Local Symmetry Breaking During Search in CSPs

Belaïd Benhamou and Mohamed Réda Saïdi

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
Belaid.Benhamou@cmi.univ-mrs.fr, saidi@cmi.univ-mrs.fr

Abstract. Many research works on symmetry in CSPs appeared recently. But, most of them deal only with the global symmetry[1] of the studied problem and give no strategy that can be used to detect and eliminate local symmetry[2]. Eliminating global symmetry is shown to be useful, but exploiting only these symmetries could not be sufficient to solve some hard locally symmetrical problems. That is, a problem can have few or no initial symmetries and become very symmetrical at some nodes during the search. In this paper we study a general principle of semantic symmetry and define a syntactic symmetry which is a sufficient condition for semantic symmetry. We define a weakened form of this syntactic symmetry, and show how to detect and how to eliminate it locally to increase CSP tree search methods efficiency. Experiments confirm that local symmetry breaking is profitable for CSP solving.

1 Introduction

As far as we know, the principle of symmetry is first introduced by Krishnamurty [1] to improve resolution in propositional logic. Symmetries for Boolean constraints are studied in depth in [2,3,4], the authors showed how to detect them and proved that their exploitation is a real improvement for several automated deduction algorithms. The notion of interchangeability in CSPs is introduced in [5] and symmetry for CSPs are studied in [6,7].

Since that, many research works on symmetry have appeared. For instance, the static approach used by James Crawford et al. in [8] for propositional logic theories consists in adding constraints to break the global symmetries of the problem. This technique has been improved in [9] and extended to 0-1 Integer Logic Programming in [10].

Since a great number of constraints could be added, 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

[1] The symmetry of the initial problem appearing at the root of the search tree.
[2] The symmetry of the resulting CSP at a node of the search tree corresponding to a partial instantiation.

C. Bessiere (Ed.): CP 2007, LNCS 4741, pp. 195–209, 2007.

in case of backtracking. In [14,15,16,17], authors proposed to use each subtree as a no-good to avoid exploration of some symmetric interpretations and the group equivalence tree conceptual for value symmetry elimination is introduced in [18]. These techniques are called respectively SBDS, SBDD and GE-Tree.

Recently, a method which breaks symmetries between the variables of an Alldiff constraint is studied in [19], a nice method which eliminates all value symmetries in surjection problems is given in [20], and a work gathering all the known different symmetry definitions to *solution symmetry* or to *constraint symmetry* is done in [21]. More recently, in [22], Puget studied a new lex constraints symmetry breaking method in the term of dynamic lex leader solutions, and in [23] Walsh studied various new propagators to break various symmetries among them the one acting simultaneously on both variables and values.

One drawback of all these approaches is that only the symmetry of the initial problem is considered (the global symmetry) and no method that allows dynamic detection and exploitation of local symmetry is given. Recently, researchers called this, conditional symmetry [24,25].

In this paper we developed the general concept of semantic symmetry for CSPs that Benhamou first initiated in [7]. We also study and extend the principle of syntactic symmetry that we prove to be a sufficient condition for semantic symmetry. We show how local symmetry is detected and eliminated during search, and show how its removal simplifies the search space of tree search algorithms.

This paper is organized as following: CSP background is given in Section 2. *Semantic symmetry* is defined in Section 3. Section 4 discusses the notion of *syntactical symmetry* which is a sufficient condition for *semantic symmetry*. Section 5 shows how symmetry is detected and eliminated locally during search and how a tree search method (here Forward Checking) takes advantage of symmetrical values to reduce its search space. In section 6 we evaluate the proposed techniques by experimental results and Section 7 concludes the work.

2 Background

A CSP is a quadruple $\mathcal{P} = (V, D, C, R)$ where: $V = \{v_1, ..., v_n\}$ is a set of n variables; $D = \{D_1, \ldots, D_n\}$ is the set of finite discrete domains associated to the CSP variables, D_i includes the set of possible values of the CSP variable v_i, d_i denotes the fact that the value d belongs to the domain D_i, $C = \{C_1, ..., C_m\}$ is a set of m constraints each involving a subset of the CSP variables. A binary constraint is a constraint which involves at most two variables v_i, v_j, and is denoted by C_{ij}; $R = \{r_1, ..., r_m\}$ is a set of relations corresponding to the constraints of C. r_i represents the list of value tuples permitted by the constraint $C_i \in C$. A binary CSP $\mathcal{P}$ (a CSP involving only binary constraints) can be represented by a constraint graph $G(V, E)$ where the set of vertices V is the set of the CSP variables and each edge $(v_i, v_j) \in E$ connects the variables v_i and v_j involved in the constraint $C_{ij} \in C$. The microstructure [5,26,21] of the CSP $\mathcal{P}$ is a graph $\mathcal{M}_{\mathcal{P}}(V \times \cup_{i\in[1,n]} D_i, \acute{E})$, where each edge of $\acute{E}$ corresponds either to a tuple allowed by a specific constraint or to an

allowed tuple because there is no constraints between the associated variables. An instantiation $\mathcal{I} = (\langle v_1, a_1\rangle, \langle v_2, a_2\rangle, \ldots, \langle v_n, a_n\rangle)$ is the variable assignment $\{v_1 = a_1, v_2 = a_2, \ldots, v_n = a_n\}$ where a value a_i of the domain D_i is assigned to the variable v_i . A constraint $C_i \in C$ is satisfied by $\mathcal{I}$ if the projection of $\mathcal{I}$ on the variables involved in C_i is a tuple of r_i. The instantiation $\mathcal{I}$ is consistent if it satisfies all the constraints of C, thus $\mathcal{I}$ is a solution of the CSP. An instantiation of a subset of the CSP variables V is called a partial instantiation, it defines a nogood when it is inconsistent. Each partial instantiation $\mathcal{I}$ defines a node $n_{\mathcal{I}}$ in the search tree which corresponds to the local CSP $\mathcal{P}_{\mathcal{I}}$ resulting from $\mathcal{P}$ by considering $\mathcal{I}$ and its induced propagations. An instantiation is total if it is defined on all the CSP variables. Given a CSP, the main question is to decide its consistency or to find its set of solutions. We assume that the reader knows a minimal background on permutations and groups. For the sake of simplicity we restricted the study to binary CSPs, however, the notion of symmetry remains valuable for non-binary CSPs as well.

