Local symmetry breaking in the satisfiability problem

Belaïd Benhamou, Tarek Nabhani, Richard Ostrowski, Mohamed Réda Saïdi

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
Email:{benhamou; nabhani; ostrowski; saidi}@cmi.univ-mrs.fr

Abstract. The SAT problem is shown to be the first decision NP-complete problem (Cook,71). It is central in complexity theory. In the last decade, the satisfiability proof procedures are improved by symmetry elimination. A CNF formula usually contains an interesting number of symmetries. There are two kinds of symmetry exploitation. The first one corresponds to global symmetry breaking, that is, only the symmetries of the initial problem (the problem at the root of the search tree) are detected and eliminated. The second one deals with all local symmetries that appear at each node of the search tree. Local symmetry has to be detected and eliminated dynamically during the search. Exploiting such symmetries seems to be a hard task. Almost all of the known works on symmetry in satisfiability are on global symmetry. Only few works are carried on local symmetry, despite their importance in practice. An important challenge is then to detect and break local symmetries efficiently during the search. The work that we present here is a contribution towards an answer to this hard challenge. We present a new method for local symmetries breaking that consists in detecting dynamically local symmetries by reducing the remaining partial SAT instance at each node of the search tree to a graph that has an equivalent automorphism group than the symmetry group of the partial SAT instance. We used the software Saucy to compute the automorphism group and implemented a local symmetry cut in a SAT solver. We experimented this method on several SAT instances and compared it with a method exploiting global symmetries. The results obtained are very promising. Local symmetry improves global symmetry on some hard instances and is complementary to global symmetry.

1 Introduction

Krishnamurthy in [20] 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 are used earlier for the resolution of many problems such as the eight queens [15]. They are also introduced into the resolution of a constraints satisfaction problem [13, 26, 3], in an intelligent algorithm of Backtracking [9] and in the first order logic [11]. Symmetry becomes an important notion in constraint programming. During the last decade, several works on symmetry breaking for the satisfiability problem and in CSPs appeared. Nevertheless, few search works deal with dynamic symmetry detection and elimination [4–7, 14]. Most of the methods exploiting symmetries deal only with global symmetries [17, 2, 1], that is, the symmetries of the initial problem corresponding to the root of the search tree . A symmetry of a logical formula is a literal permutation leaving invariant the formula. There are many problems in artificial intelligence which contain an important number of symmetries when expressed in CNF formulas. The importance of symmetry breaking when solving these problems can be seen on some difficult problems which are badly handled by the classic search methods. Take for instance the Pigeon-holes problem [8, 16], or the Ramsey problem [18]. Both problems are known to be hard for the classic resolution methods, and they are well represented in first order logic by a small set of formulas that becomes very wide when we calculate all the propositional terminal instantiations. The set of propositional clauses obtained, contains an important number of symmetries. That is, the set of clauses remains invariant under several variable permutations. Exploiting such permutations results in a polynomial complexity for the satisfiability proof while both problems are known to be exponential for the methods that do not take into account symmetry breaking. 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 [17, 2, 1]. But, almost all of them deal only with global symmetry and ignore the treatment of local symmetries. This is due to the difficulty of detecting and exploiting them dynamically, contrary to global symmetries that can be treated by static approaches that are easier to implement. An approach of dynamic detection of local symmetries in propositional logic was proposed in [4–6]. but this method is incomplete, in the sense that it detects only some local symmetries, not the total group of local symmetry. An alternative to this method is to adapt and use the graph automorphism computational tool Saucy [2] that is able to detect all the local symmetries during search, since the group of automorphisms of the graph deduced from the SAT instance is identical to the symmetry group of the SAT instance. In this paper, we present an alternative local symmetry breaking method for the SAT problem that exploits the total group of symmetry. This method consists in reducing incrementally the logical sub-formula defined at each search node to a graph on which we apply graph automorphism detection tools like Saucy [2]. Symmetry elimination is implemented in a SAT solver that we experimented and compared on several SAT instances. The obtained results are very promising and show that local symmetry breaking outperforms global symmetry breaking on some SAT instances and its combination with global symmetry seems to be complementary. The rest of the paper is organized as follows: Section 2 gives some background on the satisfiability problem and permutations. Section 3 defines the symmetry principle and gives some symmetry properties. The fourth section describes the new symmetry detection and elimination method that we propose. Section 5 shows how the symmetry cut is integrated in a tree search method like Davis and Putnam procedure. We evaluate the proposed method in the sixth section where several SAT instances are tested and where a comparison of our method with some other existing methods is given. Finally, we conclude the work in Section 7.

