

Reasoning by dominance in Not-Equals binary constraint networks

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
email: Belaïd.Benhamou@cmi.univ-mrs.fr, saidi@cmi.univ-mrs.fr

Abstract

Dynamic detection and elimination of symmetry in constraints, is in general a hard task, but in *Not-Equals binary constraint networks*, the symmetry conditions can be simplified. In this paper, we extend the principle of symmetry to dominance in *Not-Equals Constraint Networks* and show how dominated values are detected and eliminated at each node of the search tree. A Linear time complexity algorithm which detects the dominated values is proposed. Dominance is exploited in an enumerative method adapted to solve Not-Equals CSPs. This enumerative method augmented by Dominance is experimented on both randomly generated instances of graph coloring and Dimacs graph coloring benchmarks and its performance is compared to the same method augmented by symmetry and to the well known DSATUR method. The obtained results show that reasoning by dominance improves symmetry reasoning and our method outperforms both previous methods in solving graph coloring.

1 Introduction

As far as we know the principle of symmetry is first introduced by [Krishnamurty, 1985] to improve resolution in propositional logic. Symmetry for boolean constraints is studied in [Benhamou and Sais, 1992] where the authors showed that its exploitation is a real improvement for several automated deduction algorithms' efficiency. The notion of interchangeability in CSP's is introduced in [Freuder, 1991] and symmetry in CSPs is studied in [Puget, 1993; Benhamou, 1994a]. Since that, many research works on symmetry appeared. For instance, the static approach used by James Crawford et al. in [James Crawford et al., 1996] for propositional logic theories consists in adding constraints expressing global symmetry of the problem. This technique has been improved in [F.A. Aloul et al., 2003] and extended to 0-1 Integer logic Programming in [F.A. Aloul et al., 2004].

Since a great number of constraints could be added in the static approach, some researchers proposed to add the constraints during the search. In [I.P. Gent et al., 2002], authors add some conditional constraints which remove the

symmetric of the partial interpretation in case of backtrack (this technique is called SBDS). In [Fahle et al., 2001; F. Focacci and M. Milano, 2001; Jean F. Puget, 2002] authors proposed to use each subtree as a no-good to avoid exploration of some symmetric interpretations (this technique is called SBDD) and the GE-trees conceptual for symmetry elimination is introduced in [Colva M. Roney-Dougal et al., 2004]. More recently a method which breaks symmetries between the variables of an AllDiff constraint is studied in [Puget, 2005b], a method which eliminates all value symmetries in surjection problems is given in [Puget, 2005a], and a work gathering the different symmetry definitions is done in [Cohen et al., 2005].

We investigate in this article the principle of dominance in *Not-Equals binary Constraint Networks* (notation *NECSPs*). Dominance is a weak symmetry principle which extend the Full substitutability notion [Freuder, 1991]. Dominance is first introduced in [Benhamou, 1994b] for general CSPs, but it is shown that its detection is harder than symmetry. Here, we show how dominance is adapted, detected, and exploited efficiently in NECSPs. Of course, the NECSPs is a limited framework, but in theory, this restriction remains NP-complete. Indeed, Graph coloring fits in the NECSPs framework and is NP-complete [M.R. Garey and D.S. Johnson, 1979], thus solving Not-Equals CSPs is in general a NP-complete problem. Besides, in practice, this framework is quite expressive, it covers a broad range of problems in artificial intelligence, such as Time-tabling and Scheduling, Register Allocation in compilation, and Cartography [A. Ramani et al., 2004].

Detecting symmetrical domain values of a CSP variable during search is in general a hard task. A symmetry detection method is proposed in [Benhamou, 1994a], but its complexity is exponential in the worst case. In case of Not-Equals CSPs, some symmetrical values are detected with a linear time complexity [Benhamou, 2004].

We show in this article how symmetry is extended to dominance in NECSPs, and how the symmetry condition given in [Benhamou, 2004] is weakened in the case of failing to instantiate a variable with a value of its domain during the search. We give a weak symmetry/dominance condition which leads to a dominance detection algorithm whose efficiency is better and which detects both the dominance and

more symmetries than the algorithm defined in [Benhamou, 2004].

The rest of this article is organized as follows: Section 2 gives a brief background on CSPs. In section 3 we discuss the dominance notion and show how the symmetry condition given in [Benhamou, 2004] is weakened and extended to dominance. We give in the subsection 3.4 an efficient dominance detection algorithm in Not-Equal CSPs. We show in section 4 how Dominance is exploited in a simplified forward checking (SFC) method adapted to Not-Equals CSPs. In section 5 we evaluate and compare the effectiveness of our result by carrying experiments on both Dimacs graph coloring benchmarks and randomly graph coloring generated instances. Section 6 discusses some related works and Section 7 concludes.

