

Symmetry in not-equals binary constraint networks

Belaïd Benhamou

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

Abstract

Symmetrical values of a CSP variable are in a sense redundant. Their detection and removal will simplify the problem search space. Many research works on symmetry in CSP's appeared recently. Most of them use the global symmetries of the studied problem to prune isomorphic search sub-spaces and less interest is given to local symmetry detection and exploitation. Local symmetry is very important for pruning the search space, but its detection is in general a hard task. In this paper we study local symmetry in Not-equals constraint networks. We show how this symmetry is detected efficiently and how it is used to increase Not-equals CSP resolution efficiency. Experiments show that our proposed approach is a considerable improvement for solving hard graph coloring and pigeon-hole problems. Some Dimacs graph coloring benchmarks had been solved efficiently.

1 Introduction

As far as we know the principle of symmetry is first introduced by Krishnamurty[17] to improve resolution in propositional logic. Symmetry for boolean constraints is studied in [2]. They showed that their exploitation is a real improvement for several automated deduction algorithms' efficiency. The notion of interchangeability in CSP's is introduced in [8] and symmetry in CSPs is studied in [22; 1]. Since that, many research works on symmetry appeared. For instance, the static approach used by James Crawford et al. in [4] for propositional logic theories consists in adding constraints expressing global symmetry of the problem. The same technique is used by Masayuki Fujita et al. in [18] to search for finite models of quasi-group problems.

Since a great number of constraints could be added, some researchers proposed to add the constraints during the search. In [13], authors add some conditional constraints which remove the symmetric of the partial interpretation in case of backtrack (this technique is called SBDS). In [7; 6; 23], authors proposed to use each subtree as a no-good to avoid exploration of some symmetric interpretations (this technique is called SBDD) and more recently the GE-trees conceptual for symmetry elimination is introduced in [24]. Other works on

symmetry can be found in the working notes of SymCon01, SymCon02 and SymCon03 CP workshops.

By removing statically all symmetric interpretations, Gilles Dequen and Olivier Dubois proved the non existence of the quasi-group QG2 of order 10 [5]. Finally in the same spirit, Pedro Meseguer and Carme Torras proposed a heuristic to remove symmetry as soon as possible in the search tree [20].

In this paper we deal with symmetry in Not-equals constraint networks. Not-equals constraint networks is a quite expressive framework. In theory, there is no loss of generality if we restrict our study to this framework, since each CSP can be reduced to a Not-equals constraint network. That is, the graph coloring problem is a particular Not-equals CSP and is shown in [11] to be NP-complete, then a polynomial reduction of each CSP to a not-equals CSP exists. A symmetry search method in general CSPs was given in [1], but its complexity is exponential in the worst case. Here we will show how the symmetry conditions are simplified in Not-equals constraint networks. We give a new sufficient condition of symmetry which leads to a linear complexity symmetry search algorithm with respect to the CSP size. We show how local symmetrical domain values are detected efficiently and how their removal simplifies the search space of a Not-equals constraint backtracking algorithm. This paper is organized as following :

Some CSP background is given in section 2. Section 3 discusses the notion of symmetry and shows how symmetrical values are detected efficiently in Not-equals constraint networks. We show in section 4 how a simplified forward checking method for Not-equals constraints takes advantage of symmetrical values to reduce the search space. In section 5 we evaluate the proposed techniques by experimental results. Section 6 compares our approche to other previous works and section 8 gives a conclusion.

In this work, we consider only binary CSP's, the CSP's involving only constraints between pairs of variables.

2 Background

A CSP involves a finite set $V = \{v_1, v_2, \dots, v_n\}$ of variables, a finite set $D = \{D_1, D_2, \dots, D_n\}$ of discrete domain values in which D_i is the domain associated with the variable v_i ; d_i denotes the fact that the value d belongs to the domain D_i , a finite set $C = \{c_1, c_2, \dots, c_m\}$ of constraints, and a finite set $R = \{R_1, R_2, \dots, R_m\}$ of relations corresponding to the con-

straints in C , R_i represents the list of tuples form in which the tuples of values satisfying the constraint c_i are enumerated. Thus, a CSP can be seen as a mathematical statement $\mathcal{P}(V, D, C, R)$ as defined in [21] and [19].

