Enhancing clause learning by symmetry in SAT solvers

Belaïd Benhamou, Tarek Nabhani, Richard Ostrowski and Mohamed Réda Saiïdi*

*Université de Provence
Laboratoire des Sciences de l’Information et des Systèmes (LSIS)
Centre de Mathématiques et d’Informatique.
39, rue Joliot Curie - 13453 Marseille cedex 13, France.
Emails: {benhamou; nabhani; ostrowski; saidi}@cmi.univ-mrs.fr

Abstract—The satisfiability problem (SAT) is shown to be the first decision NP-complete problem. It is central in complexity theory. A CNF formula usually contains an interesting number of symmetries. That is, the formula remains invariant under some variable permutations. Such permutations are the symmetries of the formula, their elimination can lead to make a short proof for a satisfiability proof procedure. On other hand, many improvements had been done in SAT solving. Conflict-Driven Clause Learning (CDCL) SAT solvers are now able to solve great size and industrial SAT instances efficiently. The main theoretical key behind these modern solvers is, they use lazy data structures, a restart policy and perform clause learning at each fail end point in the search tree. Although symmetry and clause learning are shown to be powerful principles for SAT solving, but their combination, as far as we now, is not investigated. In this paper, we will show how symmetry can be used to improve clause learning in CDCL SAT solvers. We implemented the symmetry clause learning approach on the MiniSat solver and experimented it on several SAT instances. We compared both MiniSat with and without symmetry and the results obtained are very promising and show that clause learning by symmetry is profitable for CDCL SAT solvers.

I. INTRODUCTION

Krishnamurthy in [1] introduced the symmetry principle in propositional calculus and showed that some tricky formulas can have short proofs when augmenting the resolution proof system by the symmetry rule. Symmetries for Boolean constraints are studied in depth in [2], [3], [4]. The authors showed how to detect them and proved that their exploitation makes a considerable improvement for several automated deduction algorithms. Since that, many research works on symmetry appeared. For instance, the static approach used by James Crawford et al. in [5] for propositional logic theories consists in adding constraints expressing the global symmetry of the initial problem. This technique has been improved in [6] and extended to 0-1 Integer Logic Programming in [7]. Although symmetry is introduced in propositional logic, but in the last years, it had been more investigated in constraint programming. The notion of interchangeability in Constraint Satisfaction Problems (CSPs) is introduced in [8] and symmetry for CSPs is studied earlier in [9], [10].

Since a great number of constraints could be added in the static approach, in CSPs, some researchers proposed to add the constraints during the search. In [11], [12], [13], authors post some conditional constraints which remove the symmetric of the partial interpretation in case of backtracking. In [14], [15], [16], [17], authors proposed to use each sub-tree as a nogood to avoid exploration of some symmetric interpretations and the group equivalence tree conceptual for value symmetry elimination is introduced in [18]. After that, Walsh in [19] studied various new propagators to break various symmetries among them the one acting simultaneously on both variables and values.

The satisfiability problem is generic, several problems in other field can be reduced to the satisfiability checking. For example, automatic deduction, configuration, planning, scheduling, etc. Several symmetry elimination methods for the satisfiability problem are introduced. Among them the static approaches [5], [6], [7] that eliminate the global symmetry of the initial problem and the dynamic approaches [2], [3], [4] that detect and break the local symmetry at each node of the search tree during the search. Both approaches had been shown to be profitable for SAT solving.

On other hand, considerable improvements are made these last years on SAT solvers. Modern SAT solvers use some sophisticated data structures, implement clause learning [20], [21], perform non-chronological backtrack [20], [22] and restarts [23], [24], [25]. It is shown that clause learning is a powerful notion [26], [27], [28], [29] that improves dramatically the efficiency of solvers. Symmetry had been shown to be useful for SAT solvers but never used in clause learning.

In this paper, we present a new approach for clause learning that uses symmetry. This method consists first in detecting all the global symmetries of the considered SAT instance, then use them when an assertive clause is detected during the search to deduce all its symmetrical assertive clauses that we will add to the clause base. Learning by symmetry is different from global symmetry breaking approaches. These are static methods which add in a preprocessing phase symmetry breaking predicates to eliminate symmetrical interpretations. The symmetry clause learning approach that we propose here, is dynamic. It generates during the search the symmetrical clauses of the current asserting clause then adds them to the clause base to avoid exploring isomorphic search sub-spaces. The advantage of our approach is that it does not eliminate
symmetrical models as in symmetry breaking methods, but only avoid exploring search sub-spaces corresponding to the symmetrical no-goods of the current conflicted partial interpretation.

The rest of the paper is organized as follows: Section 2 gives some background on the satisfiability problem and permutations. Section 3 defines symmetry and gives the theoretical symmetry results that we use for clauses learning to provide a new learning scheme. The fourth section describes how learning by symmetry is exploited in a CDCL Solver. We implemented the symmetry learning scheme on the modern SAT solver MiniSat and evaluated the resulting method in the sixth section where several SAT instances are tested and where a comparison of MiniSat with an without symmetry is given. Finally, we conclude the work in Section 7 and give some perspectives.