2 The CSP formalism

A CSP is a quadruple $P = (X, D, C, R)$ where: $X = \{X_1, \dots, X_n\}$ is a set of n variables; $D = \{D_1, \dots, D_n\}$ is the set of finite discrete domains associated to the CSP variables, D_i includes the set of possible values of the CSP variable X_i ; $C = \{C_1, \dots, C_m\}$ is a set of m constraints each involving some subsets of the CSP variables. A binary constraint is a constraint which involves two variables; $R = \{R_1, \dots, R_m\}$ is a set of relations corresponding to the constrains of C . R_i represents the list of value tuples permitted by the constraint C_i . A CSP P can be represented by a constraint graph $G(X, E)$ where the set of vertices X is the set of the CSP variables and each edge of E connects two variables involved in the same constraint $C_i \in C$.

A binary constraint is called a Not-Equal constraint if it forces the two variables X_i and X_j to take different values (it is denoted by $X_i \neq X_j$). A Not-Equal CSP (NECSP) is a CSP whose all constraints are Not-Equal constraints.

An instantiation $I = (a_1, a_2, \dots, a_n)$ is the variable assignment $\{X_1 = a_1, X_2 = a_2, \dots, X_n = a_n\}$ where each variable X_i is assigned to a value a_i of its domain D_i . A constraint $C_i \in C$ is satisfied by I if the projection of I on the variables involved in C_i is a tuple of R_i . The instantiation I is consistent if it satisfies all the constraints of C , thus I is a solution of the CSP. An instantiation of a subset of the CSP variables is called a partial instantiation. An instantiation is total if it is defined on all the CSP variables. Given a CSP, the main question is to decide its consistency.

Example 2.1 Take the binary NECSP whose constraint graph is shown in the figure 1. The CSP variables are the vertices X_1, \dots, X_5 and the domains are include in boxes. Each edge of the constraint graph connecting two vertices X_i and X_j , expresses a Not-Equal constraint between the corresponding CSP variable X_i and X_j .

3 Dominance in NECSPs

Symmetrical values of a CSP variable are values which have the same semantical relevance to participate in the solutions of the CSP. But, the values of domains in a CSP are not all equally semantical relevant. Some of them can be more likely to participate in solutions than other ones. The values in the

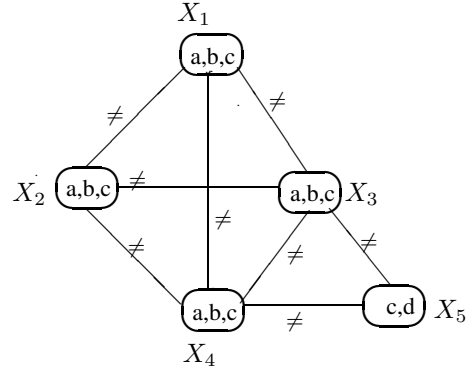


Figure 1: Constraint graph of a NECSP

first group *dominate* the latter ones. It is important to detect the dominant values in order to consider them prior in assignments. This will reduce the search space. Freuder in [Freuder, 1991] introduced the notion of Substitutability as a weak Interchangeability. In the same spirit, Benhamou in [Benhamou, 1994b] defined a weak symmetry called *Dominance* which extends the Full substitutability notion. Here we adapt the principle of dominance to NECSPs and give some sufficient conditions which leads to a linear algorithm for dominance detection.

3.1 The principle of Dominance

Definition 3.1 [Semantic Dominance] A value a_i dominates another value b_i for a CSP variable $v_i \in V$ (notation $a_i \succeq b_i$) iff [There exist a solution of the CSP which assigns the value b_i to the variable $v_i \Rightarrow$ there exist a solution of the CSP which assigns the value a_i to v_i].

The value b_i participates in a solution if the value a_i does; otherwise it does not. The value b_i can be removed from D_i without affecting the CSP consistency.

Proposition 3.1 If $a_i \succeq b_i$ and a_i doesn't participate in any solution of \mathcal{P} , then b_i doesn't participate in any solution of \mathcal{P} .

Proof 1 The proof is a direct consequence of Definition 3.1

Proposition 3.1 allows to remove all the values which are dominated by the value $a_i \in D_i$ without affecting the CSP consistency. Thus, Dominance complements the usual CSP inconsistency methods, which attempt to remove values that doesn't participate in any solution. Dominance can be generalized to values of different variables, but here we restrict the study to values of a same domain.

Remark 3.1 • Symmetrical domain values are domain values which mutually dominate each other. That is, two values $a_i \in D_i$ and $b_i \in D_i$ are symmetrical iff $a_i \succeq b_i$ and $b_i \succeq a_i$. Thus, a_i participates in a solution of the CSP iff b_i does.

- Dominance extends the Full Substitutability notion given in [Freuder, 1991].
- Here, Dominance means "more likely to participate in solutions" as it is defined in [Benhamou, 1994b], it

should not be confused with the Symmetry By Dominance Detection notion (SBDD) [Fahle et al., 2001].

3.2 A sufficient condition for dominance

A symmetry detection algorithm in general discrete finite CSPs is proposed in [Benhamou, 1994a]. Its complexity is exponential in the worst case. It is shown in [Benhamou, 2004] how the symmetry conditions can be simplified in NECSPs and how the symmetrical values can be detected efficiently with a simpler algorithm having a linear time complexity *w.r.t* to the NECSP size. This result is based on the following property:

Theorem 3.1 *Let a_i and b_i be two values of the domain D_i of a Not-Equals CSP \mathcal{P} . If a_i and b_i appear in the same domains of the un-instantiated variables, then they are symmetrical.*

Proof 2 *See [Benhamou, 2004].*

It is a very simple property, but very useful for detecting and eliminating symmetrical values of the same domain. By using this property, we can deduce that the values a and b of the domain of the CSP variable X_1 illustrated in Figure 1 are symmetrical. Indeed, they both appear in the domains of X_2 , X_3 , X_4 and do not appear in the domain of X_5 . By a similar reasoning, we can also deduce that the values a and b of the domains of the variables X_2 , X_3 and X_4 are symmetrical.