A Not-equals constraint between two CSP variables v_1 and v_2 (notation $v_1 \neq v_2$) is a constraint which forces them to be instantiated to different values. A Not-equals constraint network (notation NECSP) is a CSP whose constraints are Not-equals constraints. We will see in the sequel that both the graph coloring and the pigeon-hole problems can be represented in the framework of Not-equals constraint networks.

A value assignment is a mapping which specifies a value for each variable. It satisfies a constraint if it gives a combination of values to variables that is permitted by the constraint; otherwise it falsifies it. It satisfies a Not-equals constraint if it gives different values to the involved variables. A constraint satisfaction problem is the task of finding one or all value assignments for the constraint network such that all the constraints are satisfied.

3 Symmetry

We consider here, symmetry between values of a same domain.

3.1 Symmetry in CSPs

We recall in this section some symmetry notions first introduced in [1], and which we shall use to prove symmetry results in NECSPs. The main contribution in this work is not the symmetry definition itself, but the simplification of the symmetry conditions, the simplification of symmetry search and exploitation in NECSPs which we shall introduce in the sequel.

Definition 3.1 A permutation σ of domain values in a binary CSP $\mathcal{P} = (V, D, C, R)$ is defined as: $\sigma : \cup_{i \in [1, n]} D_i \rightarrow \cup_{i \in [1, n]} D_i$, such that $\forall i \in [1, n]$ and $\forall d_i \in D_i$, $\sigma(d_i) \in D_i$.

The permutation σ has no effect on the sets $\{V, D, C\}$ of the CSP \mathcal{P} . However, it induces a permutation σ_t on the tuples in each relation $R_{ij} \in R$ and then a permutation σ_R ¹ on the relations themselves. A symmetry of a CSP $\mathcal{P} = (V, D, C, R)$ is a permutation of domain values that leaves the CSP \mathcal{P} invariant.

Definition 3.2 A domain value permutation σ is a symmetry of a binary CSP $\mathcal{P} = (V, D, C, R)$ iff $\forall R_{ij} \in R$, $\langle d_i, d_j \rangle \in \text{tuples}(R_{ij}) \Rightarrow \langle \sigma(d_i), \sigma(d_j) \rangle \in \text{tuples}(R_{ij})$.

Remark 3.1 A symmetry of a CSP is a domain value permutation σ such that $\forall R_{ij} \in R$, $\sigma_R(R_{ij}) = R_{ij}$.

Example 3.1 (Pigeon-hole problem) The problem consists in putting n pigeons in $n-1$ holes such that each hole holds at most one pigeon. Take for instance 4 pigeons and 3 holes. The pigeons are represented by the set of variables, the holes by the domain values, as it is shown in the constraint graph of figure 1, the constraint c_{13} is given in its micro-structure

¹Both σ_t resp. σ_R are natural generalizations of σ to tuples resp. relations.

form showing the permitted tuples in the relation R_{13} . All the constraints c_{ij} are Not-equals constraints, they form a Not-equals constraint network.

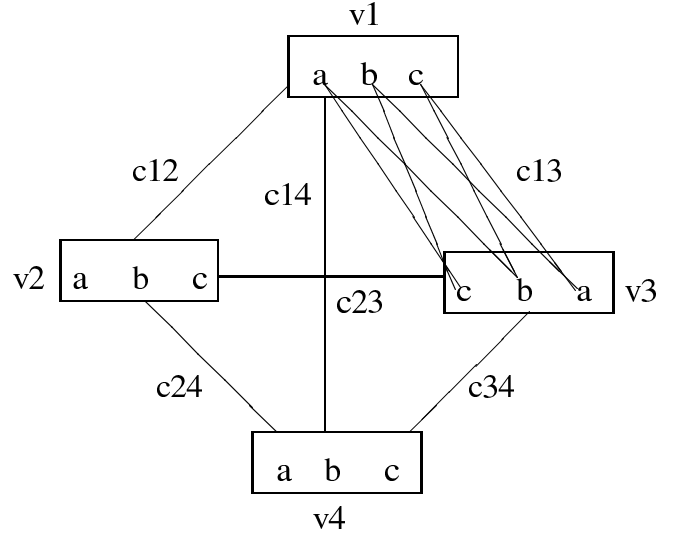


Figure 1: Pigeon-hole problem for 4 pigeons and 3 holes.

The permutation σ defined as: $\sigma(a_i) = (b_i)$, $\sigma(b_i) = (c_i)$, $\sigma(c_i) = (a_i)$, $\forall i \in [1, 4]$ keeps the pigeon-hole corresponding CSP of figure 1 invariant. Thus, it is a symmetry of the CSP.

