wedge (c_3 \vee \neg x_6) \wedge (\neg x_5 \vee \neg x_6) \wedge (\neg c_3 \vee x_5 \vee x_6) \wedge \\ (c_4 \vee \neg x_7) \wedge (c_4 \vee \neg x_8) \wedge (\neg x_7 \vee \neg x_8) \wedge (\neg c_4 \vee x_7 \vee x_8). \end{array}$$

At most one among the commander variables is assigned to TRUE*:*

$$\begin{array}{ll} AMO(c_1, c_2, c_3, c_4) \equiv & (\neg c_1 \vee \neg c_2) \wedge (\neg c_1 \vee \neg c_3) \wedge \\ & (\neg c_1 \vee \neg c_4) \wedge (\neg c_2 \vee \neg c_3) \wedge \\ & (\neg c_2 \vee \neg c_4) \wedge (\neg c_3 \vee \neg c_4). \end{array}$$

Compared with the AMO *pairwise* encoding, the AMO *commander* encoding requires a fewer number of clauses but introduces auxiliary variables. The AMO *commander* encoding also has the UPaAC property (see Table 1 in Section 4.1).

2.4 The AMO Product Encoding

Chen [14] proposed an AMO encoding, named the AMO *product* encoding. Instead of encoding the AMO constraint $AMO(x_1, \ldots, x_n)$, the author encoded a constraint consisting of n corresponding points, denoted by $\leq_1 \{(u_i, v_j) \mid 1 \leq i \leq p, 1 \leq j \leq q, p \times q \geq n\}$.

The idea can be explained as follows:

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 AMO *product* encoding is obtained as:

$$AMO(X) = AMO(U) \wedge AMO(V) \bigwedge_{1 \leq i \leq p, 1 \leq j \leq q}^{1 \leq k \leq n, k=(i-1)q+j} ((\neg x_k \vee u_i) \wedge (\neg x_k \vee v_j)),$$

where $AMO(U)$ and $AMO(V)$ can be encoded by either another encoding or a recursive application of the AMO *product* encoding.

Example 4. *With regard to the running example, by choosing $p = 3$, $q = 3$, and using the AMO pairwise encoding for $AMO(U)$ and $AMO(V)$, the derived clauses are:*

$$\begin{array}{l} AMO(U) = (\neg u_1 \vee \neg u_2) \wedge (\neg u_1 \vee \neg u_3) \wedge (\neg u_2 \vee \neg u_3) \\ AMO(V) = (\neg v_1 \vee \neg v_2) \wedge (\neg v_1 \vee \neg v_3) \wedge (\neg v_2 \vee \neg v_3) \\ AMO(X) = AMO(U) \wedge AMO(V) \wedge \\ \quad (\neg x_1 \vee u_1) \wedge (\neg x_1 \vee v_1) \wedge (\neg x_2 \vee u_2) \wedge (\neg x_2 \vee v_1) \wedge \\ \quad (\neg x_3 \vee u_3) \wedge (\neg x_3 \vee v_1) \wedge (\neg x_4 \vee u_1) \wedge (\neg x_4 \vee v_2) \wedge \\ \quad (\neg x_5 \vee u_2) \wedge (\neg x_5 \vee v_2) \wedge (\neg x_6 \vee u_3) \wedge (\neg x_6 \vee v_2) \wedge \\ \quad (\neg x_7 \vee u_1) \wedge (\neg x_7 \vee v_3) \wedge (\neg x_8 \vee u_2) \wedge (\neg x_8 \vee v_3) \end{array}$$

2.5 The AMO Sequential Counter Encoding

By building a count-and-compare hardware circuit and translating this circuit to an equivalent CNF formula, Sinz [40] introduced cardinality constraints $\leq_k (x_1, \ldots, x_n)$. Here, we only consider the case $k = 1$ and obtain the AMO *sequential counter encoding* [40, 39]:

$$(\neg x_1 \vee s_1) \wedge (\neg x_n \vee \neg s_{n-1}) \bigwedge_{1<i<n} ((\neg x_i \vee s_i) \wedge (\neg s_{i-1} \vee s_i) \wedge (\neg x_i \vee \neg s_{i-1})),$$

where $s_i, 1 \leq i \leq n-1$, are auxiliary variables. The above formula is a constraint which guarantees that whenever any $x_i, 1 \leq i \leq n$, is assigned to *TRUE*, then the other variables $x_{i'}$ must be assigned to *0*, for any $1 \leq i' \neq i \leq n$.