We give in the following the sufficient conditions for dominance which represent the main key of this work.

Theorem 3.2 *Let a_i and b_i be two values of a domain D_i corresponding to a variable X_i of a Not-Equals CSP \mathcal{P} , I a partial instantiation of \mathcal{P} , and Y the set of un-instantiated variables of \mathcal{P} . If the two following conditions:*

1. $a_i \in D_j \Rightarrow b_i \in D_j$, for all $X_j \in Y$ such that X_j shares a constraint with X_i .
2. $a_i \in D_j \Leftrightarrow b_i \in D_j$ for each variable X_j of Y which does not share a constraint with X_i

hold, then a_i dominates b_i in \mathcal{P} .

Proof 3 *We have to prove under the conditions (1) and (2) that if b_i participates in a solution, then a_i participates in a solution too. Let $I_{X_i=b_i}$ be a solution of the CSP \mathcal{P} where the variable X_i is instantiated to b_i , we have to show the existence of a solution where the variable X_i is instantiated to a_i . Let \mathcal{P}' be the CSP obtained from \mathcal{P} by removing the value b_i from the domains of the variables having a constraint with X_i where the value a_i does not appear. This operation does not render the domains empty, since $I_{X_i=b_i}$ is a solution of the CSP \mathcal{P} . That is, each reduced domain must contain at least two values before removing the value b_i . The CSP \mathcal{P}' is a sub-CSP of \mathcal{P} , it has the same set of variables, the same set of constraints, but the reduced domains become sub-sets of the original domains from which they derive. The CSP \mathcal{P}' is more constrained than \mathcal{P} . That is, each solution of \mathcal{P}' is a solution of \mathcal{P} . According to the conditions (1) and (2) which hold on \mathcal{P} , the values a_i and b_i become symmetrical in the CSP \mathcal{P}' . The conditions of theorem 3.1 hold, then the value a_i participates in a solution of \mathcal{P}' iff the value b_i does. On the other hand, the CSP $\mathcal{P}_{X_i=b_i}$ resulting from \mathcal{P} by considering*

the assignment $X_i = b_i$ is identical to the CSP $\mathcal{P}'_{X_i=b_i}$ resulting from \mathcal{P}' by considering the assignment $X_i = b_i$. We deduce that $I_{X_i=b_i}$ is a solution of \mathcal{P}' , and then there exists a solution $I'_{X_i=a_i}$ of \mathcal{P}' where X_i is instantiated to a_i , since a_i and b_i are symmetrical in \mathcal{P}' . As the CSP \mathcal{P}' is more constrained than the CSP \mathcal{P} , then $I'_{X_i=a_i}$ is a solution of \mathcal{P} . We proved the existence of a solution where a_i is assigned to X_i , thus a_i dominates b_i in \mathcal{P} .

Example 3.1 *Consider the domain of the variable X_3 of Figure 1, the value a dominates the value c .*

3.3 The weakened dominance sufficient conditions

Before introducing the weakened sufficient conditions, we define the notion of assignment trees and failure trees corresponding to the enumerative search method used to prove the consistency of the considered CSP.

Definition 3.2 *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 its consistency proof, where the nodes represent the variables of the CSP and where the edges out coming from a node X_i are labeled by the different values used to instantiate the corresponding CSP variable X_i .*

The root of the tree is the first variable in the ordering. In this work, the considered backtracking method is Forward Checking [R. M. Haralik and G. L. Elliot, 1980].

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

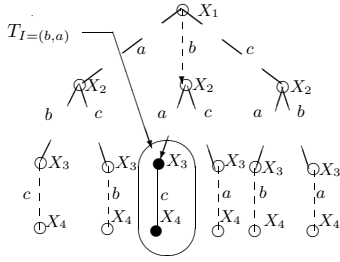
We associate to each inconsistent partial instantiation, corresponding to a given path in the assignment tree, a failure tree defined as follows:

Definition 3.3 *Let T be an assignment tree corresponding to a consistency proof of a CSP \mathcal{P} , $I = (a_1, a_2, \dots, a_i)$ an inconsistent partial instantiation of the variables X_1, X_2, \dots, X_i corresponding to the path $\{X_1, X_2, \dots, X_i\}$ in T . We call a failure tree of the instantiation I , the sub-tree of T noted by $T_{I=(a_1, a_2, \dots, a_i)}$ such that:*

1. *The root of the tree T and the root of the sub-tree $T_{I=(a_1, a_2, \dots, a_i)}$ are joined by the path corresponding to the instantiation I ;*
2. *All the CSP variables corresponding to the leaf nodes of $T_{I=(a_1, a_2, \dots, a_i)}$ have empty domains.*