3 Semantic Symmetry

Because we are interested in two problems in CSPs: the problem of finding a solution and the problem of finding all the solutions of the CSP, we define two levels of semantic symmetry.

Definition 1 (Semantic symmetry for consistency). *Two variable-value pairs $\langle v_i, b_i\rangle \in V \times D_i$ and $\langle v_j, c_j\rangle \in V \times D_j$ are symmetrical for consistency iff the following assertions are equivalent:*

1. *There is a solution of the CSP which assigns the value b_i to the variable v_i;*
2. *There is a solution of the CSP which assigns the value c_j to the variable v_j.*

Variable-value pairs can be not only symmetrical for consistency, but symmetrical for the set of all solutions as well. Thus, if $sol(\mathcal{P})$ denotes the set of solutions of the CSP $\mathcal{P}$, then we define a second level of semantic symmetry as follows:

Definition 2 (Semantic symmetry for all solutions). *Two variable-value pairs $\langle v_i, b_i\rangle \in V \times D_i$ and $\langle v_j, c_j\rangle \in V \times D_j$ are symmetrical for $sol(\mathcal{P})$ if and only if each solution of the CSP assigning the value b_i to v_i can be mapped into a solution assigning the value c_j to v_j and vice-versa.*

This means that the set of solutions in which the assignment $v_i = b_i$ participates is isomorphic to the one in which $v_j = c_j$ participates. These are symmetrical solutions.

Remark 1.
1. If the variables v_i and v_j designate a same variable, then both previous definitions concern symmetry of values of a same domain.
2. Symmetry for all solutions implies symmetry for consistency.

Identifying semantic symmetry is clearly time consuming, since this requires solving the problem. We study in the next section the syntactical symmetry notion which is more tractable computationally and which is a sufficient condition to handle semantic symmetry.

4 Syntactic Symmetry

In [7], the author studied syntactical symmetry of values of a same CSP domain variable, here syntactic symmetry is extended to the possible variable-value pairs of the CSP. This leads to a similar definition as the one of constraint symmetry given in [21]

Definition 3. *A syntactical symmetry of a CSP* $\mathcal{P} = (V, D, C, R)$ *having the microstructure* $\mathcal{M}_\mathcal{P}$*, is a mapping* $\sigma : V \times \cup_{i\in[1,n]} D_i \longrightarrow V \times \cup_{i\in[1,n]} D_i$*, that preserves the edges and the non-edges of* $\mathcal{M}_\mathcal{P}$*.*

Remark 2. A syntactical symmetry of a CSP $\mathcal{P}$ is an automorphism of its microstructure $\mathcal{M}_\mathcal{P}$. The set of syntactic symmetries of a CSP $\mathcal{P}$ is identical to the set of its *constraint symmetries* [21] which is equivalent to the automorphism group $Aut(\mathcal{M}_\mathcal{P})$ of its microstructure. Syntactical symmetries preserve the solutions of the CSP.

Definition 4. *Two variable-value pairs* $\langle v_i, b_i\rangle \in V \times D_i$ *and* $(v_j, c_j) \in V \times D_j$ *are syntactically symmetrical iff there exists a syntactical symmetry* σ *of* $\mathcal{P}$*, such that* $\sigma(\langle v_i, b_i\rangle)=\langle v_j, c_j\rangle$*.*

Theorem 1. *If two variable-value pairs* $\langle v_i, b_i\rangle \in V \times D_i$ *and* $\langle v_j, c_j\rangle \in V \times D_j$ *are syntactically symmetrical, then they are semantically symmetrical for all solutions of the CSP.*

Proof. It is based on the fact that syntactical symmetry preserves solutions.

4.1 The Weakened Syntactic Symmetry Conditions

A weakened symmetry condition has been defined in [27,28] for the restricted framework of Not-equals CSPs and had been shown to be useful in practice. Here, we show how to extend this weakened condition to General CSPs. Before doing that, we define the notion of assignment trees and failure trees corresponding to the enumerative search method used to prove the consistency of a considered CSP.

Definition 5. *We call an assignment tree of a CSP* $\mathcal{P}$ *corresponding to a given search method and a fixed variable ordering, a tree which gathers the history of all the variable assignments made during search, where the nodes represent the variables of the CSP and where the edges outgoing from a node* v_i *are labeled by the different values used to instantiate the corresponding CSP variable* v_i*.*

The root of the tree is the first variable in the ordering. In this work, the considered search method is Forward Checking [29].

In an assignment tree of a CSP, a path connecting the root of the tree to a node defines a partial instantiation $\mathcal{I}$ of the CSP. The variables of the partial instantiation $\mathcal{I}$ are the nodes of the considered path. The last node $n_\mathcal{I}$ of the path corresponds to the last affected variable in the instantiation or to a variable having an empty domain.