EXAMPLE 5. *By introducing seven auxiliary variables for the AMO sequential counter encoding, we obtain the following formula:*

$$
\begin{array}{llllll}
\neg x_1 \vee s_1 & \wedge & & & & \\
\neg x_2 \vee s_2 & \wedge & \neg s_1 \vee s_2 & \wedge & \neg x_2 \vee \neg s_1 & \wedge \\
\neg x_3 \vee s_3 & \wedge & \neg s_2 \vee s_3 & \wedge & \neg x_3 \vee \neg s_2 & \wedge \\
\neg x_4 \vee s_3 & \wedge & \neg s_3 \vee s_4 & \wedge & \neg x_4 \vee \neg s_3 & \wedge \\
\neg x_5 \vee s_4 & \wedge & \neg s_4 \vee s_5 & \wedge & \neg x_5 \vee \neg s_4 & \wedge \\
\neg x_6 \vee s_5 & \wedge & \neg s_5 \vee s_6 & \wedge & \neg x_6 \vee \neg s_5 & \wedge \\
\neg x_7 \vee s_6 & \wedge & \neg s_6 \vee s_7 & \wedge & \neg x_7 \vee \neg s_6 & \wedge \\
\neg x_8 \vee \neg s_7. & & & & &
\end{array}
$$

3. THE AMO BIMANDER ENCODING

This paper proposes a new AMO SAT-encoding, the so-called AMO *bimander* encoding. The general idea of the new encoding is based on both the ideas of the AMO *binary* encoding and the AMO *commander* encoding.

We partition a set of propositional variables $X = \{x_1, \ldots, x_n\}$ into m (between 1 and n) disjoint subsets $\{G_1, \ldots, G_m\}$ such that each subset G_i consists of $g = \lceil \frac{n}{m} \rceil$ variables. However, instead of introducing *commander variables* like in the AMO *commander* encoding, the AMO *bimander* encoding introduce a set of auxiliary propositional variables $b_1, \ldots, b_{\lceil log_2 m \rceil}$ as in the AMO *binary* encoding. The variables $b_1, \ldots, b_{\lceil log_2 m \rceil}$ play the role of the commander variables in the AMO *commander* encoding.

The AMO *bimander encoding* is the conjunction of the following clauses:

1. At most *one* variable in each subset can be *1*. One must encode this constraint for each subset $G_i, 1 \leq i \leq m$, by using the AMO *pairwise* encoding:
$$\bigwedge_{i=1}^{m} AMO(G_i). \tag{1}$$

2. The following clauses are generated by the constraints between each variable and commander variables in a subset:
$$\bigwedge_{i=1}^{m} \bigwedge_{h=1}^{g} \bigwedge_{j=1}^{\lceil log_2 m \rceil} x_{i,h} \rightarrow \phi(i,j) \equiv \bigwedge_{i=1}^{m} \bigwedge_{h=1}^{g} \bigwedge_{j=1}^{\lceil log_2 m \rceil} \neg x_{i,h} \vee \phi(i,j), \tag{2}$$
where $\phi(i,j)$ denotes b_j (or $\neg b_j$) if the bit j of $i-1$ represented by a unique binary string is 1 (or 0).

