At-Most-One Constraints in Efficient Representations of Mutex Networks

Pavel Surynek
Faculty of Information Technology
Czech Technical University in Prague
Czech Republic
pavel.surynek@fit.cvut.cz

***Abstract*—The At-Most-One (AMO) constraint is a special case of cardinality constraint that requires at most one variable from a set of Boolean variables to be set to $TRUE$. AMO is important for modeling problems as Boolean satisfiability (SAT) from domains where decision variables represent spatial or temporal placements of some objects that cannot share the same spatial or temporal slot. The AMO constraint can be used for more efficient representation and problem solving in mutex networks consisting of pair-wise mutual exclusions forbidding pairs of Boolean variable to be simultaneously $TRUE$. An on-line method for automated detection of cliques for efficient representation of incremental mutex networks where new mutexes arrive using AMOs is presented. A comparison of SAT-based problem solving in mutex networks represented by AMO constraints using various encodings is shown.**

***Index Terms*—mutex networks, at-most-one constraint, incremental mutex, Boolean satisfiability, cardinality constraints, Boolean encodings, clique detection**

I. INTRODUCTION

We address the problem of representing the incremental mutex network efficiently using the At-Most-One (AMO) constraint [1]–[5]. The problem is addressed in the context of Boolean satisfiability (SAT) [6]–[10] where the task is to decide whether there exists a truth value assignment satisfying a given Boolean formula. Usually we assume that formula $\mathcal{F}$ is specified using the *conjunctive normal form* (CNF) [11] as a finite list of *clauses* where each clause is a (finite) disjunction of *literals* and literal is either a variable or a *negation* of a variable. Let us denote a set of Boolean decision variables on top of which $\mathcal{F}$ is expressed $Var(\mathcal{F}) = X = \{x_1, x_2, ..., x_n\}$.

A mutex network over set of variables X is defined as a set of pair wise mutual exclusions that forbid a pair of propositional variables to be simultaneously $TRUE$. Formally the mutex network of size $k \in \mathbb{N}$ denoted $\mathcal{M}_k$ is a set of clauses $\{(\neg x_{u_i} \vee \neg x_{v_i}) \mid u_i, v_i \in \{1, 2, ..., n\} \quad \forall i = 1, 2, ..., k\}$. Mutex network $\mathcal{M}_k$ can be regarded as a subset of clauses of given Boolean formula $\mathcal{F}$ (in such case all binary clauses of the formula are included in $\mathcal{M}_k$).

Mutex networks appear in many problems expressed through the means of SAT. Many difficult problems that arise in *circuit design* [12], *scheduling* [13], *classical planning* [14]–[17] or various cases of *domain dependent planning* [18], [19] can be expressed using networks of mutual exclusions. Generally mutex networks are often a product of spatial and temporal constraints between some objects. In real physical domains, objects usually cannot share the same spatial or temporal slot which directly leads to mutual exclusions of occurence of different objects in the same slot or the same object in multiple slots. Using common *direct encodings* [20] is which decision Boolean variables are used directly to represent some object relations (for example x_1^A and x_1^B represent occurrence of object A and object B in slot 1 respectively) can often give rise to complex mutex network (consisting of clauses like $\neg x_1^A \vee \neg x_1^B$ saying that objects A and B cannot appear simultaneously in slot 1).

This research has been supported by GAČR - the Czech Science Foundation, grant registration number 19-17966S.

A. Contribution

We contribute by a method for automated detection of the AMO constraint in mutex networks. We propose algorithms for detecting cliques in mutex networks that can operate in an on-line mode which means that mutexes are processed as they arrive into the mutex network. Such feature is of great interest in domains where lazy Boolean encodings are used. Several Boolean encodings of the AMO constraint are recalled. After detecting cliques in mutex network these cliques can be substituted by the AMO constraint using the encoding of user's choice.

The paper is organized as follows: first we introduce existing encodings of the AMO constraint. Then an on-line clique detection algorithms are introduced: one exact and one relaxed. Both algorithms are theoretically analyzed. Finally, we implement and evaluate the suggested AMO substitution method on a set of benchmarks including various difficult SAT instances.

II. BACKGROUND

The *At-Most-One* constraint (AMO) over Boolean variables $X_{\leq} = \{x_{i_j}\}_{j=1}^{m}$ with $i_j \in \{1, 2, ..., n\}$ for $j = 1, 2, ..., m$ denoted $_{\leq 1}\{x_{i_1}, x_{i_2}, ...x_{i_m}\}$ or $_{\leq 1}\{x_{i_j}\}_{j=1}^{m}$ requires that at most one Boolean variable from x_{i_j} variables can be set to $TRUE$. AMO is a special case of more general *cardinality constraints* [3], [4] denoted $_{\leq c}\{x_{i_1}, x_{i_2}, ...x_{i_m}\}$ with $c \in \mathbb{N}$ requiring that at most c variables from $X_{\leq}$ are assigned $TRUE$.

Cardinality constraints have wide use in problem modeling since they enable bounding various numeric values in inside

the yes/no environment of Boolean formulae. This is a significant contribution that brings SAT towards practical use in problem solving [21].

Definition 1: **(clique clustering).** Given an undirected graph $G = (V, E)$, a *clique clustering* of G is a collection of subsets of vertices, $\mathcal{C} = \{K \subseteq V\}$ such that each $K \in \mathcal{C}$ induces a complete sub-graph of G and each edge of G is covered by $\mathcal{C}$, that is, $\forall\{u, v\} \in E(\exists K \in \mathcal{C}$ such that $u, v \in K)$.

The concept of clique clustering can be naturally converted for mutex networks by substituting of indices Boolean variables from X instead of V and the set of binary clauses of $\mathcal{M}_k$ instead of E (that is, clause $(\neg x_{u_i} \vee \neg x_{v_i})$ represents edge $\{u_i, v_i\}$).

A. Encodings of the At-Most-One Constraint

The At-Most-One constraint can be expressed by various encodings often using additional auxiliary Boolean variables. The set of auxiliary variables is denoted $A_{\leq}$ in this context. The basic encoding of AMO often called *pairwise* can simply use the mutex network of size $\frac{m\cdot(m-1)}{2}$ consisting of clauses $\{\neg x_{u_i} \vee \neg x_{v_i} \mid u_i, v_i \in \{i_1, i_2, ..., i_m\} \wedge u_i < v_i\}$. Even though this encodings supports achieving *arc-consistency* [22] through *unit propagation* [23], [24] it lacks any understanding of the global structure of the constraint [25]. It has been experimentally shown that different encodings of the At-Most-One constraint significantly outperform the pairwise encoding in a number of benchmarks [1].

We illustrate the process of encoding on two AMO representations below: the *binary encoding* and the *commander encoding*.

