TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science | Institute of Artificial Intelligence | Knowledge Representation and Reasoning

An Efficient Encoding of the at-most-one Constraint

Steffen Hölldobler   Van Hau Nguyen

KRR Report 13-04

Mail to
Technische Universität Dresden
01062 Dresden

Bulk mail to
Technische Universität Dresden
Helmholtzstr. 10
01069 Dresden

Office
Room 2006
Nöthnitzer Straße 46
01187 Dresden

Internet
http://www.wv.inf.tu-dresden.de

An Efficient Encoding of the at-most-one Constraint

Steffen Hölldobler and Van Hau Nguyen

Knowledge Representation and Reasoning Group
Technische Universität Dresden, 01062 Dresden, Germany
`sh,hau@iccl.tu-dresden.de`

Abstract. One of the most widely used constraint during the process of translating a practical problem into an equivalent SAT instance is the at-most-one (AMO) constraint. Besides a brief survey of well-known AMO encodings, we will point out the relationship among several AMO encodings - the *relaxed ladder*, *sequential*, *regular* and *ladder* encodings. Therefore, it could help SAT community, especially researchers working in SAT encoding to avoid confusing among these encodings. The major goal of this paper is to propose a new encoding for the AMO constraint, named the *bimander* encoding which can be easily extended to cardinality constraints. Experimental results reveal that the proposed method is a significantly competitive one among other recently efficient methods. We will prove that the *bimander* encoding allows the unit propagation to achieve arc consistency. Furthermore, we will show that one of special case of *bimander* encoding outperforms the *binary* encoding, a well-known AMO encoding, in all experiments.

1 Introduction

Boolean Satisfiability problem (SAT) has been significantly investigated for the last two decades. SAT solving comprises two essential phases: encoding a certain problem into an equivalent SAT instance, and then solving the resulting instance by advanced SAT solvers. Compared with considerable improvements in the design and implementation of SAT solvers, in the last decade the progress on SAT encoding has been very limited. Moreover, different encodings when translating a Constraint Satisfaction Problem (CSP) into a SAT instance can get different sizes and difficulties of the resulting CNF formula.

Generally, it is well-known that no particular encoding performs better than others, whereas the following aspects of an encoding are always considered:

– the number of clauses required;
– the number of auxiliary variables required; and
– the strength of the encoding in terms of performance of unit propagation in SAT solvers.

The most natural and common way to translate a CSP is the *direct* encoding (see [1]). The *direct* encoding requires the at-least-one (ALO) and at-most-one

(AMO) constraints to let one CSP variable to assign exactly one value. While the ALO constraint is trivial to translate to a single clause, the AMO constraint is more complicated and it has been intensively studied ([2,3,4,5,6]). The AMO constraint, as its name, requires that at most one of n propositional variables is allowed to be *TRUE*, shortly denoted by $\leq_1 (X_1, ..., X_n)$. Interest of the AMO constraint has increased to meet requirements of different applications, such as computer motographs [7], partial Max-SAT [8], and cardinality constraints [3].

Inspired by many interesting and recent results [2,3,4], especially when Prestwich used the *binary* $AMO(X)$ encoding [9,10] to solve successfully many large instances with a standard SAT local search method [5,6], after surveying several of well-known methods of the AMO constraint, we will introduce a new way of encoding this constraint.

In the brief survey, we will point out the identity of three AMO encodings -the *relaxed ladder*, the *sequential* and the *regular* encodings. These encodings are exact the *ladder* encoding after removing redundant clauses. Therefore, it could help SAT community, especially researchers working in SAT encoding to avoid confusing among these encodings.

The new encoding, named the *bimander* encoding, requires $\frac{n^2}{2m} + nlog_2 m - \frac{n}{2}$ binary clauses, and $log_2 m (1 \leq m \leq n)$ additional variables, where m is the number of disjointed subsets by dividing from the original set of n Boolean variables $\{X_1, ..., X_n\}$. Additionally, the *bimander* encoding can be easily extended to cardinality constraints, $\leq k(X_1, ..., X_n)$, which expresses that there are no more than k of the n Boolean variables $X_i, 1 \leq i \leq n$, assigned simultaneously to *TRUE* values. To the best of our knowledge, our encoding is the one that requires least number of additional variables among known encoding methods, except for the *pairwise* encoding which needs no additional variables. With respect to scalability, the *bimander* encoding can be adjusted by changing the number of subsets m to get a suitable encoding. For example, by setting the parameter m to specific values, the *binary* and *pairwise* encodings can be expressed as special cases of the *bimander* encoding. Interestingly, a special case of the *bimander* encoding, setting $m = \lceil \frac{n}{2} \rceil$, outperforms the *binary* encoding, a well-known encoding, in all our experiments. It is important to note that our encoding allows unit propagation (UP) to preserve arc consistency.

The structure of the paper is as follows. In Section 2, we briefly represent efficient and recent approaches to encode the AMO constraint. In Section 3 we describe the new encoding for the AMO constraint , the so-called *bimander* encoding. In section 4, we compare the *bimander* encoding with others through experiments. Finally, we give conclusions and future research works in Section 5.

2 Existing Encodings

Before giving a brief survey of almost all of well-known existing AMO encodings of SAT encoding, we first define notions and notations, mainly following [3].