Definition 3.3 Two domain values b_i and c_i for a CSP variable $v_i \in V$ are symmetrical (notation $b_i \sim c_i$) if there exists a symmetry σ of the CSP \mathcal{P} such that $\sigma(b_i) = c_i$ or $\sigma(c_i) = b_i$

Freuder in [9] defined the notion of Neighborhood Interchangeability (NI) as a sufficient condition to Full Interchangeability. As far as symmetry is concerned, two domain values b_i and c_i of the CSP variable v_i are Neighborhood Interchangeable iff there exists a symmetry σ of the CSP, such that $\sigma(b_i) = c_i$, $\sigma(c_i) = b_i$ and σ is the identity mapping for the other values.

Neighborhood Interchangeability is a particular symmetry which permutes two domain values and which is the identity elsewhere. Choueiri and Noubir in [3] studied the notion of Neighborhood Partial Interchangeability (NPI). Unfortunately, using only such symmetries is not sufficient in practice to solve hard symmetrical problems. The pigeonhole problem is a good example to demonstrate that point, it contains a lot of symmetries but no local interchange-abilities. In example 1, the domain values a_1, b_1 and c_1 of the variable v_1 are symmetrical but not Neighborhood Interchangeable. The notion of symmetry is more general than the Neighborhood Interchangeability, but symmetry detection is a harder problem. Using more general symmetry may be more useful in practice if this symmetry is detected and exploited efficiently.

Now we will show how symmetry is involved in CSP consistency. Let I be a value assignment of the CSP \mathcal{P} , σ a symmetry of the CSP \mathcal{P} and I/σ the value assignment obtained

by substituting in I every domain value d_i of the CSP variable v_i by $\sigma(d_i)$, formally: $I/\sigma[v_i] = \sigma(I[v_i])$, $\forall i \in [1, n]$. Now, we give a property that can be used to compute new solutions from known ones using symmetry:

Proposition 3.1 *I is a solution of \mathcal{P} if and only if I/σ is a solution of \mathcal{P} .*

Proof. Suppose that I is a solution of \mathcal{P} and R_{ij} is the relation corresponding to a constraint $c_{ij} \in C$. I is a solution of the CSP, thus I satisfies c_{ij} . In other words, there exists a tuple $t_1 \in R_{ij}$ such that $t_1 \subset I$. As $\sigma_R(R_{ij}) = R_{ij}$ (σ is a symmetry), then $\sigma_t(t_1) = t_2 \in R_{ij}$. Thus, $t_2 \in I/\sigma$ (by definition of I/σ and the fact that $t_1 \in I$). Therefore, I/σ satisfies c_{ij} , and I/σ is also a solution for the CSP \mathcal{P} . A similar proof can be given for the converse case by considering the converse symmetry σ^{-1} (QED).

Now we give the main property which relates symmetry and CSP consistency.

Theorem 3.1 *If b_i and c_i are two symmetrical values of a CSP variable $v_i \in V$ then b_i participates in a solution of the CSP if and only if the value c_i participates in a solution of the CSP.*

Proof. The values b_i and c_i are symmetrical by hypothesis, then there exists a symmetry σ of the CSP \mathcal{P} such that $\sigma(b_i) = c_i$. Suppose that b_i appears in a solution I of the CSP \mathcal{P} , then I/σ is a solution in which c_i appears (proposition 3.1). In other words the solution I is mapped into I/σ using the mapping σ .

We prove the converse by considering the converse symmetry σ^{-1} , (QED).

The previous theorem states that symmetrical values with a domain value $d_i \in D_i$ can be removed without affecting the CSP consistency when d_i is already shown not participating in any solution of the CSP.

3.2 Symmetry in NECSPs

All the symmetry notions defined in the previous section (section 3.1) for general CSPs are available for NECSPs. NECSPs is a quite expressive framework, in theory there is no matter to restrict our study to NECSPs, since each CSP can be reduced to a NECSP. A symmetry detection algorithm in binary CSPs is given in [1], but its complexity is exponential (in the worst case) when the backtracking is not limited. Here we shall show how the conditions of symmetry are simplified in NECSPs and how symmetrical values are computed efficiently a simpler algorithm. Now we give a new sufficient condition of symmetry for NECSPs which leads to a linear time complexity symmetry search algorithm. The following property is the main key of this work, it is very simple, but very useful for symmetry detection.