Binary Encoding: The *binary encoding* [26] uses the idea of mapping m possible settings of exactly one variable in set $X_{\leq}$ to $TRUE$ to m possible different settings of bits in a bit vector of length $\lceil \log_2 m \rceil$. The binary encoding uses $\lceil \log_2 m \rceil$ auxiliary variables: $A_{\leq} = \{b_1, b_2, ..., b_{\lceil \log_2 m \rceil}\}$ to represent the bit vector. Whenever x_{i_j} is set to $TRUE$ the bit vector variables must be set following the binary encoding to represent value j in the bit vector. Let $\overset{j}{l} = \begin{cases} b_l & \text{if } l\text{-th bit of binary encoding of } j \text{ is } 1 \\ \neg b_l & \text{if } l\text{-th bit of binary encoding of } j \text{ is } 0 \end{cases}$

Specifically we have the following formula to represent ${}_{\leq 1}\{x_{i_j}\}_{j=1}^{m}$ using the binary encoding:

$$(\neg x_{i_1} \vee \overset{1}{1}) \wedge ...(\neg x_{i_1} \vee \overset{1}{\lceil \log_2 m \rceil}) \wedge$$
$$(\neg x_{i_2} \vee \overset{2}{1}) \wedge ...(\neg x_{i_2} \vee \overset{2}{\lceil \log_2 m \rceil}) \wedge$$
$$...$$
$$(\neg x_{i_m} \vee \overset{m}{1}) \wedge ...(\neg x_{i_m} \vee \overset{m}{\lceil \log_2 m \rceil})$$

Commander Encoding: The *commander encoding* [27] partitions $X_{\leq}$ into disjoint subsets $Y_1 = \{y_1^1, ...y_{g_1}^1\}$, $Y_2 = \{y_1^2, ..., y_{g_2}^2\}$, ... $Y_d = \{y_1^d, ..., y_{g_d}^d\}$. For each group of variables Y_i we introduce an auxiliary commander variable c_i. The interpretation is that c_i set to $TRUE$ selects a candidate from group Y_i. In other words, commander variables introduce a hierarchical structure for selecting at most one variable from the original set $X_{\leq}$: at most one commander variable can be set to $TRUE$ while it permits to select at most one variable from its group; no other variable in other groups can be set to $TRUE$. Following clauses need to be introduces to carry out the encoding:

$$(\neg c_1 \vee y_1^1 \vee y_2^1 \vee ...y_{g_1}^1) \wedge ...(\neg c_d \vee y_1^d \vee y_2^d \vee ...y_{g_d}^d)$$

In addition to this following AMO constraints need to be introduced: ${}_{\leq 1}\{\neg c_1 \vee y_1^1 \vee y_2^1 \vee ...y_{g_1}^1\}$, ${}_{\leq 1}\{\neg c_2 \vee y_1^2 \vee y_2^2 \vee ...y_{g_2}^2\}$, ..., ${}_{\leq 1}\{\neg c_d \vee y_1^d \vee y_2^d \vee ...y_{g_d}^d\}$. Finally, we need to enforce that at most one commander variable is selected by: ${}_{\leq 1}\{c_1 \vee c_2 \vee ...c_d\}$.

The number of groups partitioning $X_{\leq}$ is set to $d = \lceil \sqrt{m} \rceil$ which is a good compromise between the number of groups and their size.

III. ON-LINE CLIQUE DETECTION IN MUTEX NETWORKS

We present here techniques for detecting *cliques* (complete sub-graph) in mutex networks. Having cliques detected on top of set of mutexes one can introduce AMO constraints using some more advanced encoding instead the set of mutex clauses. Finding cliques in an undirected graph is an NP-hard problem hence certain trade-offs between the quality of discovered cliques and the complexity of method must be adopted. Despite progress in algorithms for clique detection [28], [29] simple on-line algorithms are currently missing.

We therefore present an exact exponential time/space algorithm for finding all possibly overlapping cliques in the input mutex network. Then we will show how to relax the algorithm to reduce its complexity to acceptable polynomial level while keeping its original idea.

A. Exact Algorithm

The exact algorithm for clique detection in mutex network is presented using pseudo-code as Algorithm 1. The algorithm relies on the idea of merging variables into clusters while valued meta-edges between clusters are maintained. The meta-edge between clusters corresponds to the set of edges interconnecting the two clusters. The value of meta-edge is the number of edges connecting the clusters.

When new mutexes arrive to the mutex network the algorithm updates (increments) the value of meta-edges between all pairs of clusters the mutex variables participate in (line 8 in EXACT-CLIQUES). If this value of the meta-edge reaches the size of the complete graph between the pair of cluster, the clusters are merged together to form a new cluster (the original pair of clusters is maintained). This is done in the INCREMENT-CLUSTER-LINK procedure.

Without proof let us state that the EXACT-CLIQUES algorithm is sound and complete and returns clique clustering of the input mutex network. All cliques contained in the input mutex network are detected by the algorithm, that is, if a subset of variables $X \subseteq X_{\leq}$ induces a clique of mutexes w.r.t. $\mathcal{M}_k$ then $X \in \mathcal{C}$.

To provide deeper insight into how the algorithm proceeds we summarize its complexity in the following proposition.

Algorithm 1: Exact clique detection algorithm.

1 **EXACT-CLIQUES**($\mathcal{M}_k = \{(\neg x_{u_i} \vee \neg x_{v_i}) \mid u_i, v_i \in \{1,2,...,n\}\ \forall i = 1,2,...,k\}$)
2 let $\mathcal{E}(K_v, K_u) = 0$ for any $K_u, K_v \subseteq \{1,2,...,n\}$
3 **for** *each* $i = 1,2,...,n$ **do**
4 $\mathcal{C} \leftarrow \{\{i\}\}$
5 **for** *each* $(\neg x_{u_i} \vee \neg x_{v_i}) \in \mathcal{M}_k$ **do**
6 **for** *each* $K_u \in \mathcal{C}$ *such that* $u_i \in K_u$ **do**
7 **for** *each* $K_v \in \mathcal{C}$ *such that* $v_i \in K_v$ **do**
8 INCREMENT-CLUSTER-LINK(K_u, K_v)
9 **return** $\mathcal{C}$
10 **INCREMENT-CLUSTER-LINK**(K_u,K_v)
11 $\mathcal{E}(K_u, K_v) \leftarrow \mathcal{E}(K_u, K_v) + 1$
12 **if** $\mathcal{E}(K_u, K_v) = |K_u \setminus (K_u \cap K_v)| \times |K_v \setminus (K_v \cap K_v)|$ **then**
13 **for** *each* $K \in \mathcal{C}$ *such that* $\mathcal{E}(K, K_u) > 0 \vee \mathcal{E}(K, K_v) > 0$ **do**
14 $\mathcal{E}(K, K_u \cup K_v) \leftarrow \mathcal{E}(K, K_u) + \mathcal{E}(K, K_v) - \mathcal{E}(K, K_u \cap K_v)$
15 $\mathcal{C} \leftarrow \mathcal{C} \cup \{K_u \cup K_v\}$

Proposition 1: **(EXACT-CLIQUES complexity).** The EXACT-CLIQUES algorithm for mutex network $\mathcal{M}_k$ over set of variables $X = \{x_1, x_2, ..., x_n\}$ has the worst case time complexity $\mathcal{O}(k \cdot 2^{3n})$ and the worst case space complexity $\mathcal{O}(2^{2n})$.