II. BACKGROUND AND PRELIMINARIES

A. Propositional logic

We will assume that the reader is familiar with the propositional calculus. We give here, a short description. Let $\mathbb{V}$ be the set of propositional variables. Propositional variables will be distinguished from literals, which are propositional variables with an assigned parity 1 or 0 that means True or False, respectively. For a propositional variable $\ell$, there are two literals: $\ell$ the positive literal and $\neg \ell$ the negative one.

A clause is a disjunction of literals $\ell_1 \lor \ell_2 \lor \ldots \lor \ell_n$. A formula $\mathcal{F}$ is in the conjunctive normal form (CNF) iff it is a conjunction of clauses.

A truth assignment (or an interpretation) of CNF $\mathcal{F}$ is a mapping $I$ defined from the set of variables of $\mathcal{F}$ into the set $\{\text{True}, \text{False}\}$. Also, we can consider $I$ as the set of the interpreted literals. The value of a clause $\ell_1 \lor \ell_2 \lor \ldots \lor \ell_n$ in $I$ is True, if the value True is assigned to at least one of its literals in $I$, it is False otherwise. By convention, we define the value of the empty clause ($n=0$) to be False. The value $I[\mathcal{F}]$ is True if the value of each clause of $\mathcal{F}$ is True, otherwise it is False. We say that a CNF $\mathcal{F}$ is satisfiable if there exists some truth assignments $I$ that assign the value True to $\mathcal{F}$, otherwise it is unsatisfiable. In the first case $I$ is called a model of $\mathcal{F}$. A formula $\mathcal{G}$ is a logical consequence of a formula $\mathcal{F}$ if $\mathcal{F} \vdash \mathcal{G}$. If an interpretation $I$ is defined only on a subset of variables of $\mathcal{F}$, then it is called a partial interpretation, it is called a no-good when $I[\mathcal{F}] = \text{False}$. Usually, SAT solvers handle a partial interpretation $I$ that is made by a subset of decision literals $D$ and a subset of propagated literals $P$ by unit resolution ($I = D \cup P$). More precisely a partial interpretation is a product $I = \prod_{x_i \in D} (x_i, y_{i,1}, y_{i,2}, \ldots, y_{i,k_i})$ formed by $m$ decision literals $x_i$ and all their ordered propagated literals $y_{i,1}, y_{i,2}, \ldots, y_{i,k_i}$. The assignment level of a decision literal $x_i$ in an interpretation $I$ (denoted by $\text{level}(x_i)$) is its assignment ordering $i$ in $I$, and all its propagated literals $y_{i,1}, y_{i,2}, \ldots, y_{i,k_i}$ have the same assignment level $i$.

B. Permutations

Let $\Omega = \{1, 2, \ldots, N\}$ for some integer $N$, where each integer might represent a literal of a CNF formula $\mathcal{F}$. A permutation of $\Omega$ is a bijective mapping $\sigma$ from $\Omega$ to $\Omega$ that is usually represented as a product of cycles of permutations. We denote by $\text{Perm}(\Omega)$ the set of all permutations of $\Omega$ and $\circ$ the composition of the permutation of $\text{Perm}(\Omega)$. The pair ($\text{Perm}(\Omega), \circ$) forms the permutation group of $\Omega$. That is, $\circ$ is closed and associative, the inverse of a permutation is a permutation and the identity permutation is a neutral element. A pair ($T, \circ$) forms a sub-group of ($\Omega, \circ$) iff $T$ is a subset of $\Omega$ and forms a group under the operation $\circ$.

The orbit $\omega_{\text{Perm}(\Omega)}$ of an element $\omega$ of $\Omega$ on which the group $\text{Perm}(\Omega)$ acts is $\omega_{\text{Perm}(\Omega)} = \{\omega^\sigma : \omega^\sigma = \sigma(\omega), \sigma \in \text{Perm}(\Omega)\}$.

A generating set of the group $\text{Perm}(\Omega)$ is a subset $\text{Gen}(\Omega)$ of $\text{Perm}(\Omega)$ such that each element of $\text{Perm}(\Omega)$ can be written as a composition of elements of $\text{Gen}(\Omega)$. We write $\text{Perm}(\Omega) = \langle \text{Gen}(\Omega) \rangle$. An element of $\text{Gen}(\Omega)$ is called a generator. The orbit of $\omega \in \Omega$ can be computed by using only the set of generators $\text{Gen}(\Omega)$.

III. SYMMETRY AND CLAUSE LEARNING

A. Symmetry

We recall in the following the definition of symmetry which is given in [2], [3].