2 Some background on propositional logic

2.1 Propositional logic

We shall assume that the reader is familiar with the propositional calculus. We give here, a short description, a more complete description can be found in [21]. Let V be the set of propositional variables called only variables. Variables will be distinguished from literals, which are variables with an assigned parity 1 or 0 that means True or False, respectively. This distinction will be ignored whenever it is convenient, but not confusing. For a propositional variable p, there are two literals: p the positive literal and $\neg p$ the negative one.

A clause is a disjunction of literals $\{p_1, p_2, \ldots, p_n\}$ such that no literal appears more than once, nor a literal and its negation at the same time. This clause is denoted by $p_1 \vee p_2 \vee \ldots \vee p_n$. A system $\mathcal{F}$ of clauses is a conjunction of clauses. In other words, we say that $\mathcal{F}$ is in the conjunctive normal form (CNF).

A truth assignment to a system of clauses $\mathcal{F}$ is a mapping I defined from the set of variables of $\mathcal{F}$ into the set $\{$True, False$\}$. If $I[p]$ is the value for the positive literal p then $I[\neg p] = 1 - I[p]$. The value of a clause $p_1 \vee p_2 \vee \ldots \vee p_n$ in I is True, if the value True is assigned to at least one of its literals in I, False otherwise. By convention, we define the value of the empty clause ($n = 0$) to be False. The value $I[\mathcal{F}]$ of the system of clauses is True if the value of each clause of $\mathcal{F}$ is True, False, otherwise. We say that a system of clauses $\mathcal{F}$ is satisfiable if there exists some truth assignments I that assign the value True to $\mathcal{F}$, it is unsatisfiable otherwise. In the first case I is called a model of $\mathcal{F}$. Let us remark that a system which contains the empty clause is unsatisfiable.

It is well-known [27] that for every propositional formula $\mathcal{F}$ there exists a formula $\mathcal{F}'$ in conjunctive normal form(CNF) such that the length of $\mathcal{F}'$ is at most 3 times as long as the formula $\mathcal{F}$ and $\mathcal{F}'$ is satisfiable iff $\mathcal{F}$ is satisfiable. In the following we will assume that the formulas are given in a conjunctive normal form.

2.2 Permutations

Let $\Omega = \{1, 2, \ldots, N\}$ for some integer N, where each integer might represent a propositional variable. A permutation of Ω is a bijective mapping σ from Ω to Ω that is usually represented as a product of cycles of permutations. We denote by $Perm(\Omega)$ the set of all permutations of Ω and $\circ$ the composition of the permutation of $Perm(\Omega)$. The pair $(Perm(\Omega), \circ)$ forms the permutation group of Ω. 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 $(S, \circ)$ iff T is a subset of S and forms a group under the operation $\circ$.

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

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

3 Symmetry

Since Krishnamurthy's [20] symmetry definition in propositional logic, several other definitions are given in the CP community. Freuder in his work [13], introduced the notions of full and neighborhood interchangeabilities, where two domain values are interchangeable in a CSP, if they can be substituted for each other without any effects to the CSP. In the other hand Benhamou in [3] defined two levels of semantic symmetry and a notion of syntactic symmetry. He also showed that the Full interchangeability of Freuder is a particular case of semantic symmetry and Neighborhood interchangeability is a particular case of syntactic symmetry. More recently a work of Cohen et al [10] discussed most of the known symmetry definitions in CSPs and gathered them in two definitions: symmetry of solutions (semantic) and symmetry of constraints (syntactic). Almost all of these definitions can be identified to belong to the two families of symmetry: syntactic symmetry or semantic symmetry. We will define in the following both semantic and syntactic symmetry in propositional logic and show their relationship with the solution and constraint symmetries in CSPs.