We associate to each inconsistent partial instantiation, corresponding to a given path in the assignment tree, a failure tree defined as follows:

Definition 6. *Let T be an assignment tree of the CSP $\mathcal{P}$, $\mathcal{I} = (\langle v_1, a_1\rangle, \langle v_2, a_2\rangle, \ldots, \langle v_i, a_i\rangle)$ an inconsistent partial instantiation of the variables v_1,v_2,...,v_i corresponding to the path $\{v_1, v_2, ..., v_i\}$ in T. We call a failure tree of the instantiation $\mathcal{I}$, the sub-tree of T denoted by $T_{\mathcal{I}}$ such that:*

1. *The root of the tree T and the root of the sub-tree $T_{\mathcal{I}}$ are joined by the path corresponding to the instantiation $\mathcal{I}$;*
2. *All the CSP variables corresponding to the leaf nodes of $T_{\mathcal{I}}$ have empty domains.*

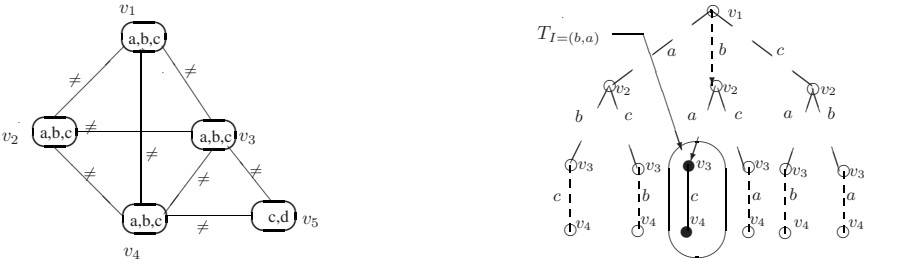

Fig. 1. The constraint graph of a graph coloring instance, its assignment tree and the failure tree of $\mathcal{I}$=($\langle v_1, b\rangle, \langle v_2, a\rangle$)

Example 1. Take the CSP on the left part of Figure 1 and apply a Forward Checking process on it w.r.t the variable ordering $\{v_1, v_2, v_3, v_4, v_5\}$. The right part of Figure 1 illustrates the assignment tree of the considered CSP. If we take the partial instantiation $\mathcal{I} = (\langle v_1, b\rangle, \langle v_2, a\rangle)$, then the failure tree $T_{\mathcal{I}}$ of the instantiation $\mathcal{I}$ is the part of the assignment tree shown in a box.

We can now give the weakened conditions of syntactic symmetry. The main idea is to weaken the syntactic symmetry conditions when an inconsistent partial instantiation is generated during the search. That is, for a local CSP $\mathcal{P}_{\mathcal{I}}$ where the partial instantiation $\mathcal{I}$ is inconsistent, the conditions of syntactic symmetry (Definition 3) are restricted to only the variables involved in the failure tree $T_{\mathcal{I}}$, rather than to all the un-instantiated variables.

Theorem 2. *Let $\mathcal{P}(V, C, D, R)$ be a CSP, $\mathcal{I}_0 = (\langle v_1, a_1\rangle, \ldots, \langle v_{i-1}, a_{i-1}\rangle)$ a partial instantiation of $i-1$ variables instantiated before the current variable v_i such that the extension $\mathcal{I} = \mathcal{I}_0 \bigcup \{\langle v_i, a_i\rangle\}$ is inconsistent, $T_{\mathcal{I}}$ is the failure tree of $\mathcal{I}$ and $Var(T_{\mathcal{I}})$ the set of the variables corresponding to the nodes of $T_{\mathcal{I}}$. If $\langle v_i, a_i\rangle$ is syntactically symmetrical to $\langle v_j, b_j\rangle$ in the CSP $\mathcal{P}_{\mathcal{I}_0}$ derived from $\mathcal{P}_{\mathcal{I}_0}$ by restricting its set of variables to $Var(T_{\mathcal{I}}) \cup \{v_i\}$, then the extension $\mathcal{J} = \mathcal{I}_0 \bigcup \{\langle v_j, b_j\rangle\}$ is inconsistent.*

Proof. The CSP $\acute{\mathcal{P}}_{\mathcal{I}_0}$ is $\mathcal{P}_{\mathcal{I}_0}$ where the set of variables is restricted to $Var(T_{\mathcal{I}}) \cup \{v_i\}$. By the hypothesis, $T_{\mathcal{I}}$ is a failure tree of $\mathcal{I}$ in $\mathcal{P}$. This implies that the assignment of the value a_i to v_i leads to a failure in $\acute{\mathcal{P}}_{\mathcal{I}_0}$. In other words, $\langle v_i, a_i\rangle$ does not participate in any solution of $\acute{\mathcal{P}}_{\mathcal{I}_0}$. By the hypothesis, the pair $\langle v_i, a_i\rangle$ is symmetrical to $\langle v_j, b_j\rangle$ in $\acute{\mathcal{P}}_{\mathcal{I}_0}$. This implies that $\langle v_j, b_j\rangle$ does not participate in any solution of $\acute{\mathcal{P}}_{\mathcal{I}_0}$. Thus $\langle v_j, b_j\rangle$ does not participate in any solution of $\mathcal{P}_{\mathcal{I}_0}$ as well. This implies that the partial instantiation $\mathcal{J} = \mathcal{I}_0 \bigcup \{\langle v_j, b_j\rangle\}$ is inconsistent in $\mathcal{P}$. (QED)

Remark 3. In the case of a failure during search, the symmetry conditions are restricted to only the variables participating in the failure.