Definition 1: Let $\mathcal{F}$ be a propositional formula given in CNF and $L_\mathcal{F}$ its complete set of literals. A symmetry of $\mathcal{F}$ is a permutation $\sigma$ defined on $L_\mathcal{F}$ such that the following conditions hold:

1) $\forall \ell \in L_\mathcal{F}, \sigma(\neg \ell) = \neg \sigma(\ell)$,
2) $\sigma(\mathcal{F}) = \mathcal{F}$

In other words, a symmetry of a formula is a literal permutation that leaves the formula invariant. If we denote by $\text{Perm}(L_\mathcal{F})$ the group of permutations of $L_\mathcal{F}$ and by $\text{Sym}(L_\mathcal{F}) \subset \text{Perm}(L_\mathcal{F})$ the subset of permutations of $L_\mathcal{F}$ that are symmetries of $\mathcal{F}$, then $\text{Sym}(L_\mathcal{F})$ is trivially a subgroup of $\text{Perm}(L_\mathcal{F})$.

Definition 2: Let $\mathcal{F}$ be a formula, the orbit of a literal $\ell \in L_\mathcal{F}$ on which the group of symmetries $\text{Sym}(L_\mathcal{F})$ acts is $\mathcal{O}(L_\mathcal{F}) = \{\sigma(\ell) : \sigma \in \text{Sym}(L_\mathcal{F})\}$.

Example 1: Let $\mathcal{F}$ be the following set of clauses: $\mathcal{F} = \{a \lor b \lor c, \neg a \lor b, \neg b \lor c, \neg c \lor a, \neg a \lor \neg b \lor \neg c\}$ and $\sigma_1$ and $\sigma_2$ two permutations defined on the complete set $L_\mathcal{F}$ of literals occurring in $\mathcal{F}$ as follows:

$\sigma_1(a, b, c) = (\neg a, \neg b, \neg c)$

$\sigma_2(a, b, c) = (a, \neg a, b)$

Both $\sigma_1$ and $\sigma_2$ are two symmetries of $\mathcal{F}$, since $\sigma_1(\mathcal{F}) = \mathcal{F} = \sigma_2(\mathcal{F})$. The orbit of the literal $a$ is $a^{\text{Sym}(L_\mathcal{F})} = \{a, b, c, \neg a, \neg b, \neg c\}$. We can see that all the literals are in the same orbit. Thus, they are all symmetrical.

The set of literals $L_\mathcal{F}$ contains each literal and its negation.

---

1The symmetry learning scheme is generic, it can be used by all solver that uses clause learning.

2The set of literals $L_\mathcal{F}$ contains each literal and its negation.
A main property of a symmetry $\sigma \in \text{Sym}(L_\mathcal{F})$ of formula $\mathcal{F}$, is that it conserves its set of models. Formally Benhamou et al [2], [3] introduced the following property

**Proposition 1:** Given a CNF formula $\mathcal{F}$, and a symmetry $\sigma \in \text{Sym}(L_\mathcal{F})$, if $I$ is a model of $\mathcal{F}$, then $\sigma(I)$ is a model of $\mathcal{F}$.

Also, a symmetry $\sigma$ transforms each no-good $I$ of $\mathcal{F}$ to a no-good $\sigma(I)$.

1) **Symmetry detection:** We used Bliss [31] to detect the symmetries of CNF $\mathcal{F}$. Bliss is a tool for computing the automorphism group of a graph. It is shown in [5], [6], [7] that each CNF formula $\mathcal{F}$ can be represented by a graph $G_\mathcal{F}$ that is built as follows:

- Each Boolean variable is represented by two vertices (literal vertices) in $G_\mathcal{F}$: the positive literal and its negation. These two vertices are connected by an edge in the graph $G_\mathcal{F}$.
- Each non binary clause is represented by a vertex (a clause vertex). An edge connects this vertex to each vertex representing a literal of a clause.
- Each binary clause is represented by an edge connecting the vertices representing its two literals. We do not need to add vertices for binary clauses.

An important property of the graph $G_\mathcal{F}$ is that it preserves the group of symmetries of $\mathcal{F}$. That is, the symmetry group $\text{Sym}(L_\mathcal{F})$ of the formula $\mathcal{F}$ is identical to the automorphism group $\text{Aut}(G_\mathcal{F})$ of its graph representation $G_\mathcal{F}$. For this, we use Bliss on $G_\mathcal{F}$ to detect the symmetry group $\text{Sym}(L_\mathcal{F})$ of $\mathcal{F}$. Bliss returns a set of generators $\text{Gen}(\text{Aut}(G_\mathcal{F}))$ of the automorphism group from which we can deduce all the symmetries of $\mathcal{F}$. Bliss offers the possibility to color the vertices of the graph such that, a vertex is allowed to be permuted with another vertex if they have the same color. This restricts the permutations to the nodes of the same color. Two colors are used in $G_\mathcal{F}$, one for the vertices corresponding to the clauses of $\mathcal{F}$ and the other color for the vertices representing the literals of $L_\mathcal{F}$. This allows to distinguish the clause vertices from the literal vertices, then prevent the generation of symmetries between clauses and literals. The source code of Bliss can be found at (http://www.tcs.hut.fi/Software/bliss/index.html).