EXAMPLE 6. *In the running example by choosing* $m = \lceil \sqrt{n} \rceil = 3$ *we obtain* $G_1 = \{x_1, x_2, x_3\}$, $G_2 = \{x_4, x_5, x_6\}$, *and* $G_3 = \{x_7, x_8\}$. *Consequently, Formula 1 generates the following set of clauses:*

$$AMO(x_1, x_2, x_3) \wedge AMO(x_4, x_5, x_6) \wedge AMO(x_7, x_8).$$

In the second step, we introduce a set of auxiliary variables $\{b_1, \ldots, b_{\lceil log_2 m \rceil}\} = \{b_1, b_2\}$. *Then, the following set of clauses is generated:*

$$
\begin{array}{llllll}
\neg x_1 \vee \neg b_1 & \wedge & \neg x_4 \vee b_1 & \wedge & \neg x_7 \vee \neg b_1 & \wedge \\
\neg x_1 \vee \neg b_2 & \wedge & \neg x_4 \vee \neg b_2 & \wedge & \neg x_7 \vee b_2 & \wedge \\
\neg x_2 \vee \neg b_1 & \wedge & \neg x_5 \vee b_1 & \wedge & \neg x_8 \vee \neg b_1 & \wedge \\
\neg x_2 \vee \neg b_2 & \wedge & \neg x_5 \vee \neg b_2 & \wedge & \neg x_8 \vee b_2 & \\
\neg x_3 \vee \neg b_1 & \wedge & \neg x_6 \vee b_1 & \wedge & & \\
\neg x_3 \vee \neg b_2 & \wedge & \neg x_6 \vee \neg b_2 & \wedge & &
\end{array}
$$

Compared with the AMO *commander* encoding, the AMO *bimander* encoding does not require any constraint among the sequences of auxiliary variables because any combination of such variables $b_1, \ldots, b_{\lceil log_2 m \rceil}$ of a corresponding subset is different from any combinations of all the other groups. Let us prove some important properties of the AMO *bimander* encoding.

THEOREM 1 (CORRECTNESS). *The AMO bimander encoding is correct.*

PROOF. Assume that we have a partial interpretation $\hat{x} = (x_1, \ldots, x_l), 1 \leq l \leq n$, with *at most one* variable assigned to *TRUE*. In case all variables are assigned to *0*), then condition (1) is trivially satisfied. The same holds for condition (2). In case that only one variable, say $x_i, 1 \leq i \leq n$, is assigned to *TRUE*, then there is a corresponding sequence of *1* values assigned to the corresponding sequence of $\{b_1, \ldots, b_{\lceil log_2 m \rceil}\}$. Hence, condition (2) is satisfied as well. Therefore, the partial interpretation $\hat{x}$ can possibly be extended to a complete interpretation that satisfies two conditions.

Now suppose that we have a partial interpretation $\hat{x} = (x_1, \ldots, x_l), 1 \leq l \leq n$, with more than one variable assigned to *TRUE*. Assume that $x_i = 1$ and $x_j = 1$, for $1 \leq i \neq j \leq l$. In order to satisfy the condition (1), the variables x_i and x_j must belong to different subsets. That leads to two differently corresponding patterns of the sequence $\{b_1, \ldots, b_{\lceil log_2 m \rceil}\}$ which are assigned to *TRUE*. As a result, the sequence contains one propositional variable $b_k, 1 \leq k \leq \lceil log_2 m \rceil$ that is assigned to both *1* and *0* at the same time. In other words, there exists a clause, which is of the form $b_k \wedge \neg b_k$. Hence, if any partial interpretation has more than one variable assigned to *TRUE*, then UP produces an empty clause. It means that this partial interpretation can not be extended to a complete interpretation.

