

Preface

Symmetry has become a topic of significant interest in the constraint programming and related communities. Many constraint satisfaction problems (CSPs) have symmetries in the variables, domains or constraints - or any combination thereof. Each of these symmetries preserve satisfiability, so that when there is symmetry in a CSP, any assignment can be transformed into an equivalent assignment without affecting whether or not it satisfies the constraints. Similarly, applying such a transformation to a partial assignment does not affect whether or not it can be extended to an assignment satisfying the constraints. For instance, in many CSPs some of the variables refer to entities which are indistinguishable, and the values assigned to these variables can be interchanged in any solution.

Symmetry increases the combinatorial complexity of CSPs. In the presence of symmetry, a constraint solver may waste a large amount of time considering symmetric but equivalent assignments or partial assignments. Hence, dealing with symmetry is often crucial to the success of solving such CSPs efficiently.

As well as exploiting symmetry when solving CSPs, CSP solving techniques have been used to solve symmetry-related problems. For example, they have been used to answer the question of whether a particular search state is symmetrically equivalent to one already explored. As another example, they have been used to derive "generators" of a symmetry group, which allow the symmetries to be represented effectively without the need to list them all explicitly. Constraint programming techniques have the potential to improve on existing algorithms for solving these and related group-theoretic problems.

SymCon'04 is the fourth workshop in the series, following the successful earlier workshops at CP 2001 in Paphos, Cyprus, at CP 2002 in Ithaca, NY, USA, and at CP 2003 in Cork, Ireland. The papers in these proceedings present research into many aspects of symmetry in CSPs and related disciplines. The number of papers shows that symmetry is an active area of research in constraint programming, and it is hoped that they will stimulate further research. We also hope that the workshop will foster a cross-discipline exchange of ideas. In order to encourage this, the workshop includes a panel session on the methods being used to tackle symmetry in various disciplines related to constraints, and the invited talk is on symmetry in integer programming.

We would like to thank all the authors who submitted papers; the invited speaker, François Margot; the members of the "Symmetry in related disciplines" panel; and the members of the Programme Committee. We also thank Filippo Focacci, Chris Jefferson and Brendan McKay for their help in reviewing papers.

These proceedings can be found online at <http://www.dis.uu.se/SymCon04/>.

August 2004

Warwick Harvey
Zeynep Kiziltan
(Programme Chairs)

Programme Committee

Rolf Backofen, Universität Jena, Germany
Belaïd Benhamou, Université de Provence (Aix-Marseille I), France
Ian Gent, University of St Andrews, U.K.
Warwick Harvey, IC-Parc, Imperial College London, U.K.
Zeynep Kiziltan, Università di Bologna, Italy
Igor Markov, The University of Michigan Ann Arbor, U.S.A.
Pedro Meseguer, IIIA-CSIC, Spain
Ian Miguel, University of York, U.K.
Michela Milano, Università di Bologna, Italy
Steve Prestwich, University College Cork, Ireland
Jean-François Puget, ILOG, France
Meinolf Sellmann, Brown University, U.S.A.
Barbara Smith, Cork Constraint Computation Centre, Ireland
Pascal Van Hentenryck, Brown University, U.S.A.
Toby Walsh, NICTA and UNSW, Australia

Workshop Schedule

08:55 - 09:00	Welcome	
09:00 - 09:25	Combining Branch & Bound and SBDD to solve Soft CSPs <i>Stefano Bistarelli and Barry O'Sullivan</i>	9
09:25 - 09:50	A Note on the Compatibility of Static Symmetry Breaking Constraints and Dynamic Symmetry Breaking Methods <i>Warwick Harvey</i>	42
09:50 - 10:15	Breaking Symmetries in All Different Problems <i>Jean-François Puget</i>	71
10:15 - 10:40	Symmetry-Breaking and Local Search: A Case Study <i>Andrea Roli</i>	88
10:40 - 11:05	Coffee break	
11:05 - 12:05	Invited Talk Symmetry in Integer Programming <i>François Margot</i>	1
12:05 - 12:30	Comparing the Use of Symmetry in Constraint Processing and Model Checking <i>Alastair Donaldson, Alice Miller and Muffy Calder</i>	18
12:30 - 14:00	Lunch break	
14:00 - 14:25	On the Extraction of Disjunctive Landmarks from Planning Problems via Symmetry Reduction <i>Peter Gregory, Stephen Cresswell, Derek Long and Julie Porteous</i>	34
14:25 - 15:25	Panel Session Symmetry in related disciplines	
15:25 - 15:50	Coffee break	
15:50 - 16:15	Approaches to Conditional Symmetry Breaking <i>Ian P. Gent, Iain McDonald, Ian Miguel and Barbara M. Smith</i>	26
16:15 - 16:40	Conditional Interchangeability and Substitutability <i>Yuanlin Zhang and Eugene C. Freuder</i>	95
16:40 - 17:05	Exploiting Dominance in Three Symmetric Problems <i>Steven Prestwich and J. Christopher Beck