Theorem 3.2 *Let a_i and b_i be two values of the domain D_i of a Not-equals constraint network \mathcal{P} . If a_i and b_i appear in the same domains of the not-instantiated variables, then they are symmetrical.*

Proof. Suppose that the values a_i and b_i appear in the same domains of the CSP \mathcal{P} . We shall show that under this condition these values are symmetrical. We have to prove that there

exists a symmetry σ of \mathcal{P} such that $\sigma(a_i) = b_i$ or $\sigma(b_i) = a_i$. To be a symmetry, the permutation σ has to leave the CSP \mathcal{P} invariant. That is, the condition $\forall R_{ij} \in R, \sigma_R(R_{ij}) = R_{ij}$ must be verified. Take the transposition $(a_i, b_i) \circ (b_i, a_i)$ of the values a_i and b_i in each domain $D_k \in D$ as the permutation σ . To prove the first inclusion $R_{ij} \subset \sigma_R(R_{ij})$, we suppose that $(a_i, b_j) \in R_{ij}$ and show that $(a_i, b_j) \in \sigma_R(R_{ij})$. It is equivalent to prove that $(\sigma(a_i), \sigma(b_j)) \in R_{ij}$, since σ is a bijective mapping, it is in fact a transposition ($\sigma = \sigma^{-1}$). As $(a_i, b_j) \in R_{ij}$, then $a_i \neq b_j$. This implies that $\sigma(a_i) \neq \sigma(b_j)$, since σ is a bijective mapping. As both a_i and b_i appear in the same domains we deduce that $(\sigma(a_i), \sigma(b_j)) \in R_{ij}$. The same proof can be done to show the second inclusion $\sigma(R_{ij}) \subset R_{ij}$.

Example 3.2 *The sets $\{a_i, b_i, c_i\}$, $i \in [1, 4]$ form four classes of symmetrical values of the CSP of figure 1.*

Symmetry search in NECSPs

Theorem 3.2 gives a very simple property for symmetry search. Detecting the class of symmetrical values of a domain value $d_i \in D_i$ is equivalent to search the values which appear exactly in the domains where d_i appears. The symmetry search procedure is sketched in figure 2.

procedure *Symmetry*($d_i \in D_i$, **var** $cl(d_i)$:class);

Input : a value $d_i \in D_i$

Output : the class $cl(d_i)$ of symmetrical value of d_i .

begin

$cl(d_i) := \{d_i\}$

for each $a_i \in D_i - \{d_i\}$ **do**

if for each domain D_k of a non-instantiated variable **we have**

$(d_i \in D_k \text{ and } a_i \in D_k)$

or $(d_i \notin D_k \text{ and } a_i \notin D_k)$

then $cl(d_i) := cl(d_i) \cup \{a_i\}$

end

Figure 2: The symmetry search algorithm in NECSPs

Let n be the number of variables of the NECSP, and d the size of the largest domain. It is easy to see that the algorithm can run at most d times the first loop and at most n times the second one. It then computes the class $cl(d_i)$ of symmetrical values with a complexity $\mathcal{O}(nd)$ in the worst case. The different classes of symmetrical values of the domain D_i can be computed with a complexity $\mathcal{O}(nd^2)$ in the worst case. This algorithm has a linear complexity *w.r.t* the NECSP size, we use it to eliminate local symmetry at each node of the search tree of a backtracking NECSP algorithm.

Below we show how a backtracking method for NECSPs can be augmented with the advantage of symmetry.

4 Symmetry advantage in NECSPs

Now we are in the position to show how these symmetrical values can be used to increase the efficiency of NECSP backtracking algorithms. Freuder and al in [9] uses interchangeable values in the pre-processing phase of CSP resolution, Haselbock in [15] gives a good use of interchangeability in

the known filtering algorithm REVISE and adapted the backtracking algorithm to compute families of interchangeable solutions. For efficiency reasons, we choose for our implementation a *Simplified Forward Checking* method to be the baseline method that we want to improve by adding the property of symmetry.