3.1 Symmetry in propositional logic

Definition 1 (Semantic symmetry). *Let $\mathcal{F}$ be a propositional formula given in CNF and $L_\mathcal{F}$ its complete* [1] *set of literals. A semantic symmetry of $\mathcal{F}$ is a permutation σ defined on $L_\mathcal{F}$ such that $\mathcal{F} \models \sigma(\mathcal{F})$ and $\sigma(\mathcal{F}) \models \mathcal{F}$.*

In other words a semantic symmetry of a formula is a literal permutation that conserves the set of the models of the formula. We recall in the following the definition of syntactic symmetry given in [4, 5]

Definition 2 (Syntactic symmetry). *Let $\mathcal{F}$ be a propositional formula given in CNF and $L_\mathcal{F}$ its complete set of literals. A syntactic symmetry of $\mathcal{F}$ is a permutation σ 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 syntactical symmetry of a formula is a literal permutation that leaves the formula invariant. If we denote by $Perm(L_\mathcal{F})$ the group of permutations of $L_\mathcal{F}$ and by $Sym(L_\mathcal{F}) \subset Perm(L_\mathcal{F})$ the subset of permutations of $L_\mathcal{F}$ that are the syntactic symmetries of $\mathcal{F}$, then $Sym(L_\mathcal{F})$ is trivially a sub-group of $Perm(L_\mathcal{F})$.

[1] The set of literals containing each literal of $\mathcal{F}$ and its negation

Remark 1. The symmetry definitions introduced in CSPs [10] are related to the former ones introduced in propositional logic. Consider for instance the direct SAT encoding $\mathcal{F}$ of a CSP P [19] where a boolean variable is introduced for each CSP variable-value pair, and where a clause forbidding each tuple disallowed by a specific constraint is added as well as another clause ensuring that a value is chosen for each variable in its domain. It is then trivial that the solution symmetry of the CSP P is equivalent to the semantic symmetry (Definition 1) of its SAT encoding $\mathcal{F}$ and the constraint symmetry of P is equivalent to the syntactic symmetry (Definition 2) of $\mathcal{F}$.

Theorem 1. *Each syntactical symmetry of a formula $\mathcal{F}$ is a semantic symmetry of $\mathcal{F}$.*

Proof. It is trivial to see that a syntactic symmetry is a sufficient condition to a semantic symmetry. Indeed, if σ is syntactic symmetry of $\mathcal{F}$, then $\sigma(\mathcal{F}) = \mathcal{F}$, thus it results that $\mathcal{F}$ and $\sigma(\mathcal{F})$ have the same set of models. Each syntactic symmetry is a semantic symmetry and the converse is in general not true.

Example 1. Let $\mathcal{F}$ be the following set of clauses: $\mathcal{F}=\{a \vee b \vee c, \neg a \vee b, \neg b \vee c, \neg c \vee a, \neg a \vee \neg b \vee \neg c\}$ and σ_1 and σ_2 two permutations defined on the complete set $L_{\mathcal{F}}$ of literals occurring in $\mathcal{F}$ as follows:
$\sigma_1=(a, b, c)(\neg a, \neg b, \neg c)$
$\sigma_2=(a, \neg a)(b, \neg b)(c, \neg c)$
Both σ_1 and σ_2 are syntactic symmetries of $\mathcal{F}$, since $\sigma_1(\mathcal{F})=\mathcal{F}=\sigma_2(\mathcal{F})$.

In the sequel we deal only with syntactic symmetry, we say only symmetry to designate syntactic symmetry.

Definition 3. *Two literals ℓ and ℓ' of a formula $\mathcal{F}$ are symmetrical if there exists a symmetry σ of $\mathcal{F}$ such that $\sigma(\ell) = \ell'$.*

Definition 4. *Let $\mathcal{F}$ be a formula, the orbit of a literal $\ell \in L_{\mathcal{F}}$ on which the group of symmetries $Sym(L_{\mathcal{F}})$ acts is $\ell^{Sym(L_{\mathcal{F}})}=\{\sigma(\ell) : \sigma \in Sym(L_{\mathcal{F}})\}$*

Proposition 1. *All the literals in the orbit of a literal ℓ are symmetrical two by two.*

Proof. The proof is a trivial consequence of the previous two definitions

Example 2. In Example 1, the orbit of the literal a is $a^{Sym(L_{\mathcal{F}})}= \{a, b, c, \neg a, \neg b, \neg c\}$. We can see that all the literals are in the same orbit. Thus, they are all symmetrical.