Let $X_i, 1 \leq i \leq n$, be Boolean variables; let A be a possibly empty set of auxiliary Boolean variables supporting the encoding; and let $\phi(X, A)$ be an encoding of $\leq_1 (X_1, ..., X_n)$. The encoding $\phi(X, A)$ is *correct* if and only if:

- any assignment α 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 $\hat{x}$ to $X = (X_1, ..., X_n)$ in which if $\hat{x}$ has *more than one* literals assigned *TRUE* values, unit propagation *UP* detects a conflict (generating an empty clause).

On the SAT side, *UP* plays a crucial role in SAT solver by being a major deduction in DPLL [11,12], whereas on the CSP side, arc consistency is the most important technique since it is the best trade-off between the amount and the cost of pruning. Therefore, when translating a CSP instance into an equivalent SAT instance, in order to know how powerful the performance of *UP* of that encoding is, one should pay much attention to determine whether *UP* of the equivalent SAT instance enforces arc consistency property.

UP of a SAT encoding of $\leq_1 (X_1, ..., X_n)$ constraints achieves the same pruning as arc consistency on the original CSP if:

- whenever any variable $X_i, 1 \leq i \leq n$, is assigned to *TRUE*, all the other variables must be forced to the values *FALSE* under UP.

In following sections, generally $AMO(X)$, $ALO(X)$ and $EO(X)$ denote the at-most-one, at-least-one and Exactly-One constraint, respectively for the set of positional variables X. Furthermore, for the sake of convenience, we will illustrate those encoding on a running example through the set consisting 8 Boolean variable, $X = X_1, ..., X_8$.

We briefly represent several well-known and efficient encodings of the $\leq_1 (X_1, ..., X_n)$ constraint. Some notations used was taken from [3] and we give comments for each encoding if they are necessary.

2.1 The Pairwise Encoding

There are several different names of this encoding: the *naive* encoding [13,2], the *pairwise* encoding [14,6], and the *binomial* encoding [3]. In this paper, we refer to it as the *pairwise* encoding. The idea of this encoding is to express that no possible combinations of two variables are simultaneously *TRUE* , therefore as soon as one literal is *TRUE*, the all others must be *FALSE*:

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

In the running example, the *pairwise* encoding produces the following clauses:

$$\begin{array}{r}\bar{X}_1 \vee \bar{X}_2,\ \bar{X}_1 \vee \bar{X}_3,\ \bar{X}_1 \vee \bar{X}_4, \ldots,\ \bar{X}_1 \vee \bar{X}_8 \\ \bar{X}_2 \vee \bar{X}_3,\ \bar{X}_2 \vee \bar{X}_4, \ldots,\ \bar{X}_2 \vee \bar{X}_8 \\ \bar{X}_3 \vee \bar{X}_4, \ldots,\ \bar{X}_3 \vee \bar{X}_8 \\ \vdots \quad \\ \bar{X}_7 \vee \bar{X}_8\end{array}$$

The *pairwise* encoding is the most widely known one for encoding the *AMO* constraint. Although this method does not need any auxiliary variables, it requires a quadratic number of clauses (see Table 1). Generally, this encoding performs acceptably well, particularly for small cases, but usually produces impractical large formulas which can be inferior to other methods, especially for encoding the At-Most-k constraint. Nevertheless, the *pairwise* encoding is not only commonly used in practical, but also easily combined with other encoding methods [2,4].

2.2 The Binary Encoding

Frisch et al. [9,10] firstly proposed the *binary* encoding (Prestwich independently named the *bitwise* encoding [6,15]), and Prestwich used this encoding to solve successfully a number of large instances with a standard SAT local search method [5,6].

New Boolean variables $B_1, ..., B_{\lceil log_2 n \rceil}$ [1] are introduced. The expected clauses are following: where B_j (or $\bar{B}_j$) is the bit j of $i-1$ represented by a binary string is 1 (or 0).

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

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

The running example is represented by the *binary* encoding as follows:

$$\begin{array}{llll}\bar{X}_1 \vee \bar{B}_1 & \bar{X}_2 \vee B_1 & \bar{X}_3 \vee \bar{B}_1 \ldots & \bar{X}_8 \vee B_1 \\ \bar{X}_1 \vee \bar{B}_2 & \bar{X}_2 \vee \bar{B}_2 & \bar{X}_3 \vee B_2 \ldots & \bar{X}_8 \vee B_2 \\ \bar{X}_1 \vee \bar{B}_3 & \bar{X}_2 \vee \bar{B}_3 & \bar{X}_3 \vee \bar{B}_3 \ldots & \bar{X}_8 \vee B_3\end{array}$$

The hidden 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*, $1 \leq i \leq n$, then we immediately infer that the other variables $X_{i'}$ must be *FALSE*, for any $i' \neq i$. With this encoding *UP* maintains arc consistency property.