The forward checking principle [14] is based on filtering non-instantiated variable domains considering instantiated ones. When the current variable v_i is to be instantiated with a value d_i , consistency checking takes place in a forward direction, from the current variable to the future variables (non-instantiated variables having a constraint with the current variable v_i). The domains of future variables are filtered such that all values which are not consistent with respect to the current instantiation of v_i are removed. In case of NECSPs, the filtering is simplified and consists only in removing the value d_i from the domains of the future variables. This leads to a *Simplified Forward Checking (SFC)* method which we consider in our implementation. If as a result of this filtering we obtain an empty current domain for a future variable v_j , forward checking stops the filtering and its effects are undone, and a new instantiation is attempted for the variable v_i . If no consistent instantiation can be found for v_i , the method performs chronological backtracking to the variable v_{i-1} . Otherwise, v_i had been instantiated with no contradiction and then we choose the next variable to instantiate and repeat the same process.

Choosing optimally the next variable to be instantiated is a hard task. Several works have been done to contrive heuristics for variable ordering. In practice, the one minimizing the ratio:

$$r = \frac{|D_i|}{Degree(v_i)}$$

where v_i is a non-instantiated variable, $|D_i|$ its domain size and $Degree(v_i)$ the number of constraints of the initial CSP in which v_i is involved, is shown to be one of the most effective. We use it in the search to select the next variable to be instantiated. In the sequel, this heuristic is denoted by (DomDeg). It's a well known heuristic in CSPs [10], there is no need to give its code here.

Theorem 3.1 expresses an important property that we use to prune search spaces of backtracking methods. Indeed if d_i participates in no solution of the CSP \mathcal{P} then all values which are symmetrical to d_i do not. Thus, we prune the sub-space which corresponds to their assignments. Formally we get the following:

Corollary 4.1 *If $d_i \sim \hat{d}_i$ and d_i doesn't participate in any solution of \mathcal{P} , then \hat{d}_i doesn't participate in any solution.*

The previous corollary is a direct consequence of theorem 3.1. We can cut $k-1$ branches in the search tree if there are k symmetrical domain values and one of them has already been identified as not participating in any solution. If $SymClass(d)$ denotes the class of the domain values which are symmetrical to d , then in the enumeration process we consider only the value d , since the other values of $SymClass(d)$ are symmetrical with it.

```

Procedure SFC-sym( $D, I, k$ );  $\{I = [d_1, d_2, \dots, d_k]\}$ 
var empty:boolean;
begin
  if  $k = n$  then  $[d_1, d_2, \dots, d_k]$  is a solution, stop
  else
    begin
      empty:=false;
      for each  $v_i \in V$ , such that  $C_{ik} \in C, v_i \in \text{future}(v_k)$  do
        if not(empty) and  $d_k \in D_i$  then
          begin
             $D_i = D_i - \{d_k\}$ ;
            if  $D_i = \emptyset$  then
              begin
                undo filtering effects;
                symmetry( $d_k \in D_k, SymClass(d_k)$ );
                 $D_k = D_k - SymClass(d_k)$ ;
                empty=true;
              end
            end
          end
        if not(empty) then
          begin
             $v_{k+1} = \text{next-variable}(v_k)$ 
            repeat
              take  $d_{k+1} \in D_{k+1}$ 
               $D_{k+1} = D_{k+1} - d_{k+1}$ 
               $I = [d_1, d_2, \dots, d_k, d_{k+1}]$ ;
              SFC-sym( $D, I, k + 1$ );
            until  $D_{k+1} = \emptyset$ 
          end
        end
      end
    end
  end

```

Figure 3: The Simplified Forward Checking method augmented by symmetry

4.1 Combining trivial symmetry with the detected one

Some trivial symmetries can be exploited without effort of detection. All the values which form the intersection of the domains of a Not-equals constraint network are trivially symmetrical.

Proposition 4.1 *If \mathcal{P} is an NECSP having n domains, then all the values in $T = \bigcap_{i=1}^n D_i$ are symmetrical.*

Proof. The proof is trivial, since all the values in T appear in the same domains, thus satisfy the condition of theorem 3.2.

This trivial symmetry is very important, since during the search of solution (i.e instantiation) the non used values of the subset T remain symmetrical at each node of the search tree. We use only one of them and the other ones are removed by symmetry. This is very useful in both graph coloring and pigeon-hole where all the variables have a same domain. There is one domain $D_i = \{0, \dots, c-1\}$ of c values and $T = D_i$ in this case. If during the search all the first values $\{0, \dots, mdn\}$ of the ordered finite domain $D_i = \{0, \dots, c-1\}$ (with $0 \leq mdn \leq c-1$) are used in the partial instantiation I , then the values of the part $\{mdn+1, \dots, c-1\}$ remain symmetrical. All the values of the domain D_i are trivially symmetrical before starting instantiation of variables ($mdn = 0$).