Proof. Let us first calculate the complexity of the procedure for incrementing the number of links between a pair of clusters K_u and K_v (INCREMENT-CLUSTER-LINK). The determining factor is updating the number of links of the newly created cluster with existing clusters (lines 13-14). This consumes time of $\mathcal{O}(2^n)$ since there is up to 2^n clusters that need to update their number of links towards the new cluster.

The dominating factor in the overall space complexity is the structure for keeping the number of links between variable clusters represented by $\mathcal{E}$. As the number of clusters is at most 2^n, the number of links connecting clusters is bounded by 2^{2n}. Hence the overall space complexity of $\mathcal{O}(2^{2n})$.

To calculate time complexity we need to observe that single variable can participate in as many as 2^n clusters. Hence line 8 in EXACT-CLIQUES where the number of links between clusters is incremented can be executed as many as 2^{2n} times per one mutex which gives altogether $k \cdot 2^{2n}$ execution across entire mutex network. Taking into account the $\mathcal{O}(2^n)$ time required by the single link increment we obtain that the algorithm needs time $\mathcal{O}(k \cdot 2^{3n})$ steps in the worst case. ■

The algorithm of such high complexity is impractical however our preliminary experiments indicate that it can be well used for detecting cliques in small mutex networks.

Observe that the EXACT-CLIQUES algorithm can operate in an *on-line* mode where new mutexes arrive piece by piece while the clique clustering is still kept up to date. This is important in many applications such as planning where the SAT model of a problem is often incrementally modified (that is new constraints including mutexes are added) so we do not need to search for cliques from scratch but only update recent changes.

We will use the algorithm as a starting point for a relaxed version which will keep the on-line functionality.

Algorithm 2: Relaxed clique detection algorithm.

1 **RELAXED-CLIQUES**($\mathcal{M}_k = \{(\neg x_{u_i} \vee \neg x_{v_i}) \mid u_i, v_i \in \{1,2,...,n\}\ \forall i = 1,2,...,k\}$)
2 let $\mathcal{E}(K_v, K_u) = 0$ for any $K_u, K_v \subseteq \{1,2,...,n\}$
3 **for** *each* $i = 1,2,...,n$ **do**
4 $\mathcal{C} \leftarrow \{\{i\}\}$
5 **for** *each* $(\neg x_{u_i} \vee \neg x_{v_i}) \in \mathcal{M}_k$ **do**
6 $K_u^* \leftarrow \text{argmax}_{K_u \in \mathcal{C} | u_i \in K_u} |K_u|$
7 $K_v^* \leftarrow \text{argmax}_{K_v \in \mathcal{C} | v_i \in K_v} |K_v|$
8 INCREMENT-CLUSTER-LINK(K_u^*, K_v^*)
9 $\mathcal{C} \leftarrow \mathcal{C} \cup \{\{u_i, v_i\}\}$
10 **return** $\mathcal{C}$

B. Relaxation of the Exact Algorithm

We relax the exact clique detection algorithm while keeping its high level structure of merging the variable clusters. We do this by restricting the set of pair of clusters for which merging attempt is made. Intuitively, as we aim on finding large cliques, it seems to be promising to focus on merging of large clusters together while smaller clusters are omitted.

The method presented using pseudo-code in Algorithm 2 attempts to merge only the largest pair of clusters. That is, for an arriving mutex $(\neg x_{u_i} \vee \neg x_{v_i})$ we identify largest cluster K_u^* containing u_i and similarly largest cluster K_v^* containing v_i (lines 6-7). The attempt to merge clusters is done only for K_u^* and K_v^*.

Proposition 2: **(RELAXED-CLIQUES complexity).** The RELAXED-CLIQUES algorithm for mutex network $\mathcal{M}_k$ over set of variables $X = \{x_1, x_2, ..., x_n\}$ has the worst case time complexity $\mathcal{O}(k^2)$ and the worst case space complexity $\mathcal{O}(k^2)$.

Proof. We first need to observe that single mutex being processed $\mathcal{M}_k$ in the main loop (lines 6-8) can give rise to at most one new cluster of variables. Hence the total number of clusters is bounded by k. The remaining calculation of the complexity relies on this bound (we assume that $n < k$). Incrementing the number of links between a pair of clusters then takes k steps since we need to update connection of the new cluster towards at most k existing clusters (lines 13-14 of INCREMENT-CLUSTER-LINK).

For each of k mutexes in input mutex network $\mathcal{M}_k$ we take the largest cluster in which its variables participate and increment the number of links between the pair of largest clusters. Assuming $\mathcal{O}(k)$ the worst case time complexity of incrementing the main loop of RELAXED-CLIQUES (lines 5-8) yields time $\mathcal{O}(k^2)$ altogether.

The worst case space complexity is determined by the need to represent number of links between clusters $\mathcal{E}$ which can be done using space $\mathcal{O}(k^2)$. ■

It is a question now if such dramatic reduction of time and space complexity through restricting the link incrementing only on largest clusters keeps ability of the algorithm to detect cliques reasonably. The example show in Figure 1 illustrates

that the restriction on largest clusters does not compromise finding important clique in the network.

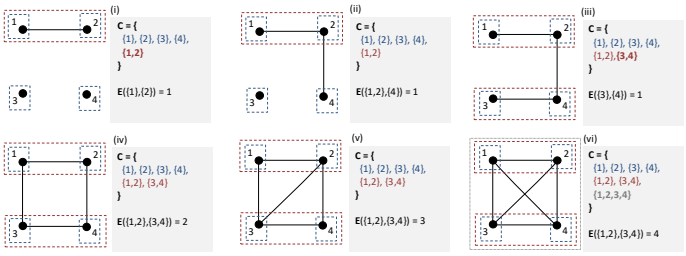

Fig. 1. The process of clique clustering construction by the RELAXED-CLIQUES algorithm. Mutexes of a network consisting of single clique on 4 variables $\{x_1, x_2, x_3, x_4\}$ are processed following the ordering: $\{1,2\}, \{2,3\}, \{3,4\}, \{1,3\}, \{2,3\}$ and $\{1,4\}$. Eventually the original clique covering all 4 variables is detected in step (vi).