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

2.3 The Commander Encoding

Klieber and Kwon [2] described the *commander* encoding by dividing the set of Boolean variables $\{X_1, ..., X_n\}$ into m disjointed subsets $G_1, ..., G_m$, and introducing a commander variable c_i considered as a candidate of each group G_i, $1 \leq i \leq m$. The *commander* encoding requires the following clauses:

1. *Exactly-One* variable in each group, consisting G_i and corresponding $\bar{c}_i$, is assiged to the *TRUE* value. Whereas the ALO constraint is trivial to translate to a single clause, *AMO* can be encoded by any known methods :

$$\bigwedge_{i=1}^{m} \langle EO(\bar{c}_i \cup G_i)\rangle = \bigwedge_{i=1}^{m} \langle AMO(\bar{c}_i \cup G_i)\rangle \wedge \bigwedge_{i=1}^{m} \langle ALO(\bar{c}_i \cup G_i)\rangle$$

   For the running example, we divide the set $X = \{X_1, ..., X_8\}$ into $m = 4$ disjointed subsets: $G_1 = \{X_1, X_2\}$, $G_2 = \{X_3, X_4\}$, $G_3 = \{X_5, X_6\}$ and $G_4 = \{X_7, X_8\}$. Then, four Boolean variables c_1, c_2, c_3 and c_4 are introduced as a candidate of G_1, G_2, G_3 and G_4 respectively. Consequently, the *commander* produces the following clauses:

$$AMO(\bar{c}_1, X_1, X_2) \wedge (\bar{c}_1 \vee\ X_1 \vee X_2) \wedge \cdots \wedge AMO(\bar{c}_4, X_7, X_8) \wedge (\bar{c}_4 \vee X_7 \vee X_8)$$

2. At most one commander variable is assigned *TRUE*. This constraint can be encoded either by the *pairwise* encoding or by the *commander* method:

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

   The following clauses are generated in the running example:

$$AMO(c_1, c_2, c_3, c_4)\rangle$$

Compared with the *pairwise* encoding, the *commander* encoding requires less the number of clauses, and introduces an acceptable number of new variables (see Table 1). The *commander* encoding also allows *UP* to preserve arc consistency property.

2.4 The Product Encoding

Chen [4] proposed an encoding for *AMO* constraint, named the *product* encoding. Insteading of encoding the constraint consisting of n propositional variables $\leq_1$ $(X_1, ..., X_n)$, he encoded the constraint consisting of corresponding n point $\leq_1$ $\{(u_i, v_j), 1 \leq i \leq u, 1 \leq j \leq v, p \times q \geq n\}$. The hidden idea can be explained as follows:

1. Firstly mapping each variable $X_k, 1 \leq k \leq n$ onto one corresponding point (u_i, v_j) where $u_i \in U = \{u_1, ..., u_p\}, v_i \in V = \{v_1, ..., v_q\}$.
2. Then the *product* encoding is represented:

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

whereas $AMO(U)$ and $AMO(V)$ can be encoded by either a recursive or another way.

Regard to the running example, we choose $p = 3$ and $q = 3$, and we use the *paiwise* encoding for $AMO(U)$ and $AMO(V)$. The derived clauses are as follows:

$$\begin{aligned}
AMO(U) &: (\bar{u}_1 \vee \bar{u}_2) \wedge (\bar{u}_1 \vee \bar{u}_3) \wedge (\bar{u}_2 \vee \bar{u}_3) \\
AMO(V) &: (\bar{v}_1 \vee \bar{v}_2) \wedge (\bar{v}_1 \vee \bar{v}_3) \wedge (\bar{v}_2 \vee \bar{v}_3) \\
AMO(X) = &AMO(U) \wedge AMO(V) \wedge \\
&(\bar{X}_1 \vee u_1) \wedge (\bar{X}_1 \vee v_1) \wedge (\bar{X}_2 \vee u_2) \wedge (\bar{X}_2 \vee v_1) \wedge \\
&(\bar{X}_3 \vee u_3) \wedge (\bar{X}_3 \vee v_1) \wedge (\bar{X}_4 \vee u_1) \wedge (\bar{X}_4 \vee v_2) \wedge \\
&(\bar{X}_5 \vee u_2) \wedge (\bar{X}_5 \vee v_2) \wedge (\bar{X}_6 \vee u_3) \wedge (\bar{X}_6 \vee v_2) \wedge \\
&(\bar{X}_7 \vee u_1) \wedge (\bar{X}_7 \vee v_3) \wedge (\bar{X}_8 \vee u_2) \wedge (\bar{X}_8 \vee v_3)
\end{aligned}$$

2.5 The Sequential Encoding

By building a count-and-compare hardware circuit and translating this circuit to an equivalent CNF formula, Sinz [13] introduced an encoding of $\leq_k (X_1, ..., X_n)$, namely the *sequential* encoding.

For the case $k = 1$, the set of $AMO(X)$ clauses is followed:

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

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

The running example is represented as follows:

$$\begin{array}{lll}
\bar{X}_1 \vee s_1 & & \\
\bar{X}_2 \vee s_2 & \bar{s}_1 \vee s_2 & \bar{X}_2 \vee \bar{s}_1 \\
\bar{X}_3 \vee s_3 & \bar{s}_2 \vee s_3 & \bar{X}_3 \vee \bar{s}_2 \\
\bar{X}_4 \vee s_3 & \bar{s}_3 \vee s_4 & \bar{X}_4 \vee \bar{s}_3 \\
\bar{X}_5 \vee s_4 & \bar{s}_4 \vee s_5 & \bar{X}_5 \vee \bar{s}_4 \\
\bar{X}_6 \vee s_5 & \bar{s}_5 \vee s_6 & \bar{X}_6 \vee \bar{s}_5 \\
\bar{X}_7 \vee s_6 & \bar{s}_6 \vee s_7 & \bar{X}_7 \vee \bar{s}_6 \\
\bar{X}_8 \vee \bar{s}_7 & &
\end{array}$$

As Marques-Silva and Lynce [14] pointed out that the sequence $s_1, ..., s_{n-1}$ is of the form *"0...01...1"* and whenever any Boolean variable X_i is assigned to *TRUE* (or *1*), $1 \leq i \leq n$, consequently, under unit propagation all the other variables X_j must be forced to *FALSE* (or *0*), $1 \leq j \neq i \leq n$.