Follow Definition 1 we conclude that the AMO *bimander* encoding correctly encodes the AMO constraint into SAT. □

THEOREM 2 (STRENGTH). *The AMO bimander has the UPaAC property.*

PROOF. Suppose that we have a partial interpretation $\hat{x} = (x_1, \ldots, x_l)$, $1 \leq l \leq n$, where *1* is assigned to exactly one variable. Now we will show that UP will assign all other variables to *0*. Assume that variable $x_{i,j} = 1$, which is the j^{th} variable in the subset $G_i, 1 \leq i \leq m$, then this interpretation forces a corresponding pattern of the sequence $\{b_1, \ldots, b_{\lceil log_2 m \rceil}\}$ to *1*. Because $x_{i,j} = 1$, all other variables in the subset G_i are set to *0*, followed by condition (1). Due to condition (2), all the other variables in the subsets $G_{i'}, 1 \leq i' \neq i \leq m$ are set to *0* because they have different patterns of the sequence $\{b_1, \ldots, b_{\lceil log_2 m \rceil}\}$ corresponding to $x_{i,j} = 1$. Follow Definition 2 we conclude that UP on the AMO *bimander* encoding achieves arc consistency. □

Complexity.

We partition a set of propositional variables $X = \{x_1, \ldots, x_n\}$ into m $(1 \leq m \leq n)$ disjoint subsets $\{G_1, \ldots, G_m\}$ of size $g = \lceil \frac{n}{m} \rceil$ variables. As we supposed, we need a set of $\lceil log_2 m \rceil$ auxiliary variables. Condition (1) uses the AMO *pairwise* encoding for m groups, and each group consists of g variables. Consequently, we have $m * [\frac{g(g-1)}{2}] = \frac{n(\frac{n}{m}-1)}{2}$ new clauses. Condition (2) requires $m * [g * log_2 m] = n * \lceil log_2 m \rceil$ clauses. Hence, the encoding uses $\frac{n(\frac{n}{m}-1)}{2} + n\lceil log_2 m \rceil = \frac{n^2}{2m} + n\lceil log_2 m \rceil - \frac{n}{2}$ clauses.

Generalization.

It is worth pointing out that the AMO *bimander* encoding can be easily generalized to encode the at-most-k constraint. Again, the set of variables is partitioned into several subsets.

1. For each subset, the at-most-k constraint is encoded by a modified pairwise (or another) encoding.

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

$$\bigwedge_{i=1}^{m} \bigwedge_{h=1}^{g} \bigvee_{l=1}^{k} \bigwedge_{j=1}^{\lceil log_2 m \rceil} \neg x_{i,h} \vee \phi(i,h,l,j),$$

   where $\phi(i,h,l,j)$ denotes $b_{l,j}$ (or $\neg b_{l,j}$) if the bit j of $i-1$ represented by a binary string is 1 (or 0).

Special Cases.

One should observe that the AMO *bimander* encoding is a general case of several encodings. For example,

- The AMO *pairwise* encoding is a special case of the AMO *bimander* encoding by setting $m = 1$.

- The AMO *commander* encoding is a special case of the AMO *bimander* encoding by setting $m = 2$ (when both encodings divide into *2* subsets).

- The AMO *binary* encoding is a special case of the AMO *bimander* encoding by setting $m = n$.

4. COMPARISON AND EXPERIMENTAL EVALUATION

4.1 Comparison

In this section, we first summarize key features of SAT encodings of the AMO constraint. Thereafter, we experimentally evaluate the encodings presented in Section 2 using different domains. 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. The disjointed subsets by dividing the set of propositional variables $\{x_1, \ldots, x_n\}$ in the AMO *bimander* encoding is denoted by m. In addition to the encodings of the AMO constraint presented in previous parts, we also mention other encodings that are mainly used for cardinality constraints, the at-most-k constraints $\leq_k (x_1, \ldots, x_n)$. In this paper, we only consider these for the case $k = 1$.

As we can see in Table 1, the AMO *bimander* encoding requires the least auxiliary variables – with the exception of the AMO *pairwise* encoding – among known encodings. The *totalizer* encoding proposed by Bailleux al et. [7] requires clauses of size at most *3*, and the AMO *commander* encoding proposed by Klieber and Kwon [26] needs m (number of disjointed subsets) clauses of size $\lceil \frac{n}{m} + 1 \rceil$, whereas the AMO *product*, *sequential counter*, *binary* and *bimander* encoding require only binary clauses. Note that binary clauses may speed up SAT solvers significantly compared to longer clauses.