![Fig. 1. The graph $G_\mathcal{F}$ corresponding to $\mathcal{F}$](http://www.tcs.hut.fi/Software/bliss/index.html)

For example consider the formula $\mathcal{F}$ given in Example 1. Its associated graph $G_\mathcal{F}$ is given in Figure 1, where the clause vertices are represented by boxes and the literal vertices by ellipses. The automorphism group of $G_\mathcal{F}$ is identical to the symmetry group of $\mathcal{F}$. We can note that the two symmetries $\sigma_1$ and $\sigma_2$ of the formula $\mathcal{F}$ of Example 1 can be seen as automorphisms of the corresponding graph $G_\mathcal{F}$.

**B. Learning by symmetry (SLS)**

The main key of SAT solvers is the Unit resolution. Given a formula $\mathcal{F}$, we write $\mathcal{F} \models \mathcal{I}$ to mean that the literal $\ell$ can be derived from $\mathcal{F}$ by using Unit resolution. A CNF $\mathcal{F}$ is U-inconsistent, iff, $\mathcal{F} \models \mathcal{I}$ False, otherwise $\mathcal{F}$ is U-consistent. Now, we summarize the definitions of an asserting clause, an asserted literal and the assertion level of a clause.

**Definition 3:** Let $\mathcal{F}$ be a CNF, $c = \ell_1 \lor \ell_2 \lor \cdots \lor \ell_k \lor x$ a clause of $\mathcal{F}$, and $I$ a partial interpretation of $\mathcal{F}$ with $m$ decision literals. The clause $c$ is an asserting clause iff $I[c] = \text{False}$, $\text{level}(x) = m$ and $\forall i \in \{1, \ldots, k\}, \text{level}(\ell_i) < m$. The literal $x$ is the asserted literal of $c$ and the assertion level of $c$ is the highest level of its other literals (the asserted literal is not included).

We introduce the following lemma that we shall use to prove our results on symmetrical asserting clauses.

**Lemma 1:** Let $\mathcal{F}$ be a CNF, and $\sigma$ a symmetry of $\mathcal{F}$. If $I=(x_1, y_1, y_2, \ldots, y_n)$ is a partial interpretation formed by the unique decision literal $x$ and all its ordered propagated literals $y_1, y_2, \ldots, y_n$, then $\sigma(I)=\{\sigma(x_1), \sigma(y_1), \sigma(y_2), \ldots, \sigma(y_n)\}$, $\text{level}(x)$ in $I$ is identical to $\text{level}(\sigma(x))$ in $\sigma(I)$, and $\forall i \in \{1, \ldots, n\}, \text{level}(y_i)$ in $I$ is identical to $\text{level}(\sigma(y_i))$ in $\sigma(I)$.

**Proof:** To prove this, we have to prove that, if $\mathcal{F} \models x \lor y_i$, then we have $\mathcal{F} \models \sigma(x) \lor \sigma(y_i)$ for all $i \in \{1, \ldots, n\}$. We will prove that by induction on $i$. For $i = 1$ we need to prove that if $\mathcal{F} \models x \lor y_1$, then $\mathcal{F} \models \sigma(x) \lor \sigma(y_1)$. We have $\mathcal{F} \models x \lor y_1$ by the hypothesis. This means, there exists a clause $c \in \mathcal{F}$ such that $c = -x \lor y_1$. As $\sigma$ is a symmetry of $\mathcal{F}$, then $\sigma(c) = -\sigma(x) \lor \sigma(y_1)$ is a clause of $\mathcal{F}$. This implies that $\mathcal{F} \models \sigma(x) \lor \sigma(y_1)$. Now, we suppose that the property holds until $i-1$ and we shall prove that it holds for $i$. We suppose that $\mathcal{F} \models x \lor y_i$, and shall prove that $\mathcal{F} \models \sigma(x) \lor \sigma(y_i)$. From $\mathcal{F} \models x \lor y_i$ we deduce that there exists a clause $c \in \mathcal{F}$ such that $c = a \lor y_i$ where $a \subseteq -x \lor y_1 \lor y_2 \lor \cdots \lor y_{i-1}$. Therefore $\sigma(c) = \sigma(a) \lor \sigma(y_i)$ is a clause of $\mathcal{F}$ such that $\sigma(a) \subseteq -\sigma(x) \lor \sigma(y_1) \lor \sigma(y_2) \lor \cdots \lor \sigma(y_{i-1})$. As $\mathcal{F} \models \sigma(x) \lor \sigma(y_i)$, then $\mathcal{F} \models x \lor y_i$, $\forall j \in \{1, \ldots, i-1\}$. By the induction hypothesis we have $\mathcal{F} \models \sigma(x) \lor \sigma(y_j)$, $\forall j \in \{1, \ldots, i-1\}$. Now, it is trivial that $\mathcal{F} \models \sigma(x) \lor \sigma(y_i)$. It is also obvious that $\text{level}(x)$ in $I$ is identical to $\text{level}(\sigma(x))$ in $\sigma(I)$, and $\forall i \in \{1, \ldots, n\}, \text{level}(y_i)$ in $I$ is identical to $\text{level}(\sigma(y_i))$ in $\sigma(I)$.