2.6 The Ladder Encoding

Gent and Nightingale used the *ladder* structure, originally proposed by Gent et al.[16], to describe a new encoding for the *alldifferent* constraint into SAT [17]. It was named the *ladder* encoding related to its *ladder* structure.

Without loosing the correctness property, we reverse the condition of $n-1$ additional Boolean variables, $s_1, ..., s_{n-1}$ in [13,14] (in [17], $y_1, ..., y_{n-1}$) which satisfy the following *ladder* clauses:

$$\bigwedge_{i=1}^{n-2} (s_{i+1} \vee \bar{s}_i) \tag{2}$$

and adds the *channeling* clauses:

$$\bigwedge_{i=1}^{n} \langle (s_i \wedge \bar{s}_{i-1}) \Longleftrightarrow X_i \rangle \tag{3}$$

where

$$s_0 = 0 \wedge s_n = 1 \tag{4}$$

are set.

The idea hidden is simple. While the sequence $s_1, ..., s_{n-1}$ is a sequence of *0's* or more *1's* values, and the rest of variables assigned *1* values; i.e., the sequence $s_1, ..., s_{n-1}$ is of the form *"0...01...1"* (in [17], *"1...10...0"*). Thus, there is at most one adjacent pair of variables s_{i-1} and s_i where $s_{i-1} = 0 \wedge s_i = 1, 1 \leq i \leq n$. As soon as a variable X_i is assigned to $TRUE, 1 \leq i \leq n$, and consequently, all the other variables X_j are forced to *FALSE*, $1 \leq j \neq i \leq n$, under unit propagation.

By combining (2),(3) and (4) we obtain the following set of clauses:

$$\bigwedge_{i=1}^{n} \langle (\bar{s}_{i-1} \vee s_i) \wedge (\bar{s}_i \vee s_{i-1} \vee X_i) \wedge (\bar{X}_i \vee s_i) \wedge (\bar{X}_i \vee \bar{s}_{i-1}) \rangle \tag{5}$$

We can prove easily that the clause $(\bar{s}_i \vee s_{i-1} \vee X_i)$ in (5) is redundant since it does not affect the correctness of the at-most-one constraint. Moreover, both the *sequential* encoding and the *ladder* encoding require the same $n-1$ additional Boolean variables. Now we realize that the *sequential* encoding is exact the *ladder* encoding without the redundant clauses.

Prestwich [6] supposed the *relaxed ladder* encoding which is the *ladder* encoding without these redundant clauses. It is easy to see that the *relaxed ladder* encoding and the *sequential* encoding are the same. Argelich et al. [18] also noticed that the *sequential* encoding is a reformulation of a *regular* encoding [19]. In fact, it is a simple matter to prove that the *regular* encoding and the *ladder* encodings are the same.

In conclusion, we shown that the two *AMO* encodings -the *relaxed ladder* encoding and the *sequential* encoding are the same. These encodings are exact the *ladder* encoding or the *regular* encoding after removing redundant clauses. Interestingly, there are various related works. Tamura et al. used this structure for the *order* encoding in their SAT-based solving system [20]. Bailleux et al. [7] referred to this structure as the *unary representation* which was used during their translation of cardinality constraints and pseudo-Boolean constraints to SAT formulas([7,21,22]).

Recently, Martins et. al [23] compared both encodings in their paper. Argelich et al. [18] compared two encodings of the *alldifferent* constraints, one based on the *sequential* encoding and the other based on the *ladder* encoding. To the best of our knowledge, we are not aware of any paper mentioning about the relationship among the *ladder*, *sequential*, *relaxed ladder* and *regular* encodings. We hope that this work could help the SAT community to recognize the similarities of these encodings.

3 The Bimander Encoding

The general idea of a new encoding is based on both the ideas of the *binary* encoding and the *commander* encoding. We refer to it as the *bimander* 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 group $G_i, 1 \leq i \leq m$, consists $g = \lceil \frac{n}{m} \rceil$ variables. However, instead of introducing commander variables like in the *commander* encoding, we introduce a set of auxiliary Boolean 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 as the roles of the commander variables in the *commander* encoding.

The *bimander* encoding is the conjunction of the clauses obtained as follows:

1. *At most* ***one*** *variable in each group can be* TRUE. we encode this constraint for each group $G_i, 1 \leq i \leq m$, by using the *pairwise* method.

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

Regard to the running example, by chooding $m = \sqrt{n} = 3$ we have:

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

2. The following clauses are constraints between each variable in a group and commander variables:

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

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 clauses are generated in the running example:

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

Compared with the *commander* encoding, in the *bimander* encoding we do not add any constraints among the binary sequences since any combination of auxiliary Boolean variables $B_1, ..., B_{\lceil log_2 m \rceil}$ of a corresponding group is different from any combinations of all the other corresponding groups.

Let us first prove some important properties of the *bimander* encoding.

Correctness.

Now we assume that we have a partial assignment $x = (X_1, ..., X_l), 1 \leq l \leq n$, with at most one assigned variable to *TRUE*. For the case of none variables assigned to *TRUE* value (all variables assigned *FALSE*), then the first condition is trivially satisfied, so is the second condition. In the case of an existing one $X_i = TRUE, 1 \leq i \leq n$, there is a corresponding sequence of truth values assigned to $\{B_1, ...B_{\lceil log_2 m \rceil}\}$. The second condition is satisfied as well. Therefore, the partial assignment x can possibly be extended to a complete assignment that satisfies the two above conditions.