If I is a model of $\mathcal{F}$ and σ a symmetry, we can get another model of $\mathcal{F}$ by applying σ on the variables which appear in I. That is, if I is a model of $\mathcal{F}$ then $\sigma(I)$ is a model of $\mathcal{F}$. A symmetry σ transforms each model into a model and each no-good into a no-good. In the following propositions, we assume that σ is a symmetry of the set of clauses $\mathcal{F}$.

Proposition 2. *Let ℓ be a literal, σ a symmetry such that $\ell' = \sigma(\ell)$ and $I' = \sigma(I)$. If I is such that $I[\ell] = True$, then I' is such that $I'[\ell'] = true$*

Proof. The proof is trivial. Indeed, if ℓ is true in the model I then $\sigma(\ell)= \ell'$ will be true in the model $\sigma(I)=I'$.

we deduce the following proposition.

Proposition 3. *If a literal ℓ has the value true in a model of $\mathcal{F}$, then $\sigma(\ell)$ will have the value true in a model of $\mathcal{F}$.*

Theorem 2. *Let ℓ and ℓ' be two literals of $\mathcal{F}$ that are in the same orbit with respect to the symmetry group $Sym(L_{\mathcal{F}})$, then ℓ is true in a model of $\mathcal{F}$ iff ℓ' is true in a model of $\mathcal{F}$.*

Proof. If ℓ is in the same orbit as ℓ' then it is symmetrical with ℓ' in $\mathcal{F}$. Thus, there exists a symmetry σ of $\mathcal{F}$ such that $\sigma(\ell) = \ell'$. If I is a model of $\mathcal{F}$ then $\sigma(I)$ is also a model of $\sigma(\mathcal{F}) = \mathcal{F}$, besides if $I[\ell] = true$ then $\sigma(I[\ell']) = true$ (Proposition 2). For the converse, consider $\ell = \sigma^{-1}(\ell')$, and make a similar proof.

Corollary 1. *Let ℓ be a literal of $\mathcal{F}$, if ℓ is not true in any model of $\mathcal{F}$, then each literal $\ell' \in orbit^{L_{\mathcal{F}}}$ is not true in any model of $\mathcal{F}$.*

Proof. The proof is a direct consequence of Theorem 2

Corollary 1 expresses an important property that we will use to break local symmetry at each node of the search tree. That is, if a failure is detected after assigning the value True to the current literal ℓ, then we compute the orbit of ℓ and assign the value false to each literal in it, since by symmetry the value true will be contradictory, then will not participate in any model of the considered formula.

Many hard problems for resolution have been shown to be polynomial when using symmetry in resolution. For instance, finding some of the Ramsey's numbers or solving the pigeon-hole problem are known to be exponential for classical resolution, while short proofs can be made for both them when adding the symmetry rule to the resolution proof system. We will show now how to detect dynamically the local symmetry.

4 Local symmetry detection and elimination

Local symmetries have to be detected dynamically at each node of the search tree. Dynamic symmetry detection had been studied in [4, 5] where a local syntactic symmetry search method had been given. However, this method is not complete, it detects only one symmetry σ at each node of the search tree when failing in the assignment of the current literal ℓ. A heuristic is used on the variable permutations of σ in order to get a maximal number of literals in the same cycle of permutations as the one where ℓ appears. Despite this heuristic, this method does not detect all the symmetrical literals with ℓ corresponding to the orbit of ℓ, since it does not use all the local symmetries.