Theorem 2 gives an interesting weakening of the conditions of syntactic symmetry that we can use when the current partial instantiation leads to an inconsistency. By using the new conditions, some symmetries not captured by the normal conditions can result from these weakened conditions. Let us consider for instance the CSP of Figure 1. If we take the partial instantiation $\mathcal{I}_0 = \langle v_1, b\rangle$ and the inconsistent extension $\mathcal{I}$=$\mathcal{I}_0 \cup \{\langle v_2, a\rangle\}$, then the pairs $\langle v_2, a\rangle$ and $\langle v_2, c\rangle$ are symmetrical. That is, the two values a and c of the domain of the current variable v_2 are symmetrical *w.r.t* the weakened conditions (Theorem 2) applied on the CSP $\acute{\mathcal{P}}_{\mathcal{I}_0}$ involving the set of variables $Var(T_{\mathcal{I}}) \cup v_2$=$\{v_2, v_3, v_4\}$, whereas the normal symmetry conditions are not verified on the CSP $\mathcal{P}_{\mathcal{I}_0}$ involving the set of all un-instantiated variables $\{v_2, v_3, v_4, v_5\}$. The branch corresponding to the assignment of c to v_2 is not explored during search thanks to Theorem 2. This defines a more powerfull symmetry cut that we use to shorten CSP search trees.

Besides, this weakening property can be used in other known symmetry breaking methods [2,3,4]. That is, symmetry conditions have to be checked only on the variables involved in the failure when a partial instantiation is shown to be inconsistent.

Below we show how local symmetry is detected and eliminated, and how a tree search method (Forward Checking) can take advantage of symmetry.

5 Local Symmetry Detection and Exploitation

5.1 Symmetry Detection and Breaking

Local symmetries have to be detected dynamically at each node of the search tree. Dynamic symmetry detection had been studied in CSPs, a local syntactic domain symmetry search method had been given in [7].

As an alternative to this symmetry search method, we adapted Saucy [9] to detect 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 [30] or the most recent methods AUTOM [31] or the one described in [32] can be adapted to search local symmetry. It is shown in [31] that AUTOM is the best method. Because the source code of AUTOM is not

free, we chose Saucy. Since the syntactic symmetry group of a CSP $\mathcal{P}$ is identical to the automorphism group of its microstructure $\mathcal{M}_{\mathcal{P}}$, we can use Saucy on $\mathcal{M}_{\mathcal{P}}$ to detect the syntactic symmetry group of $\mathcal{P}$. 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 microstructure such that, a node is allowed to be permuted with another node if they have the same color. This restricts the permutations to the nodes having the same color. We use this coloration possibility to guide the symmetry search and detect local value symmetries. The source code of Saucy can be found at (http://vlsicad.eecs.umich.edu/BK/SAUCY/).

Symmetry detection: Consider a CSP $\mathcal{P}$, and a partial instantiation $\mathcal{I}$ of $\mathcal{P}$, defining a state in the search corresponding to the current node $n_{\mathcal{I}}$. The main idea is to maintain dynamically the microstructure $\mathcal{M}_{\mathcal{P}_{\mathcal{I}}}$ of the CSP $\mathcal{P}_{\mathcal{I}}$ corresponding to the local sub-problem defined at each current node $n_{\mathcal{I}}$, then color the microstructure $\mathcal{M}_{\mathcal{P}_{\mathcal{I}}}$ and compute its automorphism group $Aut(\mathcal{M}_{\mathcal{P}_{\mathcal{I}}})$. The CSP $\mathcal{P}_{\mathcal{I}}$ can be viewed as a new problem corresponding to the unsolved part. Computing all the automorphisms of the dynamic microstructure at each node of the search tree may be expensive. To remedy this, two coloration strategies of the microstructure are considered:

1. **The multi-colors-strategy:** A first compromise is to limit permutations to only values of the same domains. To do that, a color is associated to each variable. Every node of the microstructure belonging to a variable is colored with the same color. Now by applying Saucy on this colored microstructure we can get the generator set Gen of the symmetry sub-group existing between values of the same domains of the CSP.
2. **The two-colors-strategy:** A second compromise is to associate to the current variable v_i (under instantiation) one color and all the other variables another color. That is, we color the dynamic microstructure $\mathcal{M}_{\mathcal{P}_{\mathcal{I}}}$ with two colors. All the nodes of the microstructure belonging to the current variable v_i have the first color and all the other nodes the second one. Finally apply Saucy to compute the generators of the automorphism sub-group corresponding to this coloration. This returns the generators Gen of the symmetry group allowing variable-value permutations on the other variables different from v_i, but the values of v_i are permuted together.

Remark 4. The total local symmetry group is reached when using only one color on the microstructure $\mathcal{M}_{\mathcal{P}_{\mathcal{I}}}$.

Symmetry elimination: We use Theorem 2 to prune search spaces of tree search methods. Indeed, if the assignment $\langle v_i, b_i \rangle$ of the current variable v_i at a given node $n_{\mathcal{I}}$ of the search tree is shown to participate in no solution of the CSP $\mathcal{P}$, then all the pairs $\langle v_j, c_j \rangle$ which are symmetrical to $\langle v_i, b_i \rangle$ in $\mathcal{P}_{\mathcal{I}}$ do not (i.e. these pairs are the ones corresponding to the orbit of the conflicted pair $\langle v_i, b_i \rangle$ that can be computed by using only the symmetry group generators). Then we remove the value c_j from the domain of the un-instantiated variable v_j, and prune the sub-space which corresponds to its assignment to v_j in the search tree.

If the variable v_i and v_j are the same (as in our implementation), then the previous reasoning handles symmetries between values of the domain D_i. The domain D_i of the current variable v_i is partitioned into sub-sets of symmetrical values *w.r.t* the detected local symmetries at the corresponding node of the search tree. To avoid generating local symmetrical solutions, we consider one value from each sub-set of symmetrical values in D_i. If we need to check CSP consistency only, we stop the search when a first solution is found.