Such trivial symmetries are very useful but they disappear as soon as the first variables are instantiated. The propagation process forces new values to be used, thus increases the mdn and decreases the subset $\{mdn + 1, \dots, c - 1\}$ of trivial symmetrical values. This subset becomes empty when mdn reaches the value $c - 1$. On the other hand, the subset $\{0, \dots, mdn\}$ of the used values increases and become quickly identical to the whole domain D_i .

A lot of other symmetries exist between the values of the part $\{0, \dots, mdn\}$ which we detect by using the symmetry procedure of figure 2. The trivial symmetry of the part $\{mdn + 1, \dots, c - 1\}$ and the one we detect on the part $\{0, \dots, mdn\}$ are independent, since they are defined on two disjoint parts of the domain. Their combination is then straight forward and if k is the number of detected symmetrical values then $c - mdn - 1 + k$ symmetry cuts can be made when both kind of symmetry are associated to prune the search space.

If D is the set of domains, I the partial instantiation, and k the index of the current variable v_k under instantiation, then figure 3 gives the sketch of the SFC procedure augmented by the (DomDeg) heuristic and the property of symmetry (notation SFC-sym). The structure $future(v_i)$ represents the set of non-instantiated variables remaining after the instantiation of v_i and $next-variable$ a function which encodes the (DomDeg) heuristic. In the sequel SFC will denote the SFC method augmented by the DomDeg heuristic.

5 Experiments

Now we shall investigate the performance improvement of our search technique by experimental analysis. We test two problems: the Graph coloring problem and the pigeon-hole problem. The graph coloring problem consists in coloring the vertices of a graph such that no two vertices which are joined by an edge have the same color. The pigeon-hole problem is defined in example 3.1. It is a particular graph coloring problem where the associated graph of constraints is a clique defined on the vertices representing the pigeons and where the colors are the holes (see figure 1). These are two problems which are trivially expressed as Not-equals constraint networks. They are known to be hard problems and are quite significant to show the symmetry behavior in NECSP resolution. First, we will test and compare both the simplified forward checking (SFC) and the simplified forward checking augmented by symmetry (SFC-sym) on the random graph coloring problems to show the symmetry gain. The (SFC-sym) method is applied to tackle Dimacs graph coloring benchmarks and the pigeon-hole problem. The complexity indicators are the number of nodes and time. The time needed for computing symmetry is added to the total CPU time given in seconds. The source codes are written in C and compiled on a Pentium 4, 2.5 G HZ with 256 MB. For efficiency reasons SFC and SFC-sym are implemented in an iterative way.

5.1 Random graph coloring problems

Random graph coloring problems are generated with respect to the following parameters: (1) n the number of vertices

(variables), (2) Cls the number of colors (domain values) and (3) d the density which is a number between 0 and 1 expressed by the ratio : number of constraints (number of edges in the constraint graph) to the number of all possible constraints. The samples of each test are 100 randomly generated instances and the measures are taken in average.

Pb	SFC			SFC-sym		
	Nodes	Time	Cons	Nodes	Time	Cons
8	2667	0.0	0.0	9	0.0	0.0
9	48939	0.03	0.0	16	0.0	0.0
10	2263385	15.55	3.0	66	0.0	3.0
11	11551093	66.87	73.0	158	0.0	73.0
12	643068	3.65	99.0	59	0.0	99.0
13	230	0.0	100.0	57	0.0	100.0

Table 1: Graph coloring instances with $n=50$ and $d=0.5$

SFC-sym			
Cls	Nodes	Time	Cons
13	69	0.0	0.0
14	481	0.0	0.0
15	7036	0.02	0.0
16	182447	3.02	0.0
17	1907019	26.92	44.0
18	289612	3.78	100.0
19	2407	0.0	100.0
20	240	0.0	100.0

Table 2: Graph coloring instances with $n=100$ and $d=0.5$

Table 1 gives the results of both SFC and SFC-sym methods on random graph coloring instances with $n=50$ and $d=0.5$. The table gives for each method the number of colors of the problem (Cls), the number of nodes, the time and the percentage of consistency (Cons). We can see that SFC-sym improves drastically SFC and symmetry is profitable to solve random graph coloring problems in the hard region.

Random graph coloring instances in the transition phase are known to be very hard problems. The method SFC can not solve in reasonable time random graph coloring instances in the hard region when the number of variables is greater than 50. However SFC-sym can solve large scale instances thanks to symmetry. Table 2 shows the results of SFC-sym on instances with $n=100$. We can see that the hardest problem is solved with only 26 seconds.