As an alternative to this incomplete symmetry search method, we adapted Saucy [2] to detect all the local syntactic symmetries and show how to break such symmetries during search. Saucy is a tool for computing the automorphism group of a graph. Other tools like Nauty [22] or the most recent methods AUTOM [25] or the one described in [23] can be adapted to search local symmetry. It is shown in [25] that AUTOM is the best method. Because the source code of AUTOM is not free, and more recently Saucy had been improved [12]; we chose Saucy. It is shown in [17, 2, 1] 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 the 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 syntactic group of symmetries of $\mathcal{F}$. That is, the syntactic symmetry group of the formula $\mathcal{F}$ is identical to the automorphism group of its graph representation $G_{\mathcal{F}}$, thus we use Saucy on $G_{\mathcal{F}}$ to detect the syntactic symmetry group of $\mathcal{F}$. Saucy returns a set of generators *Gen* of the symmetry group from which we can deduce each symmetry. Saucy 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 having 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 Saucy can be found at (http://vlsicad.eecs.umich.edu/BK/SAUCY/).

Example 3. Let $\mathcal{F}$ be the CNF formula given in Example 1. Its associated $G_{\mathcal{F}}$ is given in Figure 1

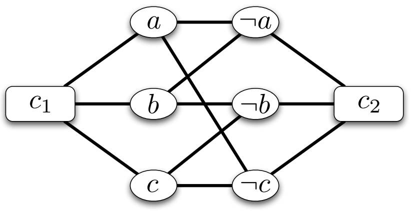

Fig. 1. The graph $G_{\mathcal{F}}$ corresponding to $\mathcal{F}$

Dynamic symmetry detection: Consider a CNF formula $\mathcal{F}$, and a partial assignment I of $\mathcal{F}$ where ℓ is the current literal under assignment. The assignment I simplifies the given formula $\mathcal{F}$ into a sub-formula $\mathcal{F}_I$ that defines a state in the search space corresponding to the current node n_I of the search tree. The main idea is to maintain dynamically the graph $G_{\mathcal{F}_I}$ of the sub-formula $\mathcal{F}_I$ corresponding to the local sub-problem defined at the current node n_I, then color the graph $G_{\mathcal{F}_I}$ and compute its automorphism group $Aut(\mathcal{F}_I)$. The sub-formula $\mathcal{F}_I$ can be viewed as the remaining sub-problem corresponding to the unsolved part. By applying Saucy on this colored graph we can get the generator set Gen of the symmetry sub-group existing between literals of $L_{\mathcal{F}_I}$ from which we can compute the orbit of the current literal ℓ that we will use to make the symmetry cut.

Symmetry elimination: We use Corollary 1 to prune search spaces of tree search methods. Indeed, if the assignment of the value true to the current literal ℓ defined at a given node n_I of the search tree is shown to be a failure, then the assignment of the value true to each literal in the orbit of ℓ will result in a failure too. Thus, the value $false$ has to be assigned to each literal in the orbit of ℓ. Therefore we prune the sub-space which corresponds to the assignment of the alternative value *true* to these literals in the search tree. That is what we call the symmetry cut.

5 Symmetry advantage in tree search algorithms

Now we will show how these detected symmetrical literals can be used to increase the efficiency of CNF SAT algorithms. We choose in our implementation the Davis Putnam (DP) procedure to be the baseline method that we want to improve by the advantage of local symmetry elimination.

If I is an inconsistent partial interpretation in which the assignment of the value *true* to the current literal ℓ is shown to be conflicting, then according to Corollary 1, all the literals in the orbit of ℓ computed by using the group $Sym(\mathcal{F}_I)$ returned by Saucy are symmetrical to ℓ. Thus, we assign the value false to each literal in $\ell^{Sym(L_{\mathcal{F}})}$ since the value true is shown to be contradictory, and then we prune the sub-space which corresponds to the value true assignments. The resulting procedure called Satisfiable is given in Figure 2.

The function $orbit(\ell, Gen)$ is elementary, it computes the orbit of the literal ℓ from the set of generators Gen returned by Saucy.