5.2 Symmetry Advantage in Tree Search Algorithms

Now we are in the position to show how these symmetrical values can be used to increase the efficiency of CSP tree search algorithms. We choose in our implementation the Forward Checking method to be the baseline method that we want to improve by local symmetry elimination. The resulting procedure called FC-sym is given in Figure 2.

If $\mathcal{I}$ is an inconsistent partial instantiation in which the assignment $\langle v_i, d_i \rangle$ of the current variable v_i is shown to participate in no solutions of the CSP $\mathcal{P}$, then according to Theorem 2, all the pairs $\langle v_j, d_j \rangle$ which are symmetrical to $\langle v_i, d_i \rangle$ in $\mathcal{P}_{\mathcal{I}}$ do not. Thus we remove d_j from the domain of v_j, and prune the sub-space which corresponds to its assignment to v_j.

The function $orbit(\langle v_{k+1}, d_{k+1} \rangle, Gen)$ is elementary, it computes the orbit of the pair $\langle v_{k+1}, d_{k+1} \rangle$ 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 classical problems to show the local symmetry behavior in CSP resolution. We expect that symmetry breaking will be more profitable in real-life applications. Here, we tested and compared four methods:

1. **No-sym:** search without symmetry breaking;
2. **Global-sym:** search with global symmetry breaking restricted to values of a same domain. The same symmetries as the ones considered in the GE-tree method, with a slight difference that we break only global symmetries between values of a same domain;
3. **Local-sym1:** search with local value symmetry breaking (the weakened symmetry). This method implements the multi-colors strategy (see Section 5.1).
4. **Local-sym2:** search with restricted local variable-value symmetry breaking (the weakened symmetry). This method implements the two-colors strategy (see Section 5.1).

on different problems: random graph coloring problems, Dimacs graph coloring instances and n-Queens problems. An implementation of the $Local-sym1$ strategy in GECODE system is successfully used in [33] to break local symmetry in the subgraph pattern matching problem. The common baseline search method

```
Procedure FC-sym(D, I, k);{I = [⟨v1, d1⟩, ⟨v2, d2⟩, ..., ⟨vk, dk⟩]}
begin
    if k = n then I is a solution, return(I)
    else
    begin
      for each v_i ∈ V, such that C_ik ∈ C, v_i ∈future(v_k) do
        for each value d_i ∈ D_i do
            if (d_i, d_k) ∉ r_ik then
              delete d_i from D_i;
      if ∀v_i ∈ future(v_k), D_i ≠ ∅ then
      begin
        v_{k+1}=next-variable(v_k)
        repeat
            take d_{k+1} ∈ D_{k+1}
            D_{k+1} = D_{k+1} − {d_{k+1}}
            I=I ∪ {⟨v_{k+1}, d_{k+1}⟩};
            J=FC-sym(D, I, k + 1);
            I=I-{⟨v_{k+1}, d_{k+1}⟩};
            if J ∈ Sol(P) then Gen=Saucy(P_I);
            else Gen=Saucy(P_{Var(T_I)∪v_{k+1}});
            SymClass(⟨v_{k+1}, d_{k+1}⟩)=orbit(⟨v_{k+1}, d_{k+1}⟩,Gen);
            D_{k+1}=D_{k+1}-SymClass(⟨v_{k+1}, d_{k+1}⟩)
        until D_{k+1} = ∅
      end
    end
end;
```

Fig. 2. Forward Checking method with symmetry

for the four previous methods is Forward Checking. The complexity indicators are the number of nodes of the search tree and the CPU time. The time needed for computing symmetries is added to the total CPU time. The source codes are written in C and compiled on a Pentium 4, 2.8 GHZ and 1 Go of RAM.

6.1 Random Graph Coloring Problems

Random graph coloring problems are generated with respect to the following parameters: (1) n : the number of vertices (variables), (2) $Colors$: the number of colors (domain values) 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 constraint 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.

Figure 3, shows the performances of the four methods in number of nodes of the search tree, respectively, in CPU time (in seconds) on random graph coloring problems, whose number of variables is fixed to $n = 15$ and the density to $d = 0.9$. We reported here experiments on instances having hight density, because they

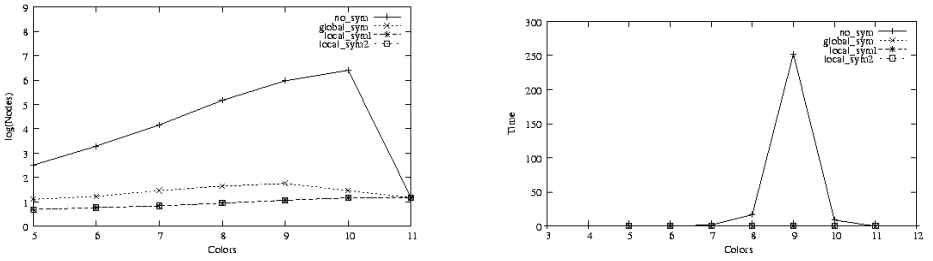

Fig. 3. Node and Time curves where $n = 15$ and $d = 0.9$

are the hardest instances, and symmetry presents a similar behavior for average and weak density instances. The curves on the left are plotted in a logarithmic scale, they represent the performances in number of nodes *w.r.t* the number of colors. The ones on the right are plotted in the usual scale and express the performances in CPU time *w.r.t* the number of colors. As expected, we can see that all the methods exploiting symmetry outperform dramatically the search without symmetry (No-sym) in both the number of nodes and the CPU time. We can also see on the node curves that local symmetry elimination (Local-sym1 and Local-sym2) reduces more the search tree than global symmetry elimination (Global-sym). That is, local symmetries are more frequent during the search than the global symmetries stabilizing the partial instantiation. Both Local-sym1 and Local-sym2 have the same behavior in number of nodes; their node curves are almost identical. We can distinguish on the CPU time curve of No-sym a critical region where the instances are harder. All the methods using symmetry solved these instances in less than 0.1 seconds, then their CPU time curves are confused with x-axis and do not appear.