4.2 Experimental Evaluation

For the experimental evaluation we have selected some well-known, difficult problems which have been used in recent CSP and SAT competitions. In case of the AMO *bimander* encoding, we have considered two different values for the parameter m, viz. $m = \sqrt{n}$ and $m = \frac{n}{2}$.

Our experiments were conducted on different problems using CLASP 2 [20] ((*clasp2.1.3x86_64linux* version) with default configuration on a 2.66-GHz Intel Core 2 Quad processor with 3.8 GB of memory. Note that we also used two other state-of-the-art Conflict-Driven Clause Learning SAT solvers [2], *Riss3G* [32] (SAT competition 2013 version) and *Lingeling* [12] (*aqw* version), and the results obtained are slighly difference compared with CLASP 2.

In the following section, 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 AMO *commander* encoding, 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.

Pigeon-Hole Problems.

The goal of the problem is to prove that p pigeons can not fit in $h = p - 1$ holes. Table 2 shows the results from different encodings on unsatisfiable Pigeon-Holes instances. It can be seen that the AMO *bimander* encoding (with $m = \frac{n}{2}$) performs best in all cases, followed by the AMO *binary* encoding, whereas the AMO *bimander* encoding (with $m = \sqrt{n}$) outperforms the rest.

All-Interval Series Problems.

The goal of the problem is to arrange a permutation of the n integers ranging from 1 to n in such a way that the differences between adjacent numbers are also a permutation, of the numbers from 1 to $n - 1$. As a result, the performance of this benchmarks is heavily influenced by the performance of encoding the AMO constraint. In fact, AIS is one of classical CSPs and usually regarded as a difficult benchmark to find all solutions (see prob007 in [22]).

Table 3 summaries the running times for different encodings on AIS instances. Excepting for the cases $n = 7$ and $n = 8$, the table shows that the AMO *bimander* encoding in case $m = \frac{n}{2}$ significantly surpasses all the others. Moreover, for three last instances this one performs in a reasonable

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

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

enc	*clauses*	*aux vars*	*UPaAC*	*origin*
pairwise	$\binom{n}{2}$	0	yes	folklore
linear (CAR.)	$8n$	$2n$	no	[44]
totalizer	$O(n^2)$	$O(nlog(n))$	yes	[7]
binary	$nlog_2n$	$\lceil log_2n \rceil$	yes	[19]
sequential counter	$3n-4$	$n-1$	yes	[40]
sorting networks (CAR.)	$O(nlog_2^2n)$	$O(nlog_2^2n)$	yes	[15]
commander	$\sim 3n$	$\sim \frac{n}{2}$	yes	[26]
product	$2n+4\sqrt{n}+O(\sqrt[4]{n})$	$2\sqrt{n}+O(\sqrt[4]{n})$	yes	[14]
card. networks(CAR.)	$6n-9$	$4n-6$	yes	[5]
PHFs-based (CAR.)	$nlog_2n$	$\lceil log_2n \rceil$	yes	[10]
bimander	$\frac{n^2}{2m}+nlog_2m-\frac{n}{2}$	$log_2m, 1 \le m \le n$	yes	[24]
bimander ($m=\frac{n}{2}$)	$nlog_2n-\frac{n}{2}$	$\lceil log_2n \rceil - 1$	yes	[24]

Table 2: A comparison of the running 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**

time, whereas the AMO *pairwise*, *sequential*, and *product* encodings carry out more than 3600 seconds. The AMO *binary* encoding gives rather good results, while the AMO *bimander* encoding in case $m = \sqrt{n}$ and the AMO *commander* encoding perform similarly. The AMO *pairwise*, *sequential*, and *product* encodings perform worse.

Quasigroup With Holes Problems.

A quasigroup is a square of values x_{ij}, $1 \le i, j \le n$ where each number $[1..n]$ occurs exactly oncee in each row and column. Achlioptas et al. [1] introduced an encoding for generating satisfiable quasigroups with holes instances in which some cells are filled. Quasigroup with holes instances can be consider as a multiple permutation problem in which the variables may occur in more than one permutation problem. Moreover, the encoding can tune the generator to output hard instances. We experimented with instances with different levels of hardness.