6 Experiments

Now we shall investigate the performances of our search techniques by experimental analysis. We choose for our study some SAT instances to show the local symmetry behavior in satisfiability. We expect that symmetry breaking will be more profitable in real-life applications. Here, we tested and compared four methods:

```
Procedure Satisfiable($\mathcal{F}$);
begin
    if $\mathcal{F} = \emptyset$ then $\mathcal{F}$ is satisfiable
    else if $\mathcal{F}$ contains the empty clause, then $\mathcal{F}$ is unsatisfiable
      else begin
        if there exists a mono-literal or a monotone literal $\ell$ then
          if Satisfiable($\mathcal{F}_\ell$) then $\mathcal{F}$ is satisfiable
          else $\mathcal{F}$ is unsatisfiable
        else begin
          Choose an unsigned literal $\ell$ of $\mathcal{F}$
          if Satisfiable($\mathcal{F}_\ell$) then $\mathcal{F}$ is satisfiable
          else
          begin
              $Gen$=Saucy($\mathcal{F}$);
              $\ell^{Sym(L_\mathcal{F})}$=orbit($\ell$,$Gen$)=$\{\ell_1, \ell_2, ..., \ell_n\}$;
              if Satisfiable($\mathcal{F}_{\neg\ell_1 \wedge \neg\ell_2 \wedge \ldots \wedge \neg\ell_n}$) then $\mathcal{F}$ is satisfiable
              else $\mathcal{F}$ is unsatisfiable
          end
        end
end
```

Fig. 2. The Davis Putnam procedure with local symmetry elimination

		No-sym		Global-sym		Local-sym		Global-Local-sym	
Instance	*Vars : clauses*	*Nodes*	*Time*	*Nodes*	*Times*	*Nodes*	*Time*	*Nodes*	*Time*
fpga10_8_SAT	120 : 448	6,637,776	44.41	449	0.02	9835	2.09	449	0.71
fpga10_9_SAT	135 : 549	-	>1,000	284	0.02	57080	20.37	284	0.53
fpga12_8_SAT	144 : 560	6,637,776	35.79	165	0.00	9835	2.14	165	0.32
fpga13_10_SAT	195 : 905	-	>1,000	4261	0.41	304,830	134.89	4261	14.08
Chnl10_11	220 : 1122	3,628,800	100.09	382	0.09	512	2.42	382	3.33
Chnl10_12_3	240 : 1344	3,628,800	120.72	322	0.10	512	2.63	322	3.41
Chnl11_12_3	264 : 1476	-	>1,000	1123	0.26	1024	6.28	1123	12.09
Chnl11_13	286 : 1742	-	>1,000	814	0.25	1024	7.38	814	10.96
Chnl11_20	440 : 4220	-	>1,000	523	0.38	1024	18.93	523	18.90
Urq3_5	46 : 470	-	>1,000	16384	0.16	30	0.09	15	0.00
Urq4_5	74 : 694	-	>1,000	-	>1,000	44	0.32	31	0.10
Urq5_5	121 : 1210	-	>1,000	-	>1,000	73	1.43	44	0.27
Urq6_5	180 : 1756	-	>1,000	-	>1,000	110	4.76	84	2.03
Urq7_5	240 : 2194	-	>1,000	-	>1,000	147	9.32	108	3.44
Urq8_5	327 : 3252	-	>1,000	-	>1,000	225	27.11	171	9.18

Table 1. Results on some SAT instances

1. **No-sym:** search without symmetry breaking by using the LSAT [24] as the baseline method;
2. **Global-sym** search with global symmetry breaking. This method uses in pre-processing phase the program SHATTER [2, 1] that detects and eliminates the global symmetries of the considered instance by adding on it symmetry breaking clauses, then apply the solver LSAT to the resulting instance. The CPU time of *Global-sym* in Table 1 includes the time that SHATTER spends to compute the global symmetry.

3. **Local-sym:** search with local symmetry breaking. This method implements in LSAT the dynamic local symmetry detection and elimination strategy described in this work. The CPU time of *Local-sym* includes local symmetry search time.
4. **Global-Local-sym:** search that combines both the global and local symmetries. It consists in applying LSAT with local symmetry elimination on the instance produced by SHATTER in the pre-processing phase.

on different SAT instances that are FPGA (Field Programmable Gate Array), Chnl, Urquhart and some random graph coloring instances. The common baseline search method for the three previous methods is LSAT. The complexity indicators are the number of nodes of the search tree and the CPU time. Both the time needed for computing local symmetry and global symmetry are added to the total CPU time of search. The source codes are written in C and compiled on a Pentium 4, 2.8 GHZ and 1 Gb of RAM.