An intuitive insight into how the relaxed clique clustering works suggests that it is important to add mutexes participating in a single clique in a sequence not much interrupted by additions of other mutexes. In such case, clusters have chance to grow to cover the clique and do not grow outside the clique. Practical applications suggests that this property is often the case since for example all objects related to given spatial slot that are spatially excluded are processed in a single uninterrupted block.

C. The At-Most-One Constraint Substitution in Mutex Networks

After detecting cliques in mutex network we can convert the encoding so that instead of the basic pair-wise representation of AMOs different encodings can be used. The soundness of substitution of AMO constraints relies on the property of clique clustering (proof omitted):

Proposition 3: Assume a clique clustering $\mathcal{C}$ of mutex network $\mathcal{M}_k$. Representing each clique $C \in \mathcal{C}$ using some encoding of the AMO constraint results in an equivalent instance as that represented by $\mathcal{M}_k$.

The equivalence here is defined as having identical set of satisfying assignments, that is the set of satisfying assignments of the conjunction of AMOs for cliques is the same as the set of satisfying assignment for the original set of mutexes $\mathcal{M}_k$ (auxiliary variables are ignored).

Moreover it is easy to see that if there are cliques $C' \in \mathcal{C}$ and $C \in \mathcal{C}$ such that $C \subseteq C'$ then it is sufficient to represent C' using the AMO constraint to keep the above equivalence valid.

IV. EXPERIMENTAL EVALUATION

We evaluated the suggested approach of AMO substitution in mutex networks experimentally. The approach can be used both in *eager* SAT encodings of problems where the target formula is first constructed in advance and then consulted with the SAT solver as well as in the *lazy setup* where the target formula is constructed incrementally piece by piece and the SAT solver is consulted multiple times during the process of construction [30].

A. Benchmarks and Setup

The experimental evaluation is based on the GLUCOSE 3.0 SAT solver [31] which has been used as a library linked to the testing program. The test itself is implemented in C++ [1].

The AMO substitution is implemented for all discussed encodings: **binary**, **sequential**, **product**, and **commander** encodings while the **pair-wise** is kept as the baseline encoding for reference comparison. We divided the experiments in three tests:

1) evaluation of AMO substitution in random mutex formulae and
2) evaluation of clique detection in random mutex network
3) evaluation of AMO substitution in standard SAT benchmarks consisting of difficult instances [12]

B. Comparison of Mutex Network Representations Using AMOs

In all tests we compared representation mutex network using detected AMO constraints and the base-line representation where mutexes are kept in their original form as pair-wise encoding. The test runs in three phases:

1) **Clique clustering.** This phase processes the input SAT instance in CNF that is ether generated synthetically or read from the input file. All clauses from the input representing mutexes (clauses of the form $(\neg x \vee \neg y)$) are treated as being part of mutex network $\mathcal{M}_k$ and are not declared to the SAT solver; other clauses, that is those of higher arity than 2 and those not representing mutual exclusion of $TRUE$ assignment to a pair of variables are directly declared to the SAT solver. $\mathcal{M}_k$ consisting of collected mutexes is processed by the RELAXED-CLIQUES algorithm which produces clique clustering $\mathcal{C}$.
2) **AMO encoding phase.** The clique clustering $\mathcal{C}$ is converted to actual AMO encoding so that the resulting formula is equivalent to the original one. We start from largest cliques in $\mathcal{C}$ and continue down to cliques of size 3. Each clique $C \in \mathcal{C}$ is checked if it is subsumed by any larger already processed clique. If not then C is encoded using selected AMO encoding and resulting clauses are recorded. If C is subsumed by some $C' \in \mathcal{C}$ then C is simply ignored as its meaning is already captured by C'. The remaining cliques of size 2 (simple mutual exclusions) are declared as clauses if they are not subsumed by any larger clique.
3) **SAT solving phase.** This phase corresponds to consulting the SAT solver with encoded instance. Encoded clauses are all declared to the SAT solver and the solver is started using its default setting.

In the SAT solving phase, various performance characteristics of the SAT solver were measured such as runtime. The

[1]To enable reproducibility of presented results we provide the source code and supporting experimental data on the author's website: http://users.fit.cvut.cz/surynpav/research/mutexAMO2020. The source code of presented algorithms is also available in author's Git repository: http://github.com/surynek/mutEX.

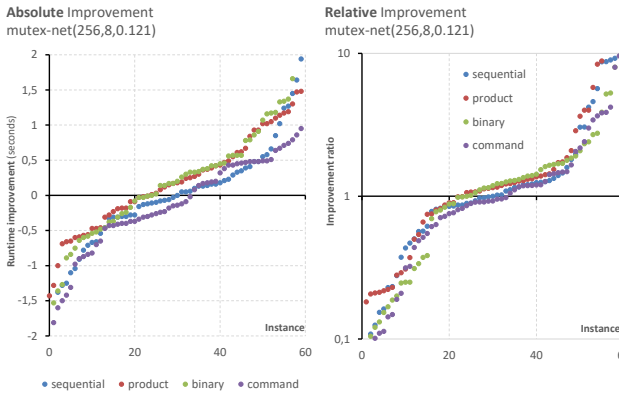

Fig. 2. Absolute and relative improvement by AMOs using various encodings in random mutex network with $N = 256$ variables, disjunctions of size $D = 8$, and probability of mutexes $p = 0.121$ (higher plot means better performance).

SAT solver has been given the time limit of 8000 seconds (approx. 2 hours 13 minutes). Preliminary experiments indicate that the runtime of the clique clustering and AMO encoding phases is negligible, hence there is no time limit of these phases. [2].

Random Mutex Formulae: A *random mutex formula* is characterized by three parameters: N, the number of variables, D, the size of disjunctive clauses, and p, the probability of a mutex. The formula is denoted mutex-net(N, D, p). The formula is constructed by declaring N Boolean variables. Then each of possible $\frac{N \cdot (N-1)}{2}$ mutexes is added with probability of p. Such a formula is trivially satisfiable by setting all variables to $FALSE$. To make the formula more interesting we divide the set variables into $\lceil \frac{N}{D} \rceil$ disjoint subsets consisting of D variables (the last group may consist of fewer variables) and a disjunction over the variables in D is added.