Example 3.2 *Take the CSP of Figure 1 and apply a forward checking process on it *w.r.t* the variable ordering $\{X_1, X_2, X_3, X_4, X_5\}$. Figure 2 illustrates the assignment tree of the considered CSP. If we take the partial instantiation $I = (b, a)$ which assigns X_1 to the value b and X_2 to the value a , then the failure tree $T_{I=(b, a)}$ of the instantiation I is shown in the figure 2 (the part in a box).*

We can now give the weakened sufficient conditions of dominance. The main idea is to weaken the dominance conditions of theorem 3.2 when an inconsistent partial instantiation


 Figure 2: An assignment tree and the failure tree of $I=(b,a)$

is generated during the search. That is, the conditions of theorem 3.2 are restricted to only the variables involved in the failure tree among the un-instantiated ones.

Theorem 3.3 Let $\mathcal{P}(X, C, D, R)$ be a CSP, $a_i \in D_i$ and $b_i \in D_i$ two values of the domain D_i of the current CSP variable X_i under instantiation, $I_0 = (a_1, \dots, a_{i-1})$ a partial instantiation of the $i-1$ variables instantiated before X_i such that the extension $I = I_0 \cup \{a_i\} = (a_1, \dots, a_{i-1}, a_i)$ is inconsistent, $T_{I=(a_1, \dots, a_{i-1}, a_i)}$ is the failure tree of I and $Var(T_{I=(a_1, \dots, a_{i-1}, a_i)})$ the set of variables corresponding to the nodes of $T_{I=(a_1, \dots, a_{i-1}, a_i)}$. If the two following conditions:

1. $b_i \in D_j \Rightarrow a_i \in D_j$, for all $X_j \in Var(T_{I=(a_1, \dots, a_{i-1}, a_i)})$ such that X_j shares a constraint with X_i .
2. $a_i \in D_j \Leftrightarrow b_i \in D_j$ for each other variable X_j of $Var(T_{I=(a_1, \dots, a_{i-1}, a_i)})$ which do not share a constraint with X_i

hold, then the extension $J = I_0 \cup \{b_i\} = (a_1, \dots, a_{i-1}, b_i)$ is inconsistent.

Proof 4 Let $\mathcal{P}'(V', C', D', R')$ be a sub-CSP of the CSP $\mathcal{P}(V, C, D, R)$ such that $V' = Var(T_{I=(a_1, \dots, a_{i-1}, a_i)}) \cup X_i$ and $C' \subseteq C$, $D' \subseteq D$ and $R' \subseteq R$ are the restrictions of C, D , and R to the variables of V' . By the hypothesis, $T_{I=(a_1, \dots, a_{i-1}, a_i)}$ is a failure tree of I in \mathcal{P} . This implies that the assignment of X_i to the value a_i leads to a failure in \mathcal{P}' . In other words, a_i does not participate in any solution of \mathcal{P}' . By the hypothesis, the values a_i and b_i verify the condition of theorem 3.2 when restricted to the variables of $Var(T_{I=(a_1, \dots, a_{i-1}, a_i)})$. This means that a_i dominates b_i in the CSP \mathcal{P}' . By application of proposition 3.1, we deduce that the value b_i does not participate in any solution of \mathcal{P}' . This implies that the partial instantiation $J = I_0 \cup \{b_i\} = (a_1, \dots, a_{i-1}, b_i)$ is inconsistent in \mathcal{P} . (QED)

This previous property of dominance is a weakening of the conditions of Theorem 3.2 when the current partial instantiation leads to an inconsistency. The case of partial inconsistent instantiation is important, because it allows to prune the consistency proof tree of a CSP *w.r.t* Proposition 3.1.

Some dominances not captured by Theorem 3.2 can result from these weakened conditions. Let us consider for instance the CSP of Figure 1. If we take the inconsistent partial instantiation $I = (b, a)$ of the variables X_1 and X_2 , then the two values a and c of the domain of the current variable X_2

dominate each other (symmetrical) by application of Theorem 3.3, whereas the conditions of both theorems 3.1 and 3.2 are not verified. The branch corresponding to the assignment of X_2 to c is not explored in the consistency proof tree thanks to Theorem 3.3. This defines a dominance cut which we use in Section 4 to shorten the CSP search tree.

3.4 Dominance detection

Now we deal with the dominance detection problem. Dominance detection is based on the conditions of theorem 3.3. The algorithm sketched in Figure 3 computes the values dominated by a value a_i of a given domain D_i *w.r.t* the conditions of theorem 3.3. These values form the class of dominance of a_i which we denote by $cl(a_i)$.

```

procedure weak_dominance( $a_i \in D_i, Var(T_{I=(a_1, \dots, a_i)})$ ),
var  $cl(a_i)$ :class;
input: a value  $a_i \in D_i$ , a set of variables  $Var(T_{I=(a_1, \dots, a_i)})$ 
Output: the class  $cl(a_i)$  of the dominated values by  $a_i$ .
begin
     $cl(a_i) := \{a_i\}$ 
    for each  $d_i \in D_i - \{a_i\}$  do
        for each domain  $D_k$  of variables
        of  $Var(T_{I=(a_1, \dots, a_i)})$ 
            if ( $c_{ik} \in C$  and ( $a_i \in D_k \Rightarrow d_i \in D_k$ ))
            or
            ( $c_{ik} \notin C$  and ( $a_i \in D_k \Leftrightarrow d_i \in D_k$ ))
            then  $cl(a_i) := cl(a_i) \cup \{d_i\}$ 
end
    
```

Figure 3: The algorithm of dominance search in NECSPs

Complexity: 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 of Figure 3 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 dominated values with a complexity $\mathcal{O}(nd)$ in the worst case. This algorithm has a linear complexity *w.r.t* the NECSP size.