Table 4 shows the results from different encodings on satisfiable quasigroup with holes problems. The AMO *bimander* encoding with parameter $m = \sqrt{n}$ is clearly the fastest with the exception of the instance qwh.order40.holes544. Surprisingly, the *pairwise* encoding performs very well followed by the *commander* encoding. The AMO *bimander* encoding with parameter $m = \frac{n}{2}$, the *binary*, and the *product* encoding are quite similar. Although the *sequential* encoding was the fastest on the instance qwh.order40.holes544, its overall performance is poor.

5. CONCLUSIONS AND FUTURE WORK

Inspired by being remarkably successful at solving hard and practical problems of SAT solving, many problems that were solved previously by other encodings can now be solved more effectively by translating them into SAT instances and applying advanced SAT solvers to find solutions. During the encoding phase, one of the most important constraints occurring naturally in a wide range of real world applications, is the at-most-one (AMO) constraint. Hence, many problems may benefit from effective encodings of this constraint.

The paper has three main contributions. Firstly, we proposed a new encoding for AMO, the so-called AMO *bimander* encoding. Compared to many other well-known AMO encodings, the AMO *bimander* encoding requires the least auxiliary variables (with the exception of the AMO *pairwise* encoding which does not erequire any auxiliary variables at all). Although the AMO *commander* encoding and the AMO *bimander* encoding use the same approach by dividing the original set of propositional variables, the AMO *commander* encoding requires clauses of size $\lceil \frac{n}{m}+1 \rceil$ (where m is the number of disjoint subsets), whereas the AMO *bimander* encoding requires only binary clauses. We believe that this helps the AMO *bimander* encoding to perform better than the AMO *commander* encoding in our experimental evaluation. Moreover, the AMO *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 AMO *pairwise* and AMO *binary* encodings are special cases of the AMO *bimander* encoding.

Secondly, this paper also proposes a special case, when dividing the propositional variables into $m = \lceil \frac{n}{2} \rceil$ disjoint subsets. From a theoretical point of view, this case is better than the AMO *binary* encoding due to fewer auxiliary variables and clauses. From a practical point of view, we show that this special case of the AMO *bimander* encoding ($m = \lceil \frac{n}{2} \rceil$) performs better than the AMO *binary* encoding in all experiments in term of running time.

Thirdly, in practice, the AMO *bimander* encoding is practical and easy to implement. Our results reveal that two particular cases of the AMO *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 AMO *bimander* encoding in real-

Table 3: **A comparison of running times for all interval series problems. 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 4: **A comparison of running times for satisfiable quasigroup with holes problems. 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
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

istic problems. It would be particularly useful to extend our findings to the at-most-k constraint. Finally, the ultimate goal should carry out a profound study of not only analytical, but also theoretical knowledge of variants of well-known encodings. We expect that this will help us to deepen our understanding on what encoding one should be selected given a particular problem. Currently, we are aiming at using the AMO *bimander* encoding for for solving the constrained clustering problems, which has recently become an emerging research topic. However, in contrast to existing works e.g. [23] which specifically designed for hierarchical clustering techniques, we focus on density-based clustering algorithms [16] due to its ubiquitousness, e.g. [28, 29, 31, 30, 27].

6. ACKNOWLEDGMENTS

We would like to thank Steffen Hölldobler and Pedro Barahona for many their fruitful suggestions.

7. REFERENCES

[1] D. Achlioptas, C. P. Gomes, H. A. Kautz, and B. Selman. Generating Satisfiable Problem Instances. In H. A. Kautz and B. W. Porter, editors, *AAAI/IAAI*, pages 256–261. AAAI Press / The MIT Press, 2000.

[2] C. Ansótegui and F. Manyà. Mapping problems with finite-domain variables to problems with boolean variables. In H. H. Hoos and D. G. Mitchell, editors, *SAT (Selected Papers*, volume 3542 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2004.