5.2 Dimacs graph coloring benchmarks

Table 3 shows the results of SFC-sym on some Dimacs graph coloring benchmarks. We find for each problem the minimal number of colors min that its corresponding graph needs to be colored. This is done by proving the consistency of the problem with min colors (denoted by yes on the column cons?) and by proving the inconsistency of the problem when using $min - 1$ colors (denoted by no on the column Cons?). SFC-sym solved all most of the instances of the classes: mul-sol, zeroin, fpsol and inithx, but we give just the results of

Problem	SFC-sym			
	Instance	Colors	Nodes	Time
mulsol.i.4.col	30	3454	0.11	no
mulsol.i.4.col	31	184	0.01	yes
mulsol.i.5.col	30	2597	0.08	no
mulsol.i.5.col	31	185	0.01	yes
zeroin.i.3.col	29	49	0.01	no
zeroin.i.3.col	30	205	0.01	yes
fpsol2.i.3.col	29	143213	3.68	no
fpsol2.i.3.col	30	450	0.3	yes
school1.col	13	37529	3.12	no
school1.col	14	111184	7.60	yes
school1-nsh.col	13	63	0.01	no
school1-nsh.col	14	734	0.02	yes
DSJ125.1.col	4	17	0.00	no
DSJ125.1.col	5	1197	0.01	yes
DSJR500.1.col	11	11	0.00	no
DSJR500.1.col	12	501	0.02	yes
1.Fullins-3.col	3	23	0.00	no
1.Fullins-3.col	4	29	0.00	yes
1.Fullins-4.col	4	10043	0.26	no
1.Fullins-4.col	5	92	0.00	yes
2.Fullins-3.col	4	39545	0.30	no
2.Fullins-3.col	5	51	0.00	yes

Table 3: Dimacs graph coloring benchmarks

some of them. The instances of the classes: school, DSJ, DSJR and Fullins shown in table 3 seem to be more important (see the Dimacs web: <http://mat.gsia.cmu.edu/COLOR02>). As we can see in table 3, SFC-sym solved them efficiently.

5.3 Instances of pigeon-hole problem

The pigeon-hole generator needs only the number of pigeon as input.

Problem	SFC-sym	
	Pigeons	Nodes
500	499	0.0
1000	999	1.0
1500	1499	4.0
2000	1999	11.0
2500	2499	20.0
3000	2999	34.0
3500	3499	59.0

Table 4: Pigeon-hole problem

The SFC method can not solve the pigeon-hole problem when the number of pigeon is greater than 15 (many hours time CPU). However, resolution complexity of this problem is linear for SFC-sym. We can see in table 4 that the number of the nodes of the search tree is equal to the number of pigeons minus one for each problem. That is, because all the values (holes) at each node of the search tree are trivially symmetrical, thus only one of them is tested. Time variation is weakly quadratic w.r.t the number of pigeon, since all the values are trivially symmetrical.

6 Some related works

- In [16] authors studied three classes of CSPs for which symmetry is tractable. The value-Interchangeable CSPs (ICSPs) class is in relation with our work. That is, when all the variable domains of a NECSP are the same (as in the graph coloring problem), it becomes an ICSP. For this particular case, the value symmetry elimination technique used in [16] for the ICSP class seems to be equivalent to the trivial symmetry elimination which we described in section 4.1. But, in general NECSPs are not ICSPs and the symmetries detected by the procedure of figure 2 are not considered in the ICSP symmetry breaking.
- In other hand Gent introduced in [12] a symmetry constraint to eliminate what he calls indistinguishable values. His approach works by addition of symmetry constraints rather than dynamic detection of symmetry. This technique may be used to break some of the global and the trivial symmetry described in section 4.1 such as the one of the graph coloring problem for example. However, it does not deal with the local symmetries which we detect by using the symmetry procedure of figure 2.

7 Acknowledgement

Many Thanks to the reviewers for their interesting comments.

8 Conclusion

We introduced a simple sufficient condition of symmetry in NECSPs. We used this condition to provide a linear time complexity symmetry detection method. Symmetrical values of a given domain of a NECSP are computed efficiently and a simplified forward checking algorithm for NECSPs is adapted to exploit this information to prune isomorphic search sub-spaces. We have experimented the resulting method on some important problems and showed the advantage of reasoning by symmetry in NECSPs. Further investigation will consist in extending the symmetry notion to values of different variables. Another investigation will consist in trying to apply this work to some clique problems which are related to both the CSP satisfaction and graph coloring problems.