6.1 The results on the different SAT instances

Table 1 shows the first results of the methods on some SAT instances. It gives the instance, the instance size ($variables/clauses$), the number of nodes of the search tree and the CPU time for each method.

Table 1 shows that *Global-sym* is in general better than *Local-sym* and No-sym in both the number of nodes and the CPU time on the $FPGA$, and $Chnl$ problems, but *Local-sym* still able to solve them too. These problems contain a great amount of global symmetries, that is why it is sufficient to break only the global symmetry to solve them efficiently, eliminating local symmetry in these problems may sometimes slow the resolution. The Urq instances are known to be harder than the $FPGA$ and the $Chnl$, we can see that No-sym is not able to solve them and *Global-sym* solved only the $Urq3\_5$ and failed to solve all the other ones under the time limit. *Local-sym* solved all the Urq instances efficiently, local symmetry elimination is then more profitable than global symmetry on the Urq instances. We can see that in average the method *Global-Local-sym* is better than all the other methods, it solved all the instances efficiently. It compares well to *Global-sym* on the $FPGA$ and $Chnl$ instances and to *Local-sym* on the Urq instances. It is then profitable to combine both symmetry eliminations to solve these problems, the results confirmed that both methods could be complementary.

6.2 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.

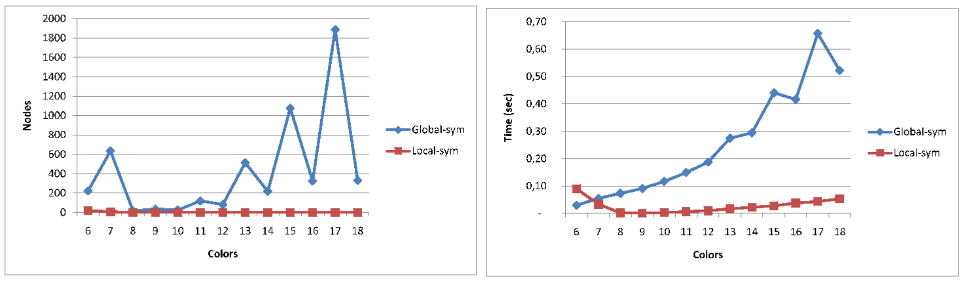

Fig. 3. Node and Time curves of the two symmetry methods on random graph coloring where $n = 30$ and $d = 0.5$

We reported in Figure 3 the practical results of the methods: *Global-sym*, and *Local-sym*, on the random graph coloring problem where the number of variables is $n = 30$ and where the density is ($d = 0.5$). The curves give the number of nodes respectivily the cpu time with respect to the number of colors for each search method.

We can see on the node curves (the curves at the left) that *Local-sym* detects and eliminates more symmetries than the *Global-sym* method and *Global-sym* is not stable for graph coloring. From the CPU time curves (the curves at the right), we can see that *Local-sym* is in average faster than *Global-sym* even that Saucy is run at each node. Local symmetry elimination is profitable for solving random graph coloring instances and outperforms dramatically global symmetry breaking on these problems.

7 Conclusion and perspectives

Here, we extended symmetry detection and elimination to local symmetry. That is, the symmetries of each CNF sub-formula defined at a given node of the search tree and which is derived from the initial formula by considering the partial assignment corresponding to that node. We adapted Saucy to compute this local symmetry by maintaining dynamically the graph of the sub-formula defined at each node of the search tree. Saucy is called with the graph of the local sub-formula as the main input, and then returns the set of generators of the automorphism group of the graph which is shown to be equivalent to the local symmetry group of the considered sub-formula. The proposed local symmetry detection method is implemented and exploited in the tree search method $LSAT$ to improve its efficiency. Experimental results confirmed that local symmetry breaking is profitable for SAT solving and improves global symmetry breaking on some of the considered problems, and its combination with global symmetry seems to be complementary.

As a future work, we are looking to implement some weakened symmetry conditions under which we may detect more symmetries, then experiment it and compare its results with the ones given here.