In theory, this algorithm has the same complexity as the one of the algorithm described in [Benhamou, 2004] in the worst case. But, this new algorithm detects dominance rather than only symmetry and detect some symmetries which are not detected by the algorithm in [Benhamou, 2004]. All the symmetries detected in [Benhamou, 2004] are detected with this new algorithm, since it works on a weakened dominance condition. That is, the dominance condition is verified on a reduced subset of the non-instantiated variables (the ones of $Var(T_{I=(a_1, \dots, a_i)})$) rather on the hole set of un-instantiated variable as it is done in [Benhamou, 2004].

4 Exploiting Dominance in NECSPs

Now, we show how the dominance property given in Theorem 3.3 is exploited to increase the efficiency of NECSP backtracking algorithms. This property can be exploited in all enumerative resolution methods. Here we implemented a *Simplified Forward Checking* method (denoted by SFC) adapted to NECSPs which we want improve by adding the dominance property.

The principle of the Forward Checking [R. M. Haralik and G. L. Elliot, 1980] is based on filtering the domains of the non-instantiated variables *w.r.t* the instantiated one.

In the case of NECSPs, the filtering is simplified. It consists only in removing the value d_i from the domains of the future variables having a constraint with the current variable v_i under instantiation. This results in a *Simplified Forward Checking* which we considered in our implementation. The rest of the method is just the classic backtracking.

```

Procedure SFC-weak-dom( $D, I, i, VFT$  : a list);
input: a set of domains  $D, I = (d_1, \dots, d_i)$  a partial instantiation
of variables  $\{v_1, \dots, v_i\}$ ;  $i$  the index of the current variable and  $VFT$ 
the set of variables  $Var(T_I)$  of the failure tree  $T_I$  (at the beginning  $VFT$  is empty).
var empty:boolean;
var VFT_tmp: a list;
var VFT_old: a list;
begin
  if  $i = n$  then  $[d_1, d_2, \dots, d_i]$  is a solution, print(l), stop
  else
    begin
      empty:=false;
      VFT_tmp:=VFT;
      VFT_old:=VFT;
      for each  $v_j \in V$ , such as  $C_{ij} \in C, v_j \in \text{future}(v_i)$  do
        if not(empty) and  $d_i \in D_j$  then
          begin
             $D_j = D_j - \{d_i\}$ ;
            if  $D_j = \emptyset$  then
              begin
                undo filtering effects;
                add( $v_j, VFT$ );
                empty:=true;
              end
            end
          end
        if not(empty) then
          begin
             $v_{i+1} = \text{next-variable}(v_i)$ 
            repeat
              take  $d_{i+1} \in D_{i+1}$ 
               $D_{i+1} = D_{i+1} - d_{i+1}$ 
               $I = [d_1, d_2, \dots, d_i, d_{i+1}]$ ;
              VFT_tmp:=VFT  $\cup$  VFT_tmp;
              VFT:=VFT_old;
              SFC-weak-dom( $D, I, i+1, VFT$ );
              weak_dominance( $d_{i+1} \in D_{i+1}, VFT, Cl(d_{i+1})$ );
               $D_{i+1} = D_{i+1} - Cl(d_{i+1})$ ;
            until  $D_{i+1} = \emptyset$ 
          end
          VFT:=VFT_tmp;
          add( $v_i, VFT$ );
        end
      end
    end
  end

```

Figure 4: The SFC method augmented by dominance

Theorem 3.3 allows to prune $k-1$ branches in the search tree if there are k dominated values by a dominant value which is shown to not participating in any solution. If $Cl(d_i)$ denote the class of values of the domain D_i which are dominated by d_i , then we consider only the value d_i , since the other values of $Cl(d_i)$ are redundant.

Figure 4 sketches the SFC procedure augmented by the dominance property of theorem 3.3 (notation SFC-weak-dom) which decides just the consistency of a NECSP. This method can be easily modified to compute all non-symmetrical solutions of a NECSPs. The structure *future*(v_i) encodes the set of non-instantiated variables remaining after the instantiation of v_i , and *next-variable* a function which encodes the (DomDeg) heuristic. It consists

in minimizing the ratio

$$r = \frac{|D_j|}{\text{Degree}(v_j)}$$

where $\text{Degree}(v_j)$ denotes the number of constraints of the initial CSP in which the variable v_j is involved to select the next variable. In the sequel SFC will denote the SFC method augmented by the (DomDeg) heuristic.