Since Figure 3 does not allow a CPU time comparison of the methods exploiting symmetry, we reported in Figure 4 the practical results of the methods: Global-sym, Local-sym1 and Local-sym2, on the random graph coloring problem where the number of variables is increased to $n = 35$ and where we keep the same density ($d = 0.9$) as in Figure 3.

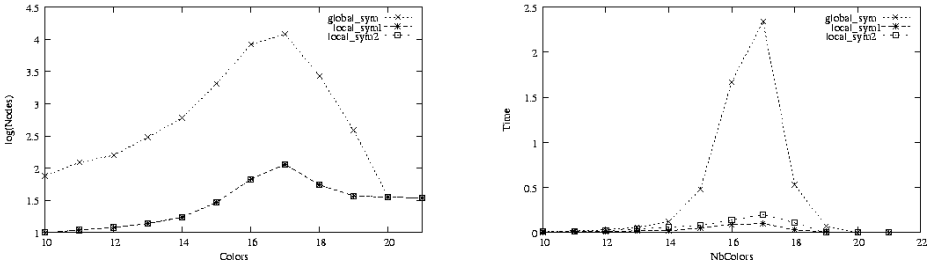

Fig. 4. Node and Time curves of the three symmetry methods on random graph coloring where $n = 35$ and $d = 0.9$

We can see on the node curves (the curves on the left plotted in a logarithmic scale) that both Local-sym1 and Local-sym2 detect and eliminate more symmetries than the Global-sym method. The reason is that the local symmetry detected at a node during the search by both Local-sym1 and Local-sym2 includes the global symmetry stabilizing the partial instantiation at that node exploited by Global-sym. The node curves of both Local-sym1 and Local-sym2 compare well. From the CPU time curves (the curves on the right), we can see that both Local-sym1 and Local-sym2 are faster than Global-sym. Near the peak of difficulty, Local-sym2 seems to be 12 times faster than Global-sym and Local-sym1 about 24 times faster than Global-sym. Therefore, Local symmetry elimination is profitable for solving random graph coloring instances in the hard region and outperforms dramatically global symmetry breaking on these problems. We can also see on the CPU time that Local-sym1 improves Local-sym2, thus the good compromise looks to be the multi-color strategy corresponding to domain value symmetry. These remarks will be confirmed by the experiments on Dimacs benchmarks in the next section.

6.2 Dimacs Graph Coloring Benchmarks

Here, we tested and compared the four methods on some graph coloring benchmarks taken from the Dimacs challenge (http://mat.gsia.cmu.edu/COLOR04/).

Table 1 shows the results of the methods on some of the benchmarks. It gives the instance, the chromatic number found (k), the number of nodes of the search tree and the CPU time for each method. We seek for each instance the minimal number k of colors needed to color the vertices of the corresponding graph (called the chromatic number). The search of the chromatic number consists in proving the consistency of the problem with k colors (the existence of a k-coloration of the graph); and in proving its inconsistency when using $k-1$ colors (not colorable). The symbol "-" means that the corresponding method does not solve the instance in one hour.

Table 1. Results on some Dimacs graph coloring benchmarks

		No-sym		Global-sym		Local-sym1		Local-sym2	
Instance	k	*Nodes*	*Time*	*Nodes*	*Times*	*Nodes*	*Time*	*Nodes*	*Time*
myciel4	5	30,976	0.16	2,764	0.03	1,260	0.01	1,260	0.04
myciel5	6	-	-	8,040,259	59.84	2,413,556	22.21	2,406,945	25.36
anna	11	-	-	3,403	0.59	168	0.05	168	0.08
david	11	-	-	3,896	0.23	124	0.03	124	0.03
queen7_7	7	2,452	0.01	513	0.02	502	0.01	502	0.0
queen8_8	9	-	-	10,629,131	262.54	1,399,436	29.7	1,396,774	30.16
school1	14	-	-	-	-	76,192	17.28	75,985	17.85
school1_nsh	14	-	-	-	-	1,487,287	257.57	1,486,523	270.4
2-Insertion_3	4	832,150	1.02	277,408	0.73	135,953	0.48	115,737	0.52
2-FullIns_3	5	2,294,396	7.63	193,347	1.14	49,202	0.59	48,076	0.65
mugg88_1	4	-	-	-	-	2,882,284	53.91	2,882,284	93.55
mugg88_25	4	-	-	-	-	881,784	6.74	881,784	9.4
mugg100_1	4	-	-	-	-	3,325,453	24.85	3,325,453	40.15
mugg100_25	4	-	-	-	-	2,727,178	17.3	2,727,178	30.92
zeroin.i.1	49	-	-	-	-	268	7.0	268	35.49
zeroin.i.2	30	-	-	-	-	262	0.75	262	3,675
zeroin.i.3	30	-	-	-	-	262	0.76	262	3,675
mulsol.i.2	31	-	-	-	-	237	0.85	237	10.14
mulsol.i.3	31	-	-	-	-	237	0.9	237	10.14
le450_5a	5	178,753	13.88	170,123	13.75	167,787	32.0	167,703	32.23
le450_5b	5	1,349	0.11	1,110	0.09	927	0.11	927	0.19
le450_5c	5	1,984	0.15	1,984	0.17	1,983	0.31	1,975	0.34
le450_5d	5	5,795	0.54	4,563	0.34	3,452	0.62	3,452	0.68
DSJC125.1	5	55,358	0.85	43,773	1.34	40,809	1.44	40,809	1.48