From the previous lemma we can deduce the following proposition.

**Proposition 2:** Given a CNF formula $\mathcal{F}$, and a symmetry $\sigma$ of $\mathcal{F}$. If $I=\prod_{i=1}^n(x_i, y_{i,1}, y_{i,2}, \ldots, y_{i,k_i})$ is a partial interpretation formed by $m$ decision literals $x_i$ and all their ordered propagated literals $y_{i,1}, y_{i,2}, \ldots, y_{i,k_i}$ by unit resolution, then we have $\sigma(I) = \prod_{i=1}^n(\sigma(x_i), \sigma(y_{i,1}), \sigma(y_{i,2}), \ldots, \sigma(y_{i,k_i}))$, and $\forall x \in I, \text{level}(x)$ in $I$ is identical to $\text{level}(\sigma(x))$ in $\sigma(I)$.

**Proof:** The proof can be derived by application of the previous lemma recursively on the decision literals $x_i$ and their corresponding propagated literal $y_{i,1}, y_{i,2}, \ldots, y_{i,k_i}$.
Now we introduce the main symmetry property that we will use to improve clause learning in CDCL SAT solvers.

**Proposition 3**: Given a CNF $F$, a symmetry $σ$ of $F$, and a partial interpretation $I = \prod_{i=1}^{m} \langle (x_i), y_{i,1}, y_{i,2}, \ldots, y_{i,k_i} \rangle$ formed by $m$ decision literals $x_i$ and all their ordered propagated literals $y_{i,1}, y_{i,2}, \ldots, y_{i,k_i}$. If $c = \ell_1 \lor \ell_2 \lor \cdots \lor \ell_k \lor x$ is an asserting clause corresponding to $I$ with $x$ its asserted literal, then $σ(c)$ is an asserting clause corresponding to the symmetrical partial interpretation $σ(I)$ with $σ(x)$ as its asserted literal.

**Proof**: We have to prove that $σ(I)[σ(c)] = False$, level($σ(x)$) = $m$ and $∀i \in \{1, \ldots, k\}$, level($σ(\ell_i)$) < $m$ in $σ(I)$. By the hypothesis $c$ is an asserting clause corresponding to $I$, then $I[c] = False$. As $σ$ is a symmetry, then $σ(I)[σ(c)] = False$. From Proposition 2 we can verify the level conditions: level($σ(x)$) = $m$ and $∀i \in \{1, \ldots, k\}$ level($σ(\ell_i)$) < $m$ in $σ(I)$.

This property allows CDCL SAT solvers to add at once to the set of clauses, not only the asserting clause $c$ inferred at the search node corresponding to the no-good $I$, but all its symmetrical asserting clauses $σ(c)$ as well. This avoid to explore isomorphic sub-spaces corresponding to symmetrical no-goods $σ(I)$ of $I$. We will see in the experiments that this property improves dramatically the efficiency of CDCL SAT solvers.

An important happy case, happen when the asserting clause deduced at a given node in the search tree, is unit. In this case, we deduce by symmetry that all the literals in the orbit of the literal forming the asserting clause are unit asserting clauses, then we can propagate directly (thanks to symmetry) all their negations. Formally we have the following property:

**Proposition 4**: Given a CNF $F$, and a partial interpretation $I$. If $ℓ$ is a unit asserting clause, then for all $σ′ ∈ L Sym(L_F)$, $σ′ \land \neg ℓ$ is a unit asserting clause, and $F$ is satisfiable iff $(F \land \neg ℓ \land \bigwedge_{i ∈ L Sym(L_F)} \neg ℓ_i)$ is satisfiable.

**Proof**: The proposition is a particular case of Proposition 2.

The symmetry learning scheme (SLS) is then a classical learning scheme (SL) augmented by the two previous symmetry properties that infer symmetrical asserting clauses to boost clause learning. We show, in the following, how this new learning scheme is implemented in a CDCL SAT solver.

**V. SYMMETRY ADVANTAGE IN TREE SEARCH ALGORITHMS**

Now we will show how the new symmetry learning scheme (SLS), described in the previous section, can be incorporated in a CDCL SAT Solver. The resulting modern clause learning SAT solver is based on unit resolution, clause learning [20], [21] augmented by symmetry, a restarting policy [23], and a backjumping [20], [22]. Below in Algorithm 1, we describe the procedure of the CDCL SAT solver augmented by symmetry. The pseudo-code of the procedure Symmetry Clause Learning SAT solver with restarts (called SCLR) is based on the one given in [29]. All the basic theoretical background, for instance, the sequence of decision points $D$, the assertion level, the non-chronological backtrack, and the restarts are the same as the ones of the classical CDCL algorithms [29]. If $I = D \cup P$ is a partial interpretation of the formula $F$, then the SAT state corresponding to $I$ in the search tree of the considered SAT solver is denoted by $S = (F, Γ, D)$ where $D = (ℓ_1, ℓ_2, \ldots, ℓ_k)$ is the ordered set of decision literals of $I$ and $ℓ_i$ is the literal decision at level $i$. $Γ$ is a CNF such that $F = Γ$. A SAT state $S = (F, Γ, D)$ is U-inconsistent, respectively U-consistent, if only if $F \land Γ \land D$ is U-inconsistent, respectively U-consistent. The clauses of $Γ$ are the asserting clauses that are added to the initial CNF $F$ to express the partial generated interpretations which are no-goods.