5 Experiments

We will now evaluate the performances of our implementation. The tests are made on both randomly generated graph coloring instances and some graph coloring benchmarks of the 2nd challenge of Dimacs (<http://dimacs.rutgers.edu/Challenges>). Graph coloring is trivially expressed as a NECSP. We will test and compare the Simplified Forward Checking augmented by the symmetry property defined in [Benhamou, 2004] (SFC-sym), the Simplified Forward Checking augmented by the advantage of the dominance property of theorem 3.3 (SFC-weak-dom) and an improved version [Sewell, 1995] of the well known method DSATUR [D. Brelaz, 1979]. This method is based on a heuristic which consists in coloring the vertices of a graph according to their *saturation degree*. The saturation degree of a vertex is the number of different colors to which it is adjacent. The DSATUR heuristic repeatedly chooses a vertex having a maximal saturation degree and colors it with the lowest-numbered color possible. The complexity indicators are the number of nodes and CPU time. The source code is written in C and compiled on a P4 2.8 GHz - RAM 1 Go.

5.1 Random graph coloring problems

Random graph coloring problems are generated according to the parameters: (1) n the number of vertices (the variables), (2) Cl s the number of colors (the domain values) and (3) d the density which is the ratio expressing the number of constraints to the total number of possible constraints. For each test corresponding to some fixed value of the parameters n , Cl s and d , a sample of 100 instances are randomly generated and the measures (CPU time, nodes) are taken in average.

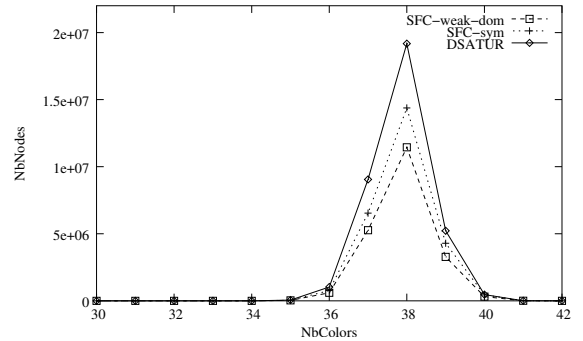


Figure 5: The curves representing the number of nodes

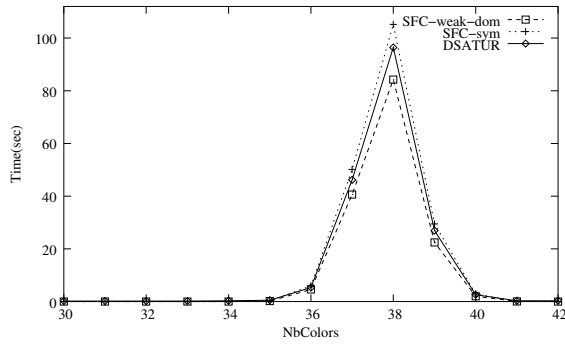


Figure 6: The curves representing the CPU times

Figures 5 and 6 give the performances of the methods DSATUR, SFC-sym and SFC-weak-dom in number of nodes of the search tree, respectively, in CPU time (in seconds) on random graph coloring problems where the number of variables is fixed to $n = 100$ and the density to $d = 0.9$. SFC without symmetry was not able to solve these instances in reasonable time. This is why we did not give its curve. We can see that SFC-weak-dom, generates less nodes than both DSATUR and SFC-sym, and spends less time than both methods to solve the problems. This proves that SFC-weak-dom detects and eliminates more symmetries than the other methods and the detection is faster in SFC-weak-dom. The symmetry behavior is now shown. DSATUR eliminates only the global¹ symmetry between colors, then generate more nodes than FC-sym. SFC-sym detects more symmetries than DSATUR since it detects both the global symmetry and local² symmetry, but less than SFC-dom-weak which considers dominance. Only SFC-weak-dom captures the dominance, and the detection is faster than in SFC-sym, thanks to the weakened condition. These remarks are confirmed by the following Dimacs benchmarks where the gain is important.

5.2 Dimacs graph coloring benchmarks

Table 1 shows the results of the methods on some graph coloring benchmarks of Dimacs. It gives the number of nodes of the search tree and the CPU time for each method. We seek for each of them the minimal number k of colors needed to color the vertices of the corresponding graph (the chromatic number). The search of the chromatic number consist in proving the consistency of the problem with k colors (existence of a k -coloration of the graph); and in proving its inconsistency when using $k - 1$ colors. The symbol “-” means that the corresponding method does answer the question in one hour.

We can remark that only SFC-weak-dom is able to solve the known hard problem “DSJR500.1c” which, as far as, we know it has never been solved by an exact method. We can see that SFC-dom-weak outperforms both DSATUR and SFC-sym, and SFC-sym is better than DSATUR on these problems. DSATUR is the less effective since it does not

¹The trivial symmetry of the initial problem which consists in the interchangeable colors.

²The existing symmetry between values at each node of the search tree.