References

- [1] B. Benhamou. Study of symmetry in constraint satisfaction problems. *In the working notes of the workshop PPCP'94*, 1994.
- [2] B. Benhamou and L. Sais. Theoretical study of symmetries in propositional calculus and application. *Eleventh International Conference on Automated Deduction, Saratoga Springs, NY, USA*, 1992.
- [3] Berthe Y. Choueiry and Guevara Noubir. On the computation of local interchangeability in discrete constraint satisfaction problems. *In Proc of AAAI/IAAI'98*, pages 326–333, 1998.

- [4] James Crawford, Matthew L. Ginsberg, Eugene Luck, and Amitabha Roy. Symmetry-breaking predicates for search problems. In *KR'96: Principles of Knowledge Representation and Reasoning*, pages 148–159. Morgan Kaufmann, San Francisco, California, 1996.
- [5] Olivier Dubois and Gilles Dequen. The non-existence of (3,1,2)-conjugate orthogonal idempotent Latin square of order 10. In *7th International Conference on Principles and Practice of Constraint Programming*, volume 2239 of *LNCS*, pages 108–121. Springer Verlag, 2001.
- [6] T. Fahle, S. Schamberger, and M. Sellmann. Symmetry breaking. In *International conference on constraint programming*, volume 2239 of *LNCS*, pages 93–108. Springer Verlag, 2001.
- [7] F. Focacci and M. Milano. Global cut framework for removing symmetries. In *International conference on constraint programming*, volume 2239 of *LNCS*, pages 77–82. Springer Verlag, 2001.
- [8] E.C. Freuder. Eliminating interchangeable values in constraints satisfaction problems. *Proc AAAI-91*, pages 227–233, 1991.
- [9] E.C. Freuder and W. Benson. Interchangeability preprocessing can improve forward checking search. In *proc. ECAI*, 1992.
- [10] D. Frost and R. Dechter. Look-ahead value ordering for constraint satisfaction search. In *Proceedings of IJCAI*, pages 301–306, 1995.
- [11] M.R. Garey and D.S. Johnson. Computers and intractability: A guide to the theory of np-completeness, w.h. freeman. Technical report, 1979.
- [12] I. Gent. A symmetry breaking constraint for indistinguishable values. In *SymCon*, 2001.
- [13] I.P. Gent, W. Harvey, and T. Kelsey. Groups and constraints: Symmetry breaking during search. In *International conference on constraint programming*, volume 2470 of *LNCS*, pages 415–430. Springer Verlag, 2002.
- [14] R. M. Haralik and G. L. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence 14*, pages 263–313, 1980.
- [15] A. Haselbock. Exploiting interchangeability in constraint satisfaction problems. In *Proceedings of IJCAI*, pages 282–287, 1993.
- [16] P. Van Hentenryck, P. Flener, J. Pearson, and M. Argen. Tractable symmetry breaking for csps with interchangeable values. In *IJCAI*, pages 277–282, 2003.
- [17] B. Krishnamurty. Short proofs for tricky formulas. *Acta informatica*, (22):253–275, 1985.
- [18] J. Slaney M. Fujita and F. Bennett. Automatic generation of some results in finite algebra. In *proceedings of the 13th International Joint Conference on Artificial Intelligence, Chambéry, France*, pages 52–57, 1993.
- [19] A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence 8*, pages 99–118, 1977.
- [20] Pedro Meseguer and Carme Torras. Solving strategies for highly symmetric csps. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 400–405. Morgan Kaufmann, 1999.
- [21] U. Montanari. Networks of constraints : Fundamental properties and applications to picture processing. *Information Science 7*, pages 95–132, 1974.
- [22] J.F. Puget. On the satisfiability of symmetrical constrained satisfaction problems. In *ISMIS*, 1993.
- [23] J.F. Puget. Symmetry breaking revisited. In *International conference on constraint programming*, volume 2470 of *LNCS*, pages 446–461. Springer Verlag, 2001.
- [24] Colva M. Rouney-Dougal, Ian P. Gent, Tom Kesley, and steve A. Linton. Tractable symmetry breaking using restricted search trees. In *proceedings of ECAI-04 (to appear)*, 1994.