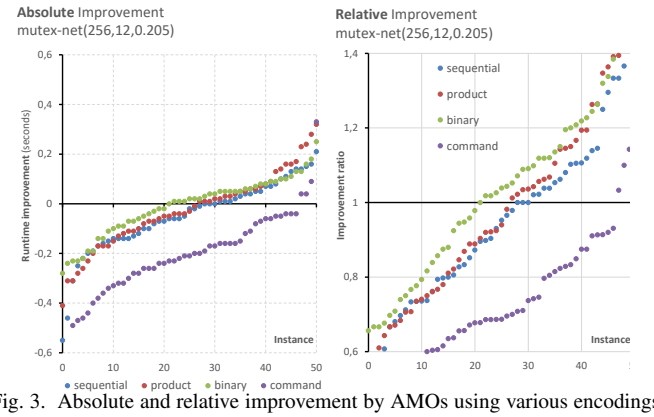

Fig. 3. Absolute and relative improvement by AMOs using various encodings in random mutex network with $N = 256$ variables, disjunctions of size $D = 12$, and probability of mutexes $p = 0.205$.

Mutex formulae tend to be relatively easy except setting of p in certain narrow analogy of phase transition region. Runtime comparison for different encodings of detected AMO constraints on mutex formulae are presented in Figures 2 and 3 showing results for mutex-net$(256, 8, 0.121)$ and mutex-net$(256, 12, 0.205)$ respectively.

The probability of mutexes p is selected to belong to the phase transition region; 60 random instances were generated for each parameter setting. The detected AMOs are encoded using all discussed encodings and compared to the baseline representation using the pair-wise encoding.

The sorted absolute and relative differences from the runtime of the base-line pair-wise encoding is shown. Generally, we cannot say there is universal improvement across all tested instances. Often the performance worsens with using the AMO constraints. The improvement is however in some cases up to 50% compared to the runtime of the pair-wise encoding. We can also observe that using the commander and sequential encodings often results in worsening of performance. On the other hand the binary encoding and the product encoding tend to improve the situation.

Comparing the two classes of random mutex formulae we can observe that AMO substitution yields better results on mutex-net$(256, 8, 0.121)$ while on mutex-net$(256,\ 12, 0.205)$ worse performance after AMO substitution occur more frequently.

It is important to note that random mutex formulae are not especially suitable for finding large cliques. We used these instances to test the potential of AMO substitution under not very promising circumstances. The size of cliques identified in this experiments is usually 3 or 4 rarely 5. Still such small discovered cliques are shown to have potential for the AMO substitution.

C. Clique Detection in Random Mutex Networks

We also evaluated the performance of the relaxed clique detection separately from the SAT solving process. We focused on the size of cliques detected by the algorithm in this test. Random mutex networks mutex-net(N, D, p) as defined in the previous section were used. The difference from the previous tests is that we take into account only the mutex clauses from mutex-net(N, D, p) while larger disjunctions (those of size D) are ignored.

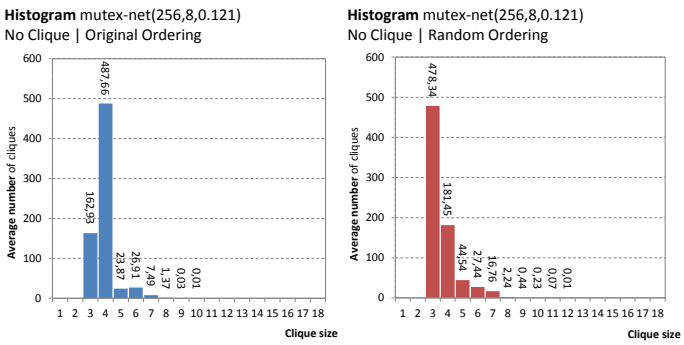

Fig. 4. A histogram showing distribution of sizes of detected cliques in mutex-net$(256, 8, 0.121)$ where no cliques are explicitely introduced.

Since the RELAXED-CLIQUES algorithm is sensitive to the ordering in which new mutexes arrive into the mutex

[2]All runtime measurements were done a system with a Ryzen 7 3GHz CPU cores and 32GB RAM running under Ubuntu Linux 19.

network, the test is divided in two cases. In the first case, mutexes arrive in the lexicographic ordering of pairs indices of mutex variables - we refer to this case as an *original ordering*. In the second case, mutexes are permuted randomly - we refer to this case as a *random ordering*. Results for mutex-net$(256, 8, 0.121)$ and mutex-net$(256, 12, 0.205)$ whose parameters are selected to belong to the aforementioned phase transition are shown in Figures 4 and 5. The figures shows histogram of clique sizes across 100 randomly generated mutex-net(N, D, p). Each of the figures shows results for the original (left part) and random ordering (right part) of the arrival of mutexes into the mutex network.

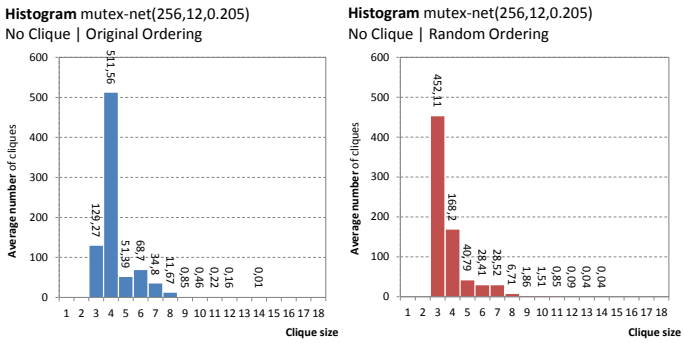

Fig. 5. A histogram showing distribution of sizes of detected cliques in mutex-net$(256, 12, 0.205)$ where no cliques are explicitly introduced.

It can be observed in both types of mutex networks that most of cliques are relatively small of sizes 3 or 4. Rarely a larger clique can be discovered - cliques of sizes up to 7 or 8 can be discovered. Moreover, the results clearly indicate that ordering of arrival of mutexes has a significant impact on what cliques are eventually discovered. Most of discovered cliques are of size 4 if the original ordering is used while most of cliques is of size 3 in the case of random ordering which can be again observed in both mutex-net$(256, 8, 0.121)$ and mutex-net$(256, 12, 0.205)$. On the other hand random ordering provides opportunity for a larger clique to grow.

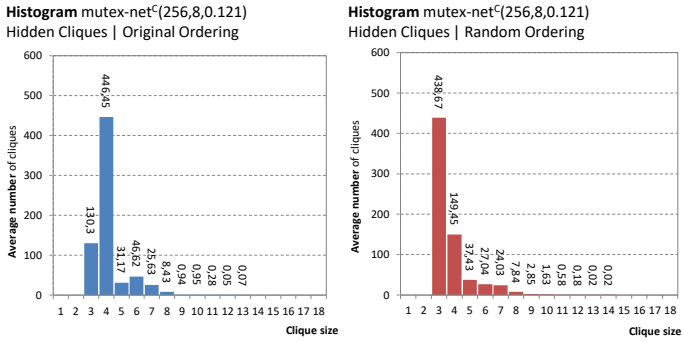

Fig. 6. A histogram showing distribution of sizes of detected cliques in mutex-net$(256, 8, 0.121)$ where hidden cliques of size 8 are present.