Table 1 shows that both No-sym and Global-sym are not able to solve several instances under the time limit, but Global-sym is better than No-sym in both numbers of nodes and CPU time on these problems. We can see that both Local-sym1 and Local-sym2 are in general better than Global-sym in both the number of nodes and the CPU time. That is, local symmetry is more profitable than global symmetry on these problems. We can also see that Local-sym1 eliminates the same symmetries as the ones eliminated by Local-sym2, but Local-sym1 is faster than Local-sym2. This confirms that eliminating local domain value symmetries by using the multi-color strategy implemented in Local-sym1 is the best compromise on these problems.

6.3 The n-Queens Problems

Finding *all* solutions of the n-queens problem is still a challenge. We compared the four methods on some instances of this problem.

Table 2 summarizes the results obtained. For each method we give the number of computed solutions ($Sols$), the number of nodes, and the CPU time in seconds. Note that for the methods exploiting symmetry, the number of solutions ($Sols$) is the number of non-symmetrical solutions found *w.r.t* the applied symmetry breaking strategy. The number of solutions of No-sym is the total set of solutions of the problem. We can see that both Local-sym1 and Local-sym2 represent the set of solutions slightly in a more compact way than Global-sym. This means that Local-sym1 and Local-sym2 compact some local symmetrical solutions in addition to the global symmetrical ones compacted by Global-sym. We can also see that Local-sym2 detects some local symmetrical solutions which are not considered by Local-sym1. This is due to some variable-value symmetries considered in Local-sym2 that detect some local symmetrical solutions which are not detected in Local-sym1. Now, if we compare globally the methods in number of solutions, in the number of nodes and in CPU time, the method Global-sym seems to be the best on the average. Indeed, global symmetry is sufficient to solve efficiently the n-queens problems and local value symmetry does not abound like

Table 2. Results on the n-queens problem

	No-sym			Global-sym		
n	$Sols$	$Nodes$	$Time$	$Sols$	$Nodes$	$Times$
8	92	1,360	0.0	46	680	0.01
9	352	5,399	0.0	179	2,800	0.0
10	724	19,744	0.03	362	9,872	0.03
11	2,680	85,939	0.1	1,382	43,958	0.07
12	14,200	416,828	0.28	7,100	208,414	0.25
13	73,712	2,154,845	2.69	37,361	1,093,606	1.99
14	365,596	11,799,746	46.95	51,726	5,899,873	20.65
	Local-sym1			Local-sym2		
n	$Sols$	$Nodes$	$Time$	$Sols$	$Nodes$	$Times$
8	45	664	0.01	45	662	0.01
9	172	2,645	0.02	168	2,625	0.05
10	355	9,656	0.07	353	9,640	0.08
11	1,309	42,154	0.25	1,305	42,078	0.31
12	6,883	204,901	2.05	6,839	203,611	2.19
13	35,525	1,055,366	11.44	35,312	1,053,053	11.58
14	44,334	5,777,244	69.6	43,257	5,765,594	75.6

in the graph coloring. We believe that local variable symmetry will be more profitable for n-queens.

7 Discussion and Conclusions

Here, we extended symmetry detection and elimination to local symmetry. That is, the symmetries of each sub-CSP defined at a given node of the search tree and which is derived from the initial CSP by considering the partial instantiation corresponding to that node. We adapted Saucy to compute this local symmetry by maintaining dynamically the microstructure of the sub-CSP defined at each node of the search tree. Unlike the methods using GAP tools, here local symmetry detection is fully automated. Saucy is called with the microstructure of the local sub-CSP as the input graph, and then return the set of generators of the automorphism group of the microstructure which is shown to be equivalent to the local symmetry group of the considered sub-CSP. Detecting and exploiting all the local symmetry groups of the different nodes generated during the search may be time consuming. To remedy this, we proposed two coloration strategies in order to guide and restrict the symmetry search to domain value permutations (the multi-color strategy) and to some restricted variable-value permutations (the two color strategy). Both local symmetry strategies are implemented and exploited in the tree search method FC to prove either CSP consistency or to compute the not-local symmetrical solutions of the CSP. Experimental results confirmed that local symmetry breaking is profitable for CSP solving and improves global symmetry breaking in most of the considered problems.

As a future work, we are looking to implement a one-color symmetry detection strategy, then experiment it and compare it with the two strategies studied in this work.

An other interesting point, is to extend our approach to variable local symmetry breaking. One can try to detect local variable symmetries and post dynamic constraints to break them, it will be important to consider the possibilities of combining local variable and local value symmetries.

Finally, we are interested to adapt our symmetry results for other look-ahead CSP methods like MAC, and export local symmetry breaking to other research domains like biology or operational research to tackle real life applications.