[3] J. Argelich, A. Cabiscol, I. Lynce, and F. Manyà. Sequential Encodings from Max-CSP into Partial Max-SAT. In O. Kullmann, editor, *SAT*, volume 5584 of *Lecture Notes in Computer Science*, pages 161–166. Springer, 2009.

[4] J. Argelich, A. Cabiscol, I. Lynce, and F. Manyà. New Insights into Encodings from MaxCSP into Partial MaxSAT. In *ISMVL*, pages 46–52. IEEE Computer Society, 2010.

[5] R. Asín, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell. Cardinality Networks: a Theoretical and Empirical Study. *Constraints*, 16(2):195–221, 2011.

[6] F. Azevedo and V.-H. Nguyen. Extra Constraints for the Social Golfers Problem. In *13th International Conference on Logic for Programming Artificial Intelligence and Reasoning- LPAR 2006, Short Papers Proceedings*, 2006.

[7] O. Bailleux and Y. Boufkhad. Efficient CNF Encoding of Boolean Cardinality Constraints. In F. Rossi, editor, *CP*, volume 2833 of *Lecture Notes in Computer Science*, pages 108–122. Springer, 2003.

[8] P. Barahona, S. Hölldobler, and V.-H. Nguyen. Efficient SAT-Encoding of Linear CSP Constraints. In *ISAIM*, 2014.

[9] P. Barahona, S. Hölldobler, and V.-H. Nguyen. Representative Encodings to Translate Finite CSPs into SAT. In H. Simonis, editor, *CPAIOR*, volume 8451 of *Lecture Notes in Computer Science*, pages 251–267. Springer, 2014.

[10] Y. Ben-Haim, A. Ivrii, O. Margalit, and A. Matsliah. Perfect Hashing and CNF Encodings of Cardinality Constraints. In A. Cimatti and R. Sebastiani, editors, *SAT*, volume 7317 of *Lecture Notes in Computer Science*, pages 397–409. Springer, 2012.

[11] C. Bessiere. Constraint Propagation. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 3, pages 27 – 81. Elsevier, 2006.

[12] A. Biere. Lingeling, Plingeling and Treengeling Entering the SAT Competition 2013. In A. B. Adrian Balint, M. Heule, and M. Järvisalo, editors, *Proceedings of SAT Competition 2013*, pages 51–52, 2013.

[13] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[14] J.-C. Chen. A New SAT Encoding of the At-Most-One Constraint. In *Proc. of the Tenth Int. Workshop of Constraint Modelling and Reformulation*, 2010.

[15] N. Eén and N. Sörensson. Translating Pseudo-Boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006.

[16] M. Ester, H. Kriegel, J. Sander, and X. Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *KDD*, pages 226–231, 1996.

[17] A. M. Frisch and P. A. Giannoros. 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.

[18] A. M. Frisch, T. J. Peugniez, A. J. Doggett, and P. W. Nightingale. Solving Non-Boolean Satisfiability Problems with Stochastic Local Search. In *in Proc. IJCAI-01*, pages 282–288, 2001.

[19] A. M. Frisch, T. J. Peugniez, A. J. Doggett, and P. W. Nightingale. Solving Non-Boolean Satisfiability Problems with Stochastic Local Search: A Comparison of Encodings. *J. Autom. Reason.*, 35:143–179, October 2005.

[20] M. Gebser, B. Kaufmann, and T. Schaub. The Conflict-Driven Answer Set Solver clasp: Progress Report. In E. Erdem, F. Lin, and T. Schaub, editors, *LPNMR*, volume 5753 of *Lecture Notes in Computer Science*, pages 509–514. Springer, 2009.

[21] I. P. Gent. Arc Consistency in SAT. In F. van Harmelen, editor, *Proceedings of the 15th Eureopean Conference on Artificial Intelligence, ECAI'2002*, pages 121–125. IOS Press, 2002.