Suppose that we have a partial assignment $x = (X_1, ..., X_l), 1 \leq l \leq n$, with *more than one* assigned variables to *TRUE*, assuming that two of them are $X_i = TRUE$ and $X_j = TRUE, 1 \leq i \neq j \leq l$. Each of these two assignments force a corresponding pattern of truth values to be assigned to the sequence of $\{B_1, ...B_{\lceil log_2 m \rceil}\}$. As a result, the sequence exists one propositional variable $B_k, 1 \leq k \leq \lceil log_2 m \rceil$, that is assigned both *TRUE* and *FALSE*. It is a contradiction!

Hence, if any partial assignment has more than one literal assigned *TRUE* values, then *UP* produces an empty clause. It means that this partial assignment can not be extended to a complete assignment. In conclusion, the *bimander* encodes correctly the at-most-one constraint.

Propagation strength. Suppose that we have a partial assignment $x = (X_1, ..., X_l)$, $1 \leq l \leq n$, consisting exactly *one* variable set to *TRUE*. Now we will show that *UP* forces all other variables to *FALSE*. Indeed, we assume a variable $X_{i,j} = TRUE$ which is the j^{th} variable in the group $G_i, 1 \leq i \leq m$, then this assignment forces a corresponding pattern of *TRUE* values to $\{B_1, ...B_{\lceil log_2 m \rceil}\}$. By following the first condition, all other variables in group G_i are set to *FALSE*. By following the second condition, all the other variables in group $G_{i'}, 1 \leq i' \neq i \leq m$ are set to *FALSE* since they have different patterns of *TRUE* values $\{B_1, ...B_{\lceil log_2 m \rceil}\}$ of the corresponding $X_{i,j} = TRUE$. In conclusion, the unit propagation (UP) of the *bimander* encoding forces arc consistency property.

Complexity. As we mentioned, we need a set of $\lceil log_2 m \rceil$ additional Boolean variables. The first constraint encoding by the *pairwise* method requires $m * [\frac{g(g-1)}{2}] = \frac{n(\frac{n}{m}-1)}{2}$ new clauses. The second constraint 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.

Related to scalability, it is interesting to note that the *bimander* encoding is a general case of several encodings. For example:

- the *pairwise* encoding is a special case of the *bimander* encoding when $m = 1$;
- the *commander* encoding is a special case of the *bimander* encoding when $m = 2$ (when both encodings divide into *2* subsets); and

– the *binary* encoding is a special case of the *bimander* encoding when $m = n$.

It is also important to note that the *bimander* encoding can be easily generalized to encode the At-Most-k constraint, which is described as follows.

1. *At most **k** variables in each group can be true.* We encode this constraint for each group $G_i, 1 \leq i \leq m$, by using the *pairwise* (or another) method.
2. The constraints between each variable in a group and commander variables are encoded by the following clauses:

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

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

4 Comparisons and Experimental Evaluations

In this section, we first show almost known methods for the *AMO* constraint. Then we compare our encoding with several common and efficient methods through experiments.

4.1 Comparisons

Table 1 presents a summary of main approaches of the *AMO* encoding methods. The "Clauses" and "auxiliary Vars" columns show the number of clauses required and auxiliary variables corresponding to the methods. The "Con." column indicates that whether *UP* of the corresponding encoding achieves the arc consistency property or not. The "Origin" column infers the original publication where the method had been introduced. We use m to denote the disjointed subsets by dividing the set of Boolean variables $\{X_1, ..., X_n\}$ occurring in the *bimander* encoding.

Table 1. A summary of almost known methods of the *AMO* encoding.

Methods	Clauses	auxiliary Vars	Con.	Origin
pairwise	$\binom{n}{2}$	0	AC	none
linear	$8n$	$2n$	search	[24]
totalizer	$O(n^2)$	$O(nlog(n))$	AC	[7]
binary	$nlog_2 n$	$\lceil log_2 n \rceil$	AC	[10]
sequential	$3n-4$	$n-1$	AC	[13]
sorting networks	$O(nlog_2^2 n)$	$O(nlog_2^2 n)$	AC	[21]
commander	$\sim 3n$	$\sim \frac{n}{2}$	AC	[2]
product	$2n + 4\sqrt{n} + O(\sqrt[4]{n})$	$2\sqrt{n} + O(\sqrt[4]{n})$	AC	[4]
card. networks	$6n-9$	$4n-6$	AC	[25]
PHFs-based	$nlog_2 n$	$\lceil log_2 n \rceil$	AC	[26]
bimander	$\frac{n^2}{2m} + nlog_2 m - \frac{n}{2}$	$log_2 m, 1 \leq m \leq n$	AC	this paper
bimander ($m = \frac{n}{2}$)	$nlog_2 n - \frac{n}{2}$	$\lceil log_2 n \rceil - 1$	AC	this paper

With respect to the scalability, the *bimander* encoding can be adjusted to get a suitable encoding. In fact, the *bimander* encoding requires the least auxiliary variables, excepting the *pairwise* encoding, among known encoding methods. The *totalizer* encoding proposed by Bailleux al et. [7] requires clauses of size at most *3*, and the *commander* encoding proposed by Klieber and Kwon [2] needs m (number of disjointed subsets) clauses of size $\lceil \frac{n}{m} + 1 \rceil$, whereas the *sequential*, *binary* and *bimander* encodings require only binary clauses.