Pb instances ($V - E$)	k	DSATUR		SFC-SYM		SFC-dom-weak	
		N	T	N	T	N	T
queen8.8 (64-728)	9	1581661	4.3	1368441	6.1	1353680	6.1
queen8.12 (96-1368)	12	162	0.0	460	0.0	460	0.0
myciel5 (47-236)	6	378310	0.6	72966	0.2	21278	0.0
myciel6 (95-755)	7	-	-	83157279	556.0	29754513	190.2
le450.5a (450-5714)	5	-	-	1408	0.1	1395	0.1
le450.5b (450-5734)	5	-	-	19884	0.6	19763	0.5
le450.25a (450-8260)	25	425	0.1	450	0.1	450	0.1
le450.25b (450-8263)	25	425	0.0	450	0.1	450	0.1
1-FullIns.3 (30-100)	4	37	0.0	51	0.0	50	0.0
1-FullIns.4 (93-593)	5	-	-	8885	0.0	1368	0.0
2-FullIns.3 (52-201)	5	156663424	193.6	678	0.0	359	0.0
qq.order30 (900-26100)	30	1680	0.2	1169	0.2	1162	0.2
qq.order40 (1600-62400)	40	-	-	12089785	302.0	10814593	266.6
school.1 (385-19095)	14	371	0.2	568	0.4	555	0.4
school_nsh (352-14612)	14	338	0.2	352	0.4	352	0.4
wap05a (905-43081)	50	855	1.3	905	1.4	905	1.4
mug88.25 (88-146)	4	-	-	22643	0.0	1631	0.0
mug100.25 (100-166)	4	-	-	99917	0.2	515	0.0
ash608GPIA (1216-7844)	4	10242	0.5	1742	0.5	1707	0.5
ash958GPIA (1916-12506)	4	-	-	10252	2.2	7167	1.6
R125.5 (125-3838)	36	1357573	9.1	55952	0.4	1051	0.0
DSJR500.1c (500-121275)	84	-	-	-	-	28044984	3096.0

Table 1: Dimacs graph coloring benchmarks

succeed to solve 9 benchmarks among the 22 proposed. For space reason, we report here the results on the most relevant Dimacs problems to compare DSATUR and our method, but it is important to inform the reader that all the others DIMACS problems which are solved by DSATUR are also solved by SFC-weak-dom with a comparable performance.

6 Some related works

- In [P. Van Hentenryck *et al.*, 2003] authors studied three classes of CSPs where 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 [P. Van Hentenryck *et al.*, 2003] for the ICSP class seems to be equivalent to the global symmetry elimination described in [Benhamou, 2004]. But, in general NECSPs are not ICSPs and the dominance/symmetry detected by the procedure of figure 3 are not considered in the ICSP symmetry breaking.
- On other hand Gent introduced in [I. Gent, 2001] a symmetry constraint to eliminate what he calls indistinguishable values. His approach works by addition of symmetry constraints rather than dynamic detection of symme-

try. This technique may be used to break some of the trivial global symmetry such as the one of the graph coloring problem for example. However, it does not deal with the local symmetry or the dominance which we detect by using the dominance procedure of figure 3.

- In [A. Ramani *et al.*, 2004], authors investigated a static approach to break symmetry in exact graph coloring problem reduced to a hybrid constraint representation formed by: boolean constraints and pseudo boolean constraints resulting from a 0-1 ILP reduction of the problem. Two kinds of symmetry are exploited: the instance-dependent symmetry and the instance-independent symmetry. Our approach is dynamic whereas theirs is static and does not deal with dominance. These are two different approaches which can be combined. Our method seems to be more efficient for solving graph coloring problems.
- The GE-tree method [Colva M. Roney-Dougal *et al.*, 2004] breaks all value symmetries at each node of the search tree of a CSP. This method can eventually be used to break the symmetries we are dealing with in NECSPs during search, but it will be time consuming, since its complexity at each node of the search tree, is approximately $O(n^4 d^4)$ in the worst case. Our method detects the needed class of symmetry of a value in $O(nd)$ and detects dominance which is not considered by the GE-tree technique.
- Recently in [Puget, 2005b; 2004], Puget studied symmetry between variables involved in a global AllDiff constraint. This is a particular case of NECSP, since the AllDiff constraint can be expressed as a NECSP whose constraint graph is complete. The symmetry elimination technique is static, it consists in adding some extra constraints to the problem formulation to eliminate the symmetries between variables. The most important result here is the fact that only a linear number of extra constraints is needed to break all the variable global symmetries. Our approach is dynamic, it focuses on local dominance/symmetry between values. Our method can be safely combined with Puget's one. It will be interesting to study the behavior of the resulting method on complete NECSPs (AllDiff constraints).

7 Conclusion

In this work we extended the symmetry principle to dominance and weakened the symmetry/dominance sufficient conditions in Not-Equals constraint networks when an inconsistent partial instantiation is generated. We implemented a more efficient dominance search algorithm which detects both symmetry and which captures the dominance. We exploited the new dominance property in a Simplified Forward Checking backtracking algorithm adapted to NECSPs. Experiments are carried on both random generated graph coloring problems and Dimacs graph coloring benchmarks. The obtained results show that reasoning by dominance is profitable for solving NECSPs. Our method beats the well known exact method for solving graph coloring.

Further investigation consists first in extending the dominance property to values of different domains, then investigate dominance detection in more general CSPs. Another point is to combine some CSP decomposition methods with our method to improve Not-Equals constraint networks solving.