The main difference between the procedure SCLR that we propose and classical CDCL SAT procedures is the implementation of the results of both Propositions 3 and 4 in SCLR (lines 10 to 14). Indeed, when there is a conflict partial interpretation $I$ and the decision set $D$ is not empty an asserted clause $c$ is generated and two cases are checked: if the asserted clause $c$ is unit (line 10), then all its symmetrical literals (lateral of its orbit, line 11) are added to $Γ$ (line 14) and their negations are propagated at the root of the search tree. If the asserting clause $c$ is not unit (line 12), then $c$ and all the symmetrical clauses $σ(c)$ induced by the generators $σ ∈ Gen(Sym(L_F))$ (line 13) are added to $Γ$ (line 14).

For efficiency reasons, in our implementation, we limited the generation of symmetrical asserting clauses of a non-unit clause to only the ones produced by the set of the generators of the symmetry group of the formula.

We choose in our implementation the MiniSat[30] solver to be the baseline method that we want to improve by the advantage of symmetry in clause learning. But our approach is generic, it can be used by any other CDCL solver. The experiment results are given in the next section.

**V. EXPERIMENTS**

Now we shall investigate the performances of our search techniques by experimental analysis. We choose for our study a variety of SAT instances to show the behavior of learning by symmetry in satisfiability. We expect that symmetry learning will be more profitable in real-life applications. We tested and compared MiniSat and MiniSat with symmetry clause learning (MiniSat+SymCDCL) on several SAT instances.

First, we performed the two methods on different SAT instances like FPGA (Field Programmable Gate Array), Chnl (Frequency allocation problems), Urq (Urquhart’s problems) and Hole (Pigeon-hole problems) problems that are known to have symmetries. Secondly, we experimented both methods on hard random graph coloring instances. Finally, we tested our approach on different instances from the last SAT competitions. The complexity indicators are the number of nodes of the search tree and the CPU time. The time needed for computing symmetry is added to the total CPU time of the search in MiniSat+SymCDCL. The source codes are written.
Algorithm 1: SCLR: Symmetry Clause Learning SAT Solver with Restarts

input : CNF formula $\mathcal{F}$
output : A solution of $\mathcal{F}$ or unsat if $\mathcal{F}$ is not satisfiable

1. $D \leftarrow \emptyset$ // Decision literals;
2. $\Gamma \leftarrow \text{True}$ // learned clauses;
3. while True do
   4. if $S = (\mathcal{F}, \Gamma, D)$ is $U$ - inconsistent then
      5. // There is a conflict.
      6. if $D = \emptyset$ then
         7. return unsat
      8. $c \leftarrow$ an asserting clause of $S$
      9. $m \leftarrow$ the assertion level of $c$
     10. if $c$ is unit then
         11. $A \leftarrow \{ \ell \mid \ell \in c^{\text{Sym}(L_{\mathcal{F}})} \}$
     12. else
         13. $A \leftarrow \{ \sigma(c) \mid \sigma \in \text{Gen}(Sym(L_{\mathcal{F}})) \}$
     14. $\Gamma \leftarrow \Gamma \cap c$
     15. $D \leftarrow D_m$ // the first $m$ decisions
   16. else
      17. // There is no conflict.
      18. if time to restart then
         19. $D \leftarrow \emptyset$
      20. Choose a literal $\ell$ such that $S \not\models \ell$ and $S \not\models \ell$
      21. if $\ell = \text{null}$ then
         22. return $D$ // satisfiable
      23. $D \leftarrow D, \ell$

in C++ and compiled on a Pentium 4, 2.8 GHZ and 1 Gb of RAM.

A. The results on the different symmetrical SAT instances

Table I shows the results of our method on some known SAT instances. It gives the instance, the instance size ($\#V/\#C$), the number of nodes of the search tree and the CPU time for each method.

Table I shows that our method MiniSat+SymCDCL is in general better than MiniSat in both the number of nodes and the CPU time on the Hole, and Chnl problems. For the FPGA problems, we obtain similar results in time. This is due to the fact that these problems are satisfiable and MiniSat succeeds to find a solution quickly. The Urq instances are known to be harder than the FPGA and the Chnl problems. We can see that minisat is just able to solve the first problem and failed to solve the two other ones by the time limit. Our method solved all of them efficiently.