4.2 Experimental Evaluations

In order to evaluate the different encodings, we choose several difficult and well-known problems which have been benchmarks not only on the CSP side, but also on the SAT side. These benchmarks have been used in the CSP-solvers and SAT-solvers competitions. Moreover, we take two different parameters m for the *bimander* encoding, one is $m = \sqrt{n}$ and the other one is $m = \frac{n}{2}$.

We used CLASP 2 [27] which is one of among state-of-the-art SAT solvers [28]. All experiments were executed on a 2.66 Ghz , Intel Core 2 Quad processor with a memory limit of 3.8 GB running Ubuntu 10.04, and all runtimes are measured in seconds. The dashes mean that running times of instances were over timeout of 3600 seconds. The italic font designates the minimum time for a certain instance. We abbreviate *pairwise*, *sequential*, *commander*, *binary*, *product* encoding, and *bimander* encoding as *pw*, *seq*, *cmd*, *bi*, *pro* and *bim* respectively.

The Pigeon-Hole Problem This problem has been a common benchmark on the SAT and CSP sides. The goal of the problem is to prove that p pigeons can not be fit in $h = p-1$ holes. We use this problem to compare the performance of the constraint $\leq_1 (X_1, ..., X_n)$ of the various encodings, like Frisch and Giannaros [3], and Klieber and Kwon [2].

Table 2. A comparison of running times of well-known encodings performed by CLASP on unsatisfiable Pigeon-Hole problem. Runtimes reported are in seconds.

method	pw	seq	cmd	bi	pro	bim	
size						$m = \sqrt{n}$	$m = \frac{n}{2}$
10	2.1	0.73	0.56	0.80	0.22	0.33	*0.22*
11	22.1	5.79	4.46	6.59	6.13	5.10	*2.10*
12	244.5	117.3	43.2	29.5	43.21	38.19	*26.06*
13	-	1604.1	352.5	142.6	736.2	546.9	*64.91*
14	-	-	-	1271	-	-	*560*

Table 2 shows that the Pigeon-Holes instances seem very hard to deal with. The *bimander* encoding performs the best for all cases followed by the *binary* encoding. The *pairwise* encoding is the worse.

The All-Interval Series Problem We take the All-Interval Series (AIS) problem as a benchmark in which the performance of an encoding is heavily influenced by the performance of encoding the *AMO* constraint. AIS is one of classical CSPs and usually regarded as a difficult benchmark to find all solutions (see prob007 in [29]).

Table 3. A comparison of running times of well-known encodings performed by CLASP solver on the AIS problem. Runtimes reported are in seconds.

method	pw	seq	cmd	bi	pro	bim		solutions
size						$m = \sqrt{n}$	$m = \frac{n}{2}$	
7	0.05	0.03	0.02	0.02	0.05	*0.01*	0.02	32
8	0.56	1.07	0.6	*0.2*	0.49	0.6	0.6	40
9	5.33	8.9	0.37	0.27	5.61	0.3	*0.24*	120
10	61.7	104	1.7	1.58	60.7	1.95	*1.46*	296
11	972	1387	11.9	8.9	269	11.3	*6.7*	648
12	-	-	78	49	-	69	*43*	1328
13	-	-	517	356	-	504	*276*	3200
14	-	-	3200	2748	-	3537	*2005*	9912

Excepting for two small cases, Table 3 shows that the variant of the *bimander* encoding with $m = \frac{n}{2}$ significantly surpasses all the others. Moreover, for three last instances this variant performs in a reasonable time, whereas the *pairwise* and *sequential* encodings carry out more than 3600 seconds. The *binary* encoding gives rather good results. Another variant of the *bimander* encoding with $m = \sqrt{n}$ and the *commander* encoding perform similarly. The *pairwise*, *sequential* and *product* encodings perform poorly.

The Langford Problem This problem is a classical one of CSPs (see prob024 in [29]) and it is used as a hard benchmark as well. The aim of problem is either to find all the sequences of $2 * n$ numbers $1, 1, 2, 2, ..., n, n$, where there exists one number between the two $1s$, and two numbers between the two $2s$, and generally k numbers between the two ks, or to prove that there are no solutions.

Table 4. A comparison of running times of well-known encodings performed by CLASP solver on the Langford problem. Runtimes reported are in seconds.

method	pw	seq	cmd	bi	pro	bim		solutions
size						$m = \sqrt{n}$	$m = \frac{n}{2}$	
8	0.03	0.04	*0.03*	0.05	0.04	0.04	0.04	150
9	0.24	0.25	0.25	0.24	0.37	*0.23*	0.26	unsat
10	1.65	1.88	1.65	1.87	2.03	*1.60*	2.02	unsat
11	*7.2*	7.6	7.5	12.5	7.2	8.93	12.2	17792
12	59.3	62.1	56.7	86.2	*53.6*	79.4	58.8	108144
13	2275	*1328*	1462	1955	1443	1927	1925	unsat
14	30842	*14946*	16204	21308	-	20125	19734	unsat

Table 4 shows that three encodings - *binary*, and two variants of *bimander* - show no clear difference. While the *pairwise*, and *product* encodings perform worse, the *sequential* tends to be the fastest one for two large cases in term of running time, and followed by the *commander* encoding.