The explanation for this behavior is that the original (lexicographic) ordering supports the growth of clique cluster around a variable that is shared across a sub-sequence of mutexes within the input mutex sequence. Such opportunity is less likely to occur when the ordering of mutexes is completely random. The explanation of detecting larger cliques with random ordering is that clusters in such case are more evenly distributed hence the chance of merging a pair large clusters is higher.

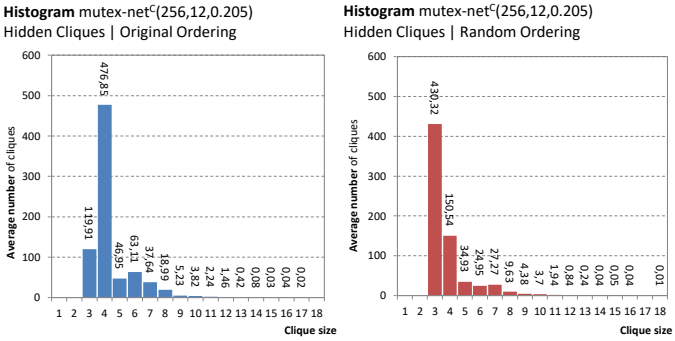

Fig. 7. A histogram showing distribution of sizes of detected cliques in mutex-net$(256, 12, 0.205)$ where hidden cliques of size 12 are present.

The next test is focused on evaluation of discovering large cliques in random mutex networks. We use mutex-net(N, D, p) as a basis but parameter D is used for generating cliques. Instead of introducing a disjunction of size D a clique of mutexes of size D is introduced. The clique can be regarded as **hidden** in the mutex network. The resulting network will be denoted mutex-net$^C(N, D, p)$. Again we use the following setups: mutex-net$^C(256, 8, 0.121)$ and mutex-net$^C(256, 12, 0.205)$ - the results are shown in Figures 6 and 7. The original ordering corresponds first to adding mutexes randomly followed by adding cliques [3]. The random ordering adds all mutexes from random phase and from cliques in a random order.

We can see in the results that clique of larger size can be detected in the networks compared to the case with no hidden cliques. However original cliques can be hardly recovered all. In mutex-net$^C(256, 8, 0.121)$ we can recover approximately 8 in 32 hidden cliques of size 8 when the original ordering is used. When the random ordering is used the chance is slightly lower. In mutex-net$^C(256, 12, 0.205)$ only 1 clique of size 12 can be recovered from 20 such cliques hidden in the network.

The explanation for the observed behavior is that mutex-net$^C(256, 8, 0.121)$ is not as densely populated by random mutexes so there is still chance that the clique cluster grows around originally hidden cliques. This contrasts to the mutex-net$^C(256, 12, 0.205)$ where two factors decrease chances to find the hidden cliques. First, these cliques are larger hence more successful steps are needed to detect them and second, the cliques are more overlaid by random mutexes. Hence harder to be detected by the relaxed clique algorithm. Despite not discovering all hidden cliques we cannot say the algorithm to be unsuccessful as it can find many smaller cliques and occasionally even larger ones.

[3] Adding cliques as first would lead to their exact discovery as by example in Figure 1. Adding random mutexes before leads to hiding the cliques in the network.

ID	**Pigeon Hole** (seconds) – **original** ordering					
	Pair Wise	*Sequential*	*Product*	*Binary*	*Command*	*Clique cluster*
hole6	**0.01**	**0.01**	**0.01**	**0.01**	**0.01**	<0.01
hole7	0.06	**0.01**	0.02	**0.01**	**0.01**	<0.01
hole8	1.49	0.04	**0.03**	0.05	0.04	0.01
hole9	16.56	0.12	0.13	0.24	**0.07**	0.01
hole10	359.13	0.85	0.51	1.05	**0.19**	0.02
hole11	3488.19	1.11	2.20	3.85	**0.77**	0.02

ID	**Quasi Group** (seconds) – **original** ordering					
	Pair Wise	*Sequential*	*Product*	*Binary*	*Command*	*Clique cluster*
qg1-8	0.34	0.27	0.27	0.21	**0.11**	<0.01
qg2-8	0.53	1.26	0.20	0.53	**0.18**	<0.01
qg3-9	2.25	**1.66**	2.39	2.01	2.46	0.02
qg5-13	4.73	3.78	**2.45**	2.66	2.58	0.04
qg6-12	0.85	1.25	0.67	0.65	**0.61**	0.04
qg7-13	0.13	0.18	0.18	0.12	**0.08**	0.01

ID	**Pigeon Hole** (seconds) – **random** ordering					
	Pair Wise	*Sequential*	*Product*	*Binary*	*Command*	*Clique cluster*
hole6	**0.01**	**0.01**	**0.01**	**0.01**	**0.01**	<0.01
hole7	**0.06**	0.16	0.14	0.11	0.15	<0.01
hole8	1.49	0.91	**0.57**	1.28	**0.11**	0.01
hole9	16.56	3.97	5.27	**1.35**	1.47	0.01
hole10	359.13	424.90	300.55	596.19	**9.07**	0.02
hole11	3488.19	4748.95	4238.59	4600.02	**260.59**	0.02

ID	**Quasi Group** (seconds) – **random** ordering					
	Pair Wise	*Sequential*	*Product*	*Binary*	*Command*	*Clique cluster*
qg1-8	**0.34**	1.52	1.42	**0.34**	1.85	<0.01
qg2-8	**0.53**	1.65	1.06	1.09	0.99	<0.01
qg3-9	2.25	0.18	**0.14**	0.15	0.20	0.02
qg5-13	4.73	7.20	5.02	**4.21**	8.32	0.04
qg6-12	0.85	1.08	**0.81**	1.81	1.83	0.04
qg7-13	0.13	0.08	0.08	**0.04**	0.17	0.01

Fig. 8. Runtime results for instances encoding the *Pigeon hole* priciple and *Quasi group* completion [32].