B. The results on the graph coloring instances

Random graph coloring problems are generated with respect to the following parameters: (1) $n$ : the number of vertices, (2) Colors: the number of colors and (3) $d$: the density which is a number between 0 and 1 expressed by the ratio : the number of constraints (the number of edges in the graph) to the number of all possible constraints. For each test corresponding to some fixed values of the parameters $n$, Colors, and $d$, a sample of 100 instances are randomly generated and the measures (CPU time, nodes) are taken on the average. The time limit was set to 800 seconds.

We reported in Figure 2 the practical results of the methods MiniSat, and MiniSat+SymCDCL, on three classes of random graph coloring problems where the number of variables (the vertices of the graph) is set to 35, 45 and 55 and where the density is $(d = 0.5)$ for each class. The big curves of Figure 2 represent the average CPU times with respect to the number of colors of both methods for each instance class. The small ones express the average number of nodes of both methods for each problem class.

From the nodes curves and the CPU time curves, we can see that in average MiniSat+SymCDCL explores less nodes in the search tree than MiniSat and is faster on these problems.
We can see that clause learning by symmetry is profitable in the hard region and our method MiniSat+SymCDCL solved the instances of the class with 55 vertices, that MiniSat fails to solve in the hard region within the time limit. This explains why both the node and the CPU time curves of MiniSat for this class are not plotted in Figure 2.

C. The results on instances from the last SAT competitions

We report here some results of both methods on 180 instances of different class instances taken from the last SAT competitions. The time limit is fixed to 1200 seconds for each instance. Our first purpose behind this, is to discover instances that contain symmetries and see the behavior of symmetry clause learning on them. Figure 3 shows a comparison of MiniSat+SymCDCL and MiniSat in number of nodes (the bottom part) and in CPU time (the top part) on the 180 instances. In both figure parts, the y-axis (resp. x-axis) corresponds to the performance of MiniSat+SymCDCL (resp. MiniSat). A plotted dot on the top figure part (resp. bottom figure part) represents the CPU time (resp. the number of nodes) of the two solvers for a given SAT instance. The projection of the plotted dot on the y-axis (resp. on the x-axis) expresses the performance of MiniSat+SymCDCL (resp. MiniSat) in CPU time for the top part and in number of nodes for the bottom part. Dots below the diagonal indicate that MiniSat+SymCDCL outperforms MiniSat, around the diagonal the performances are comparable and above the diagonal MiniSat is better than MiniSat+SymCDCL. We can see on Figure 3, that there exist several instances where clause learning take advantage of symmetry and on which MiniSat+SymCDCL outperforms MiniSat. We can also see that MiniSat is better in CPU time than MiniSat+SymCDCL on some instances. This usually happen when the instance is satisfiable and when MiniSat find quickly a solution.

Table II gives in detail the results of both methods on some instances among the ones of SAT competitions on which MiniSat+SymCDCL outperforms MiniSat in both CPU time and the number of nodes. We can remark that MiniSat+SymCDCL solved the instance gus-md5-10 that MiniSat did not solve by the time limit.