The Quasigroup With Holes Problem Achlioptas et al. [30] introduced a method for generating satisfiable Quasigroup With Holes (QWHs) instances which are NP-hard and considered as a structured benchmark domain for the study of CSP and SAT. Moreover, the method can tune the generator to output hard problem instances. We experimented these QWHs instances with different levels of hardness.

Table 5. The running time comparison of several encodings performed by CLASP solver on QWH instances. Runtimes reported are in seconds.

method	pw	seq	cmd	bi	pro	bim	
size						$m = \sqrt{n}$	$m = \frac{n}{2}$
qwh.order30.holes320	0.46	0.28	0.23	0.25	0.23	*0.20*	0.22
qwh.order35.holes405	3.6	3.5	10.3	6.5	5.7	*1.6*	2.1
qwh.order40.holes528	134	115	124	120	241	*58.9*	159
qwh.order40.holes544	39.2	*14.5*	47.8	123	46.7	70.8	154
qwh.order40.holes560	121	65.3	55.6	119	33.1	*21.2*	53.2
qwh.order33.holes381	58.7	435	174	94.2	108.0	*12.7*	92.3
total	356.96	633.58	411.93	462.95	434.73	*165.4*	460.82

As shown in Table 5, it is interesting to notice that the variant of the *bimander* encoding with $m = \sqrt{n}$ is clearly the best overall encoding in term of total runtime. Furthermore, except for the instance qwh.order40.holes544, this encoding is clearly faster than other encodings for all the other instances. Surprisingly, the *pairwise* encoding performs very well followed by the *commander* encoding. In general, the variant of the *bimander* encoding with $m = \frac{n}{2}$, the *binary* and the *product* encoding are slightly similar. Although the *sequential* encoding carries out the instance qwh.order40.holes544 fastest, its performance is poor in overall.

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

5 Conclusions and Future Works

Inspired by being remarkably successful at solving hard and practical problems of SAT solving, many problems that were solved previously by other methods can now be solved more effectively by translating them to equivalent SAT problems, and using 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 solving many problems gets benefits from the efficiency of the encoding of the *AMO* constraint.

The paper has four contributions. Firstly, we pointed out that the *ladder* encoding exactly consists of the *sequential* encoding and a set of redundant clauses. Moreover, the *relaxed ladder* encoding [6] and the *sequential* encoding [13] are the same. Two encodings - *ladder* [17] and *regular* [19]- are the same as well. Hence, the prior two encodings (*relaxed ladder* and *sequential*) are exact the latter two encodings (*ladder* and *regular*) after removing redundant clauses. Interestingly, these ideas were exploited in the *unary representation* [7] and the *order* encoding [20]. We hope that our work could help researchers working in SAT encoding to avoid confusing among these encodings.

Secondly, the major goal of the paper is to propose a new method to encode the at-most-one constraint to a SAT formula, called the *bimander* encoding. Compared to many efficient and well-known *AMO* methods, the *bimander* encoding requires the least auxiliary variables, with exception of the *pairwise* encoding (requires no additional variable). Although the *commander* and *bimander* encodings use the same approach by dividing the original set of Boolean variables, the *commander* requires clauses of size $\lceil \frac{n}{m} + 1 \rceil$ (where m is the number of disjointed subsets), whereas the *bimander* encoding requires *only binary clauses*. We believe that this helps the *bimander* encoding performs better than the *commander* encoding. Moreover, this new encoding has the advantage of high scalability, and it can easily be adjusted in term of the number of additional Boolean variables to get a suitable encoding. For example, the *pairwise* or *binary* encodings are special cases of the *bimander* encoding by setting certain parameters. The important feature of the new encoding is to allow unit propagation to preserve arc consistency property.

Thirdly, this paper also proposes a special case, when dividing the original Boolean variables into $m = \lceil \frac{n}{2} \rceil$ disjoint subsets. From a theoretical point of view, this case is better than the *binary* encoding in term of both the number of auxiliary variables and clauses required. From a practical point of view, we show that the special case of the *bimander* encoding ($m = \lceil \frac{n}{2} \rceil$) performs better than the *binary* encoding in all experiments in term of runtime.

Fourthly, in practice, the *bimander* encoding is easy to implement to get different encodings. Our experimental results reveal that the variants of the *bimander* encoding are very competitive with the others. For instance, they are the best in three of four benchmarks.

In general, a smaller encoding with respect to the number of clauses, literals or variables tends to perform better. However, a good encoding for one algorithm might be bad for others. For this reason, the best way to evaluate one encoding is to experiment on particular problems. A side benefit of our encoding is to give more the number of SAT encodings, and then to offer to SAT community more choices to be able to deal with a wide variety of real-world applications. This paper should also be viewed as a preliminary attempt to provides a further choice to encode the very common *alldifferent* constraint (see [17]).

An interesting our future research is to study how the number of disjointed subsets could affect the *bimander* encoding through realistic problems. It would be particularly useful to further supplement by implementing and comparing our extended At-Most-k encoding with others. 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 spur further what makes an encoding perform better than others (in specific situations).

Acknowledgements We would like to thank Christoph Wernhard for many useful suggestions, and Martin Gebser for his helpful discussions. We also wish to thank Carla Gomes for her kindly providing us the QWH's generator.