References

1. Krishnamurty, B.: Short proofs for tricky formulas. Acta informatica 22, 253–275 (1985)
2. Benhamou, B., Sais, L.: Theoretical study of symmetries in propositional calculus and application. In: Eleventh International Conference on Automated Deduction, Saratoga Springs, NY, USA (1992)
3. Benhamou, B., Sais, L.: Tractability through symmetries in propositional calculus. Journal of Automated Reasoning (JAR) 12, 89–102 (1994)

4. Benhamou, B., Sais, L., Siegel, P.: Two proof procedures for a cardinality based language. In: Enjalbert, P., Mayr, E.W., Wagner, K.W. (eds.) STACS 94. LNCS, vol. 775, pp. 71–82. Springer, Heidelberg (1994)
5. Freuder, E.: Eliminating interchangeable values in constraints satisfaction problems. In: Proc AAAI-91, pp. 227–233 (1991)
6. Puget, J.F.: On the satisfiability of symmetrical constrained satisfaction problems. In: Komorowski, J., Raś, Z.W. (eds.) ISMIS 1993. LNCS, vol. 689. Springer, Heidelberg (1993)
7. Benhamou, B.: Study of symmetry in constraint satisfaction problems. In: Borning, A. (ed.) PPCP 1994. LNCS, vol. 874. Springer, Heidelberg (1994)
8. Crawford, J., Ginsberg, M.L., Luck, E., Roy, A.: Symmetry-breaking predicates for search problems. In: KR'96. Principles of Knowledge Representation and Reasoning, pp. 148–159. Morgan Kaufmann, San Francisco, California (1996)
9. Aloul, F.A., Ramani, A., Markov, I.L., Sakallak, K.A.: Solving difficult sat instances in the presence of symmetry. IEEE Transaction on CAD 22(9), 1117–1137 (2003)
10. Aloul, F.A., Ramani, A., Markov, I.L., Sakallak, K.A.: Symmetry breaking for pseudo-boolean satisfiabilty. In: ASPDAC'04, pp. 884–887 (2004)
11. Backofen, R., Will, S.: Excluding symmetries in constraint-based search. In: Jaffar, J. (ed.) Principles and Practice of Constraint Programming – CP'99. LNCS, vol. 1713. Springer, Heidelberg (1999)
12. Gent, I.P., Smith, B.M.: Symmetry breaking during search in constraint programming. In: Proceedings ECAI'2000 (2000)
13. Gent, I., Harvey, W., Kelsey, T.: Groups and constraints: Symmetry breaking during search. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, pp. 415–430. Springer, Heidelberg (2002)
14. Focacci, F., Milano, M.: Global cut framework for removing symmetries. In: Walsh, T. (ed.) CP 2001. LNCS, vol. 2239, pp. 77–82. Springer, Heidelberg (2001)
15. Fahle, T., Schamberger, S., Sellmann, M.: Symmetry breaking. In: Walsh, T. (ed.) CP 2001. LNCS, vol. 2239, pp. 93–108. Springer, Heidelberg (2001)
16. Puget, J.: Symmetry breaking revisited. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, pp. 446–461. Springer, Heidelberg (2002)
17. Gent, I.P., Hervey, W., Kesley, T., Linton, S.: Generic sbdd using computational group theory. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833. Springer, Heidelberg (2003)
18. Roney-Dougal, C.M., Gent, I.P., Kelsey, T., Linton, S.A.: Tractable symmetry breaking using restricted search trees. In: Proceedings of ECAI'04, pp. 211–215 (2004)
19. Puget, J.: Breaking symmetries in all diffrent problems. In: Proceedings of IJCAI, pp. 272–277 (2005)
20. Puget, J.: Breaking all value symmetries in surjection problems. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 490–504. Springer, Heidelberg (2005)
21. Cohen, D., Jeavons, P., Jefferson, C., Petrie, K., Smith, B.: Symmetry definitions for constraint satisfaction problems. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 17–31. Springer, Heidelberg (2005)
22. Puget, J.: Dynamic lex constraints. In: Benhamou, F. (ed.) CP 2006. LNCS, vol. 4204, pp. 453–467. Springer, Heidelberg (2006)
23. Walsh, T.: General symmetry breaking constraints. In: Benhamou, F. (ed.) CP 2006. LNCS, vol. 4204, pp. 650–664. Springer, Heidelberg (2006)
24. Gent, I.P., Kelsey, T., Linton, S.A., McDonald, I., Migeul, I., Smith, B.: Conditional symmetry breaking. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 256–270. Springer, Heidelberg (2005)

25. Zampelli, S., Deville, Y., Dupont, P.: Symmetry breaking in subgraph pattern matching. In: SymCon'06, pp. 35–42 (2006)
26. Jegou, P.: Decomposition of domains based on the micro-structure of finite constraint satisfaction problems. In: Proceedings AAAI'93 (1993)
27. Benhamou, B., Saïdi, M.R.: Reasoning by dominance in not-equals binary constraint networks. In: Benhamou, F. (ed.) CP 2006. LNCS, vol. 4204, pp. 670–674. Springer, Heidelberg (2006)
28. Benhamou, B., Saïdi, M.R.: Some improvements in symmetry elimination in not-equals binary constraint networks. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 1–7. Springer, Heidelberg (2005)
29. Haralik, R.M., Elliot, G.L.: Increasing tree search efficiency for constraint satisfaction problems. Artificial Intelligence 14, 263–313 (1980)
30. McKay, B.: Practical graph isomorphism. Congr. Numer. 30, 45–87 (1981)
31. Puget, J.F.: Automatic detection of variable and value symmetries. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 474–488. Springer, Heidelberg (2005)
32. Mears, C., de la Banda, M.G., Wallace, M.: On implementing symmetry detection. In: Benhamou, F. (ed.) CP 2006. LNCS, vol. 4204, pp. 1–8. Springer, Heidelberg (2006)
33. Zampelli, S., Deville, Y., Saïdi, M.R., Benhamou, B., Dupont, P.: Breaking local symmetries in subgraph pattern matching. In: The International Symmetry Conference (ISC 2007), Edinburgh, SCOTLAND (2007)