ID	**CHNL** (seconds) – **original** ordering					
	Pair wise	*Sequential*	*Product*	*Binary*	*Command*	*Clique cluster*
chnl10-11	515.24	0.62	0.56	3.51	**0.26**	0.01
chnl10-12	684.66	1.26	0.51	1.27	**0.36**	0.01
chnl10-13	945.08	0.63	0.73	1.28	**0.31**	0.02
chnl11-12	3149.47	1.73	0.78	8.91	**0.77**	0.02
chnl11-13	2589.76	2.36	0.97	6.33	**1.16**	0.02
chnl11-20	>8000.00	**2.76**	4.05	10.82	5.59	0.03

ID	**S3** (seconds) – **original** ordering					
	Pair wise	*Sequential*	*Product*	*Binary*	*Command*	*Clique cluster*
s3-3-3-1	**0.04**	0.08	0.34	0.11	0.43	0.01
s3-3-3-3	0.49	0.35	0.33	0.29	**0.09**	0.02
s3-3-3-4	0.92	0.17	**0.07**	0.13	0.37	0.01
s3-3-3-8	0.37	**0.06**	0.11	0.31	0.48	0.02
s3-3-3-10	0.58	0.37	0.54	**0.25**	0.81	0.03

ID	**CHNL** (seconds) – **random** ordering					
	Pair wise	*Sequential*	*Product*	*Binary*	*Command*	*Clique cluster*
chnl10-11	515.24	244.57	238.54	175.18	**23.08**	0.01
chnl10-12	684.66	61.60	126.72	226.43	**17.82**	0.01
chnl10-13	945.08	803.63	1083.69	888.48	**153.13**	0.02
chnl11-12	3149.47	2092.57	3648.75	**1632.64**	3801.56	0.02
chnl11-13	2589.76	>8000.00	5222.13	>8000.00	**319.72**	0.02
chnl11-20	>8000.00	**>8000.00**	>8000.00	7893.84	**1834.13**	0.03

ID	**S3** (seconds) – **random** ordering					
	Pair wise	*Sequential*	*Product*	*Binary*	*Command*	*Clique cluster*
s3-3-3-1	**0.04**	0.29	0.19	0.17	0.25	0.01
s3-3-3-3	0.49	0.57	0.30	**0.08**	0.31	0.02
s3-3-3-4	0.92	**0.44**	0.46	**0.44**	0.65	0.01
s3-3-3-8	0.37	**0.20**	0.87	0.23	0.27	0.02
s3-3-3-10	0.58	**0.25**	0.87	0.38	0.26	0.03

Fig. 9. Runtime results for instances encoding the *Channel routing* and the *Global routing* instnaces [12].

D. Classical Benchmarks

The third test is focused on SAT solving of hard instances with relatively large mutex cliques hidden inside. The aim of this experiment is to verify if the RELAXED-CLIQUES algorithm is able to detect large enough mutex cliques so that their substitution by the AMO constraints results in a significant performance gain of the SAT solving phase.

We used standard benchmarks encoding the *pigeon hole* principle (denoted `hole`) where the question is whether $K+1$ pigeons can be placed in K holes so that no two of them share a hole. This problem is known to be difficult for SAT solvers when the direct encoding is used [25]. In the direct encoding, there are variables x_i^j encoding that i-th pigeon is placed in the j-th hole and a mutex network is introduced on top of these variables. Similarly various circuit routing problems are known to define difficult SAT instances containing cliques in their mutex networks [12] (denoted `chnl` and `S3`). Circuit routing problems often encode sub-problems similar to the pigeon hole principle. Finally, *Quasi Group* (`qg`) instances encode construction of Latin squares [32].

Runtime results are presented in Tables 8 and 9. We test the original ordering of mutexes and the random ordering. The former directly corresponds to the ordering of clauses in the input instance while the latter takes random permutation of mutex clauses in the input instance and performs relaxed clique detection with respect to this random permutation.

It can be observed for the original ordering that significant performance improvement is achieved for `hole` and `chnl` instances where the base-line pair-wise encoding often exceeds runtime of 1000 seconds while all encodings of the AMO constraint lead to runtimes in terms of seconds. The most significant improvement is achieved by using the commander encoding. This is quite surprising as in random mutex formulae the commander encoding has the worst performance. We need however to take into account that the size of cliques in `hole` and `chnl` instances is much larger (more than 10 variables) than in random mutex networks.

In `qg` and `S3` instances, the performance gain with the original ordering is less significant however using AMO substitution generally leads to better performance. The best runtime is almost in all cases achieved by some AMO encoding other than the pair-wise. If we compare individual encodings then the commander encoding and the binary encoding seem to provide the most consistent results. This is more in line with observations for random mutex networks where the binary encoding performs as best.

The results for random ordering of mutexes show generally worse performance of AMO substitution. The random ordering often leads to finding other cliques than those originally encoded in the input instance. Although the original clique covering is not detected, still relatively large cliques can be discovered. The performance gain, despite being less impressive than for the original ordering, can still be worthwhile.

We attribute the relatively good performance of AMO substitution in these classical benchmarks to two factors. First, the instances are difficult and hence there is room to improve the runtime (it is usually not good to use AMO substitution in quickly solvable instances as in such cases the overhead of clique clustering could play a role). Second, cliques in these instances are relatively large which gives chance the AMO encodings to significantly differ from the pair-wise encoding.

V. RELATED WORK

Currently there seems to be a gap between works dealing with encodings of cardinality constraints and their automated detection. The notable exceptions are [33] and [34] where methods for automated rediscovering previously encoded AMOs using different encodings is presented. The difference from our work is that we are trying to detect AMOs in online mode and do not assume explicit presence of the AMO constraints in the encoding - even partial presence is valuable.

Various works deal with efficient encodings of cardinality constraints and specially at-most-one constraints [2]–[4]. The common effort is to find compact encoding (small size) that provides good support of unit propagation. As mentioned in [20] good propagation is often supported in direct encodings while small size is supported by logarithmic (log-space) encodings. Both factors are represented in our selection of AMO encodings.

Special focus on different encodings of the AMO constraint is given in [1]. More encodings such as the *ladder encoding* and the *bimander encoding* are discussed and evaluated in this work. The difference from our work is that AMO constraints

are identified in the mutex network manually. Automated detection of cliques in mutex network is done in [25] where a greedy algorithm is presented. The limitation of the greedy algorithm is that it is applicable in unsatisfiable case only where it can detect unsatisfiability by counting arguments without solving the formula. In the satisfiable case however, the method is not able to infer any new information.

VI. Conclusion

We presented a method for on-line automated detection of At-Most-One constraints in mutex networks. Our AMO substitution method consists of a clique detection algorithm that is based on growing clusters that represent cliques. Any time a new mutex arrives, clique clusters are attempted to merge together to form a larger cluster, that is, a larger clique. The second major part of the AMO substitution method is encoding of detected cliques in mutex network as AMO constraints using one of the existent encodings.

We implemented the proposed method and performed experimental evaluation which indicates that even in random mutex networks containing small cliques, using more advanced encodings of the AMO constraint for automatically detected cliques has a potential to improve solving runtime. In hard instances containing large mutex cliques the method brings significant improvement in orders of magnitude. Moreover the clique detection and AMO encoding has negligible overhead according to our tests. Additional tests show that our method is able to find relatively large cliques even if the ordering of arriving mutexes is completely random (that is, mutexes forming a signle clique do not arrive together).

Future work include investigation of generalized mutex networks in which not only mutual exclusion between Boolean variables is considered but also mutual exclusion between literals. Hence any binary clause in such view will be treated as a mutex and included in the mutex network. While at the level of clique detection and AMO encoding the approach will not differ significantly. Different performance results may be expected.