References

- [A. Ramani *et al.*, 2004] A. Ramani, F.A. Aloul, I.L. Markov, and K.A. Sakallah. Breaking instance-independent symmetries in exact graph coloring. In *DATE*, pages 324–329, 2004.
- [Benhamou and Sais, 1992] B. Benhamou and L. Sais. Theoretical study of symmetries in propositional calculus and application. *CADE-11, Saratoga Springs, NY, USA*, 1992.
- [Benhamou, 1994a] B. Benhamou. Study of symmetry in constraint satisfaction problems. In *PPCP'94*, 1994.
- [Benhamou, 1994b] B. Benhamou. Theoretical study of dominance in constraint satisfaction problems. *6th International Conference on Artificial Intelligence: Methodology, Systems and Applications (AIMSA-94), Sofia, Bulgaria, september*, pages 91–97, 1994.
- [Benhamou, 2004] B. Benhamou. Symmetry in not-equals binary constraint networks. *SymCon'04 : 4th International Workshop on Symmetry and Constraint Satisfaction Problems*, 2004.
- [Cohen *et al.*, 2005] D. Cohen, P. Jeavons, C. Jefferson, K.E. Petrie, and B. Smith. Symmetry definitions for constraint satisfaction problems. In *proceedings of CP*, pages 17–31, 2005.
- [Colva M. Roney-Dougal *et al.*, 2004] Colva M. Roney-Dougal, Ian P. Gent, Tom Kelsey, and Steve A. Linton. Tractable symmetry breaking using restricted search trees. In *proceedings of ECAI-04*, 2004.
- [D. Brelaz, 1979] D. Brelaz. New methods to color the vertices of a graph. In *the communications of the ACM* 22, pages 251–256, 1979.
- [F. Focacci and M. Milano, 2001] F. Focacci and M. Milano. Global cut framework for removing symmetries. In *CP'01*, volume 2239 of *LNCS*, pages 77–82. Springer Verlag, 2001.
- [F.A. Aloul *et al.*, 2003] F.A. Aloul, A. Ramani, I.L. Markov, and K.A. Sakallah. Solving difficult sat instances in the presence of symmetry. In *IEEE Transaction on CAD*, vol. 22(9), pages 1117–1137, 2003.
- [F.A. Aloul *et al.*, 2004] F.A. Aloul, A. Ramani, I.L. Markov, and K.A. Sakallah. Symmetry breaking for pseudo-boolean satisfiability. In *ASPAC'04*, pages 884–887, 2004.
- [Fahle *et al.*, 2001] 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.

- [Freuder, 1991] E.C. Freuder. Eliminating interchangeable values in constraints satisfaction problems. *Proc AAAI-91*, pages 227–233, 1991.
- [I. Gent, 2001] I. Gent. A symmetry breaking constraint for indistinguishable values. In *SymCon'01*, 2001.
- [I.P. Gent *et al.*, 2002] I.P. Gent, W. Harvey, and T. Kelsey. Groups and constraints: Symmetry breaking during search. In *CP'02*, volume 2470 of *LNCS*, pages 415–430. Springer Verlag, 2002.
- [James Crawford *et al.*, 1996] James Crawford, Matthew L. Ginsberg, Eugene Luck, and Amitabha Roy. Symmetry-breaking predicates for search problems. In *KR'96*, pages 148–159. Morgan Kaufmann, San Francisco, California, 1996.
- [Jean F. Puget, 2002] Jean F. Puget. Symmetry breaking revisited. In *CP'02*, volume 2470 of *LNCS*, pages 446–461. Springer Verlag, 2002.
- [Krishnamurty, 1985] B. Krishnamurty. Short proofs for tricky formulas. *Acta informatica*, (22):253–275, 1985.
- [M.R. Garey and D.S. Johnson, 1979] M.R. Garey and D.S. Johnson. Computers and intractability: A guide to the theory of np-completeness, w.h. freeman. 1979.
- [P. Van Hentenryck *et al.*, 2003] P. Van Hentenryck, P. Flener, J. Pearson, and M. Argen. Tractable symmetry breaking for csps with interchangeable values. In *IJCAI*, pages 277–282, 2003.
- [Puget, 1993] Jean F. Puget. On the satisfiability of symmetrical constrained satisfaction problems. In *ISMIS*, 1993.
- [Puget, 2004] Jean F. Puget. Breaking symmetries in all different problems. *SymCon'04 : 4th International Workshop on Symmetry and Constraint Satisfaction Problems*, 2004.
- [Puget, 2005a] Jean F. Puget. Breaking all value symmetries in surjection problems. In *proceedings of CP*, pages 490–504, 2005.
- [Puget, 2005b] Jean F. Puget. Breaking symmetries in all different problems. In *proceedings of IJCAI*, pages 272–277, 2005.
- [R. M. Haralik and G. L. Elliot, 1980] R. M. Haralik and G. L. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence 14*, pages 263–313, 1980.
- [Sewell, 1995] Edward C. Sewell. An improved algorithm for exact graph coloring. *DIMACS series on Discrete Mathematics and Theoretical Computer Science*, 1995.