Another interesting point is to try to detect local variable symmetries and post dynamic constraints to break them, it will be important to compare the static approaches that detect only global symmetries with this approach.

References

1. F. A. Aloul, A. Ramani, I. L. Markov, and K. A. Sakallak. Symmetry breaking for pseudo-boolean satisfiabilty. *In ASPDAC'04*, pages 884–887, 2004.
2. Fadi A. Aloul, Arathi Ramani, Igor L. Markov, and Karem A. Sakallah. Solving difficult SAT instances in the presence of symmetry. In *Proceedings of the 39th Design Automation Conference (DAC 2002)*, pages 731–736. ACM Press, 2002.
3. B. Benhamou. Study of symmetry in constraint satisfaction problems. *PPCP'94*.
4. B. Benhamou and L. Sais. Theoretical study of symmetries in propositional calculus and application. *In CADE'11*, 1992.
5. B. Benhamou and L. Sais. Tractability through symmetries in propositional calculus. *In JAR*, 12:89–102, 1994.
6. B. Benhamou, L. Sais, and P. Siegel. Two proof procedures for a cardinality based language. *In STACS*, pages 71–82, 1994.
7. Belaïd Benhamou and Mohamed Réda Saïdi. Local symmetry breaking during search in csps. In *In CP*, pages 195–209, 2007.
8. W. Bibel. Short proofs of the pigeon hole formulas based on the connection method. *Automated reasoning*, (6):287–297, 1990.
9. Brown, C. A.Finkelstein, and L. P. W. Purdom. Bactrack searching in the presence of symmetry. *In Algebraic algorithms and error-correcting codes.*, (6):99–110, 1988.
10. D. Cohen, P. Jeavons, C. Jefferson, K.E. Petrie, and B. Smith. Symmetry definitions for constraint satisfaction problems. *In CP*, pages 17–31, 2005.
11. J. Crawford. A theoretical analysis of reasoning by symmetry in first-order logic. *Workshop on Tractable Resonning, AAAI-92*, 1992.
12. P. T. Darga, K. A. Sakallah, and I. L. Markov. Faster symmetry discovery using sparsity of symmetries. In *DAC*, 2008.
13. E.C. Freuder. Eliminating interchangeable values in constraints satisfaction problems. *AAAI-91*, pages 227–233, 1991.
14. I. P. Gent, T. Kelsey, S. A. Linton, J. Pearson, and C. M. Roney-Dougal. Groupoids and conditional symmetry. In *CP*, pages 823–830, 2007.
15. J. W. L. Glaisher. On the problem of the eight queens. *Philosophical Magazine*, 48(4):457–467, 1874.
16. A. Haken. The intractability of resolution. *T.Computer Science*, 39:297–308, 1985.
17. J.Crawford, M. L. Ginsberg, E. Luck, and Amitabha Roy. Symmetry-breaking predicates for search problems. In *KR'96*, pages 148–159. 1996.
18. J.G. Kalbfleisch and R.G. Stanton. On the maximal triangle-free edge-chromatic graphs in three colors. *combinatorial theory*, (5):9–20, 1969.
19. J. De Kleer. A comparison of atms and csp techniques. *IJCAI'89*, pages 290–296.
20. B. Krishnamurty. Short proofs for tricky formulas. *Acta Inf.*, (22):253–275, 1985.
21. R.C. Lyndon. *Notes of logic.* Van Nostrand Mathematical Studies, 1964.
22. B McKay. Practical graph isomorphism. In *Congr. Numer. 30*, pages 45–87, 1981.
23. C. Mears, M. Garcia de la Banda, and M. Wallace. On implementing symmetry detection. In *SymCon'06*, pages 1–8, 2006.
24. R. Ostrowski, B. Mazure, and L. Sais. Lsat solver. In *SAT*, 2002.
25. J. F. Puget. Automatic detection of variable and value symmetries. In *CP'05*, pages 474–488.
26. J.F. Puget. On the satisfiability of symmetrical constraint satisfaction problems. In *proceedings of ISMIS*, pages 350–361, 1993.
27. P. Siegel. Representation et utilisation de la connaissance en calcul propositionnel, 1987. Thèse d'état, GIA - Luminy (Marseille).