References

[1] V. Nguyen and S. T. Mai, "A new method to encode the at-most-one constraint into SAT," in *Proceedings of the Sixth International Symposium on Information and Communication Technology, Hue City, Vietnam, December 3-4, 2015*. ACM, 2015, pp. 46–53.

[2] J. Silva and I. Lynce, "Towards robust CNF encodings of cardinality constraints," in *CP*, 2007, pp. 483–497.

[3] O. Bailleux and Y. Boufkhad, "Efficient CNF encoding of boolean cardinality constraints," in *CP*, 2003, pp. 108–122.

[4] C. Sinz, "Towards an optimal CNF encoding of boolean cardinality constraints," in *CP*, 2005.

[5] P. Barahona, S. Hölldobler, and V. Nguyen, "Efficient sat-encoding of linear CSP constraints," in *Proceedings of ISAIM 2014*, 2014.

[6] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*. ACM, 1971, pp. 151–158.

[7] I. P. Gent and T. Walsh, "The satisfiability constraint gap," *Artif. Intell.*, vol. 81, no. 1-2, pp. 59–80, 1996.

[8] T. Walsh, "SAT vs CSP: a commentary," *CoRR*, vol. abs/1910.00128, 2019. [Online]. Available: http://arxiv.org/abs/1910.00128

[9] ——, "SAT v CSP," in *CP 2000*, ser. LNCS, vol. 1894. Springer, 2000, pp. 441–456.

[10] A. Biere, A. Biere, M. Heule, H. van Maaren, and T. Walsh, *Handbook of Satisfiability*. IOS Press, 2009.

[11] G. S. Tseitin, "On the complexity of derivation in propositional calculus," *Structures in Constructive Mathematics and Mathematical Logic*, pp. 115–125, 1968.

[12] F. A. Aloul, A. Ramani, I. L. Markov, and K. A. Sakallah, "Solving difficult instances of boolean satisfiability in the presence of symmetry," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 22, no. 9, pp. 1117–1137, 2003.

[13] F. A. Aloul, S. Z. H. Zahidi, A. Al-Farra, B. Al-Roh, and B. Al-Rawi, "Solving the employee timetabling problem using advanced SAT & ILP techniques," *JCP*, vol. 8, no. 4, pp. 851–858, 2013.

[14] H. A. Kautz and B. Selman, "Unifying sat-based and graph-based planning," in *Proceedings of IJCAI 1999*, 1999, pp. 318–325.

[15] ——, "Pushing the envelope: Planning, propositional logic and stochastic search," in *Proceedings of AAAI 1996*, 1996, pp. 1194–1201.

[16] J. Rintanen, "Engineering efficient planners with SAT," in *Proceedings of ECAI 2012*, vol. 242. IOS Press, 2012, pp. 684–689.

[17] N. C. Froleyks, T. Balyo, and D. Schreiber, "PASAR - planning as satisfiability with abstraction refinement," in *Proceedings of SOCS 2019*. AAAI Press, 2019, pp. 70–78.

[18] P. Surynek, "A sat-based approach to cooperative path-finding using all-different constraints," in *Proceedings of the Fifth Annual Symposium on Combinatorial Search, SOCS 2012*. AAAI Press, 2012.

[19] B. Pandey and J. Rintanen, "Planning for partial observability by SAT and graph constraints," in *Proceedings of ICAPS 2018*. AAAI Press, 2018, pp. 190–198.

[20] J. Petke, *Bridging Constraint Satisfaction and Boolean Satisfiability*, ser. Artificial Intelligence: Foundations, Theory, and Algorithms. Springer, 2015.

[21] J. Marques-Silva, "Practical applications of boolean satisfiability," in *2008 9th International Workshop on Discrete Event Systems*, 2008, pp. 74–80.

[22] A. K. Mackworth, "Consistency in networks of relations," *Artif. Intell.*, vol. 8, no. 1, pp. 99–118, 1977.

[23] W. F. Dowling and J. H. Gallier, "Linear-time algorithms for testing the satisfiability of propositional horn formulae." *J. Log. Program.*, vol. 1, no. 3, pp. 267–284, 1984.

[24] I. P. Gent, "Arc consistency in SAT," in *Proceedings of ECAI 2002*. IOS Press, 2002, pp. 121–125.

[25] P. Surynek, "Solving difficult SAT instances using greedy clique decomposition," in *Proceedings of SARA 2007*, ser. LNCS, vol. 4612. Springer, 2007, pp. 359–374.

[26] A. M. Frisch, T. J. Peugniez, A. J. Doggett, and P. Nightingale, "Solving non-boolean satisfiability problems with stochastic local search: A comparison of encodings," *J. Autom. Reasoning*, vol. 35, no. 1-3, pp. 143–179, 2005.

[27] W. Klieber and G. Kwon, "Efficient cnf encoding for selecting 1 from n objects," in *Proceedings of the Fourth Workshop on Constraint in Formal Verification (CFV)*, 2007.

[28] S. Pang, C. Chen, and T. Wei, "A realtime clique detection algorithm: Time-based incremental label propagation," in *2009 Third International Symposium on Intelligent Information Technology Application*, vol. 3, 2009, pp. 459–462.

[29] D. Duan, Y. Li, R. Li, and Z. Lu, "Incremental k-clique clustering in dynamic social networks," *Artif. Intell. Rev.*, vol. 38, no. 2, pp. 129–147, 2012.

[30] D. Kroening and O. Strichman, *Decision Procedures - An Algorithmic Point of View, Second Edition*, ser. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2016.

[31] G. Audemard, J. Lagniez, and L. Simon, "Improving glucose for incremental SAT solving with assumptions: Application to MUS extraction," in *SAT*, 2013, pp. 309–317.

[32] C. Ansótegui, A. del Val, I. Dotú, C. Fernández, and F. Manyà, "Modeling choices in quasigroup completion: SAT vs. CSP," in *Proceedings of AAAI 2004*. AAAI Press / The MIT Press, 2004, pp. 137–142.

[33] A. Biere, D. L. Berre, E. Lonca, and N. Manthey, "Detecting cardinality constraints in CNF," in *Proceedings of SAT 2014*, ser. LNCS, vol. 8561. Springer, 2014, pp. 285–301.

[34] C. Ansótegui, M. Bofill, J. Coll, N. Dang, J. L. Esteban, I. Miguel, P. Nightingale, A. Z. Salamon, J. Suy, and M. Villaret, "Automatic detection of at-most-one and exactly-one relations for improved SAT encodings of pseudo-boolean constraints," in *CP 2019*, ser. LNCS, vol. 11802. Springer, 2019, pp. 20–36.