References

1. Walsh, T.: SAT v CSP. In: Principles and Practice of Constraint Programming - CP2000. Volume 1894 of Lecture Notes in Computer Science., Springer (2000) 441–456
2. Klieber, W., Kwon, G.: Efficient CNF encoding for selecting 1 from n objects. In: the Fourth Workshop on Constraint in Formal Verification(CFV). (2007)
3. 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)
4. 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)
5. Prestwich, S.D.: Variable dependency in local search: Prevention is better than cure. In Silva, J.M., Sakallah, K.A., eds.: Theory and Applications of Satisfiability Testing - SAT 2007, 10th International Conference, Lisbon, Portugal, May 28-31, 2007, Proceedings. Volume 4501 of Lecture Notes in Computer Science., Springer (2007) 107–120

6. Prestwich, S.D.: Finding large cliques using sat local search. Volume Trends in Constraint Programming., ISTE (2007) 273–278
7. Bailleux, O., Boufkhad, Y.: Efficient CNF encoding of boolean cardinality constraints. Principles and Practice of Constraint Programming 9th International Conference CP-2003 (2003) 108–122
8. Argelich, J., Cabiscol, A., Lynce, I., Manyà, F.: Sequential encodings from max-csp into partial max-sat. In Kullmann, O., ed.: Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings. Volume 5584 of Lecture Notes in Computer Science., Springer (2009) 161–166
9. Frisch, A.M., Peugniez, T.J., Doggett, A.J., Nightingale, P.W.: Solving non-boolean satisfiability problems with stochastic local search. In: in Proc. IJCAI-01. (2001) 282–288
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** (2005) 143–179
11. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Commun. ACM **5**(7) (1962) 394–397
12. 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, ACM (2001) 530–535
13. Sinz, C.: Towards an optimal CNF encoding of boolean cardinality constraints. In: Principles and Practice of Constraint Programming, 11th International Conference, CP 2005, Spain, October 2005, Proceedings. Volume 3709 of Lecture Notes in Computer Science., Springer (2005) 827–831
14. Silva, J.M., Lynce, I.: Towards robust CNF encodings of cardinality constraints. In: Proc. 13th International Conference on Principles and Practice of Constraint Programming CP-2007. Volume 4741 of Lecture Notes in Computer Science., Springer (2007) 483–497
15. Prestwich, S.D. In: CNF Encodings. IOS Press (2009) 75–98
16. Ian P. Gent, P.P., Smith, B.M.: A 0/1 encoding of the gaclex constraint for pairs of vectors. In: ECAI 2002 workshop W9: Modelling and Solving Problems with Constraints, University of Glasgow (2002)
17. Gent, I., Nightingale, P.: A new encoding of alldifferent into sat,. In Frisch, A.M., Miguel, I., eds.: Proceedings 3rd International Workshop on Modelling and Reformulating Constraint Satisfaction Problems, Springer (2004) 95–110
18. Argelich, J., Cabiscol, A., Lynce, I., Manyà, F.: 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 (2010) 46–52
19. Ansótegui, C., Manyà, F.: Mapping problems with finite-domain variables into problems with boolean variables. In: SAT 2004 - The Seventh International Conference on Theory and Applications of Satisfiability Testing, 10-13 May 2004, Vancouver, BC, Canada, Online Proceedings, Springer LNCS (2004) 1–15
20. Tamura, N., Taga, A., Kitagawa, S., Banbara, M.: Compiling finite linear csp into sat. Constraints **14**(2) (2009) 254–272
21. Eén, N., Sörensson, N.: Translating pseudo-boolean constraints into sat. Journal on Satisfiability, Boolean Modeling and Computation **2** (2006) 1–26

22. Bailleux, O., Boufkhad, Y., Roussel, O.: New encodings of pseudo-boolean constraints into CNF. In Kullmann, O., ed.: Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings. Volume 5584 of Lecture Notes in Computer Science., Springer (2009) 181–194
23. Martins, R., Manquinho, V.M., Lynce, I.: Exploiting cardinality encodings in parallel maximum satisfiability. In: IEEE 23rd International Conference on Tools with Artificial Intelligence, ICTAI 2011, Boca Raton, FL, USA, November 7-9, 2011. (2011) 313–320
24. Warners, J.P.: A linear-time transformation of linear inequalities into conjunctive normal form. Information Processing Letters **68**(2) (1998) 63–69
25. Asín, R., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E.: Cardinality networks: a theoretical and empirical study. Constraints **16**(2) (2011) 195–221
26. Ben-Haim, Y., Ivrii, A., Margalit, O., Matsliah, A.: Perfect hashing and CNF encodings of cardinality constraints. In Cimatti, A., Sebastiani, R., eds.: Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings. Volume 7317 of Lecture Notes in Computer Science., Springer (2012) 397–409
27. Gebser, M., Kaufmann, B., Schaub, T.: The conflict-driven answer set solver clasp: Progress report. In Erdem, E., Lin, F., Schaub, T., eds.: Logic Programming and Nonmonotonic Reasoning, 10th International Conference, LPNMR 2009, Potsdam, Germany, September 14-18, 2009. Proceedings. Volume 5753 of Lecture Notes in Computer Science., Springer (2009) 509–514
28. : (http://www.satcompetition.org/)
29. Brahim Hnich, Ian Miguel, I.P.G., Walsh, T.: Csplib is a library of test problems for constraint solvers. (`http://www.csplib.org/`) [Online; accessed 24-August-2012].
30. 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 Twelfth Conference on on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA, AAAI Press / The MIT Press (2000) 256–261