<table>
<thead>
<tr>
<th>Instance</th>
<th>#V</th>
<th>#C</th>
<th>Nodes</th>
<th>Time</th>
<th>MiniSat+SymCDCL Nodes</th>
<th>Times</th>
</tr>
</thead>
<tbody>
<tr>
<td>gus-md5-05</td>
<td>668</td>
<td>27</td>
<td>3482</td>
<td>16.2</td>
<td>3928</td>
<td>2.84</td>
</tr>
<tr>
<td>gus-md5-06</td>
<td>689</td>
<td>33</td>
<td>22468</td>
<td>23.3</td>
<td>1928</td>
<td>1.43</td>
</tr>
<tr>
<td>gus-md5-07</td>
<td>10909</td>
<td>27</td>
<td>223425</td>
<td>165.9</td>
<td>16583</td>
<td>101.22</td>
</tr>
<tr>
<td>gus-md5-09</td>
<td>69487</td>
<td>27</td>
<td>226581</td>
<td>747.90</td>
<td>68221</td>
<td>474.90</td>
</tr>
<tr>
<td>gus-md5-10</td>
<td>69503</td>
<td>27</td>
<td>226618</td>
<td>1143.27</td>
<td>97146</td>
<td>714.27</td>
</tr>
<tr>
<td>mod2-3cage-9-12</td>
<td>87</td>
<td>232</td>
<td>933790</td>
<td>76.14</td>
<td>19204</td>
<td>42.04</td>
</tr>
<tr>
<td>mod2-3cage-9-14</td>
<td>87</td>
<td>232</td>
<td>875384</td>
<td>68.79</td>
<td>7490</td>
<td>0.18</td>
</tr>
<tr>
<td>gus-md5-05</td>
<td>668</td>
<td>27</td>
<td>3482</td>
<td>16.2</td>
<td>3928</td>
<td>2.84</td>
</tr>
<tr>
<td>gus-md5-06</td>
<td>689</td>
<td>33</td>
<td>22468</td>
<td>23.3</td>
<td>1928</td>
<td>1.43</td>
</tr>
<tr>
<td>gus-md5-07</td>
<td>10909</td>
<td>27</td>
<td>223425</td>
<td>165.9</td>
<td>16583</td>
<td>101.22</td>
</tr>
<tr>
<td>gus-md5-09</td>
<td>69487</td>
<td>27</td>
<td>226581</td>
<td>747.90</td>
<td>68221</td>
<td>474.90</td>
</tr>
<tr>
<td>gus-md5-10</td>
<td>69503</td>
<td>27</td>
<td>226618</td>
<td>1143.27</td>
<td>97146</td>
<td>714.27</td>
</tr>
<tr>
<td>mod2-3cage-9-12</td>
<td>87</td>
<td>232</td>
<td>933790</td>
<td>76.14</td>
<td>19204</td>
<td>42.04</td>
</tr>
<tr>
<td>mod2-3cage-9-14</td>
<td>87</td>
<td>232</td>
<td>875384</td>
<td>68.79</td>
<td>7490</td>
<td>0.18</td>
</tr>
<tr>
<td>gus-md5-05</td>
<td>668</td>
<td>27</td>
<td>3482</td>
<td>16.2</td>
<td>3928</td>
<td>2.84</td>
</tr>
<tr>
<td>gus-md5-06</td>
<td>689</td>
<td>33</td>
<td>22468</td>
<td>23.3</td>
<td>1928</td>
<td>1.43</td>
</tr>
<tr>
<td>gus-md5-07</td>
<td>10909</td>
<td>27</td>
<td>223425</td>
<td>165.9</td>
<td>16583</td>
<td>101.22</td>
</tr>
<tr>
<td>gus-md5-09</td>
<td>69487</td>
<td>27</td>
<td>226581</td>
<td>747.90</td>
<td>68221</td>
<td>474.90</td>
</tr>
<tr>
<td>gus-md5-10</td>
<td>69503</td>
<td>27</td>
<td>226618</td>
<td>1143.27</td>
<td>97146</td>
<td>714.27</td>
</tr>
<tr>
<td>mod2-3cage-9-12</td>
<td>87</td>
<td>232</td>
<td>933790</td>
<td>76.14</td>
<td>19204</td>
<td>42.04</td>
</tr>
<tr>
<td>mod2-3cage-9-14</td>
<td>87</td>
<td>232</td>
<td>875384</td>
<td>68.79</td>
<td>7490</td>
<td>0.18</td>
</tr>
<tr>
<td>gus-md5-05</td>
<td>668</td>
<td>27</td>
<td>3482</td>
<td>16.2</td>
<td>3928</td>
<td>2.84</td>
</tr>
<tr>
<td>gus-md5-06</td>
<td>689</td>
<td>33</td>
<td>22468</td>
<td>23.3</td>
<td>1928</td>
<td>1.43</td>
</tr>
<tr>
<td>gus-md5-07</td>
<td>10909</td>
<td>27</td>
<td>223425</td>
<td>165.9</td>
<td>16583</td>
<td>101.22</td>
</tr>
<tr>
<td>gus-md5-09</td>
<td>69487</td>
<td>27</td>
<td>226581</td>
<td>747.90</td>
<td>68221</td>
<td>474.90</td>
</tr>
<tr>
<td>gus-md5-10</td>
<td>69503</td>
<td>27</td>
<td>226618</td>
<td>1143.27</td>
<td>97146</td>
<td>714.27</td>
</tr>
<tr>
<td>mod2-3cage-9-12</td>
<td>87</td>
<td>232</td>
<td>933790</td>
<td>76.14</td>
<td>19204</td>
<td>42.04</td>
</tr>
<tr>
<td>mod2-3cage-9-14</td>
<td>87</td>
<td>232</td>
<td>875384</td>
<td>68.79</td>
<td>7490</td>
<td>0.18</td>
</tr>
</tbody>
</table>

VI. CONCLUSION AND PERSPECTIVES

In this paper, we augmented clause learning by symmetry. We introduced a new generic learning scheme (SLS) that can be used by any CDCL SAT solver. The symmetry learning scheme infers in addition to a generated asserting clause at a node of the search tree all its symmetrical asserting clauses. Considering the symmetrical asserting clauses often avoid the CDCL SAT solvers exploring isomorphic search sub-spaces. We implemented the symmetry learning scheme (SLS) in MiniSat solver and experimented both MiniSat and MiniSat with the SLS scheme on a great variety of SAT instances. The experimental results are very encouraging and show that symmetry is profitable for clause learning on almost all the checked instances.

As a future improvement, we are looking to find a good strategy of clause reduction in our implementation that should preserve the added symmetrical asserting clauses. Indeed, MiniSat could remove symmetrical asserting clauses that could be used to prune isomorphic sub-spaces.

Another point, is that only the symmetries of the initial problem that called global symmetries are used in the symmetry learning scheme. We are looking to extend our clause learning approach to exploit the local symmetries that can be detected dynamically at each node of the search tree.