[22] I. P. Gent and T. Walsh. CSPLib : A Benchmark Library for Constraints. In J. Jaffar, editor, *CP*, volume 1713 of *Lecture Notes in Computer Science*, pages 480–481. Springer Berlin Heidelberg, 1999.

[23] S. Gilpin and I. Davidson. Incorporating SAT Solvers into Hierarchical Clustering Algorithms: An Efficient and Flexible Approach. In *KDD*, pages 1136–1144, 2011.

[24] S. Hölldobler and V.-H. Nguyen. An Efficient Encoding of the At-Most-One Constraint. Technical report, Knowledge Representation and Reasoning Group 2013-04, Technische Universität Dresden, 01062 Dresden, Germany, 2013.

[25] H. Kautz and B. Selman. The State of SAT. *Discrete Appl. Math.*, 155:1514–1524, June 2007.

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

[27] H.-P. Kriegel, P. Kröger, J. Sander, and A. Zimek. Density-based clustering. *Data Mining and Knowledge Discovery*, 1(3):231–240, 2011.

[28] S. T. Mai, S. Goebl, and C. Plant. A similarity model and segmentation algorithm for white matter fiber tracts. In *ICDM*, pages 1014–1019, 2012.

[29] S. T. Mai, X. He, J. Feng, and C. Böhm. Efficient anytime density-based clustering. In *SDM*, pages 112–120, 2013.

[30] S. T. Mai, X. He, J. Feng, C. Plant, and C. Böhm. Anytime Density-based Clustering of Complex Data. *Knowledge and Information System (KAIS)*, (to appear).

[31] S. T. Mai, X. He, N. Hubig, C. Plant, and C. Böhm. Active density-based clustering. In *ICDM*, pages 508–517, 2013.

[32] N. Manthey. The SAT Solver RISS3G at SC 2013. volume B-2013-1 of *Department of Computer Science Series of Publications B*, pages 72–73. University of Helsinki, Helsinki, Finland, 2013.

[33] J. Marques-silva. Practical Applications of Boolean Satisfiability. In *In Workshop on Discrete Event Systems (WODES*. IEEE Press, 2008.

[34] V.-H. Nguyen and S. T. Mai. Solving the all-interval series problem: SAT vs CP. In N. T. Giang, H. Q. Thang, I. Khalil, S. H. Ngo, Y. Deville, and M. Bui, editors, *Proceedings of the Fifth Symposium on Information and Communication Technology, SoICT '14, Hanoi, Vietnam, December 4-5, 2014*, pages 65–74. ACM, 2014.

[35] V.-H. Nguyen, M. N. Velev, and P. Barahona. Application of Hierarchical Hybrid Encodings to Efficient Translation of CSPs to SAT. In *ICTAI*, pages 1028–1035. IEEE, 2013.

[36] S. D. Prestwich. Finding Large Cliques using SAT Local Search. volume Trends in Constraint Programming, pages 273–278. ISTE, 2007.

[37] S. D. Prestwich. CNF Encodings. In A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 75–97. IOS Press, 2009.

[38] F. Rossi, P. v. Beek, and T. Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006.

[39] J. M. Silva and I. Lynce. Towards Robust CNF Encodings of Cardinality Constraints . In C. Bessiere, editor, *CP*, volume 4741 of *Lecture Notes in Computer Science*, pages 483–497. Springer, 2007.

[40] C. Sinz. Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In P. van Beek, editor, *CP*, volume 3709 of *Lecture Notes in Computer Science*, pages 827–831. Springer, 2005.

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

[42] M. N. Velev. Exploiting Hierarchy and Structure to Efficiently Solve Graph Coloring as SAT. In G. G. E. Gielen, editor, *ICCAD*, pages 135–142. IEEE, 2007.

[43] T. Walsh. SAT v CSP. In *Principles and Practice of Constraint Programming - CP2000*, volume 1894 of *Lecture Notes in Computer Science*, pages 441–456. Springer, 2000.

[44] J. P. Warners. A Linear-Time Transformation of Linear Inequalities into Conjunctive Normal Form. *Information Processing Letters*, 68(2):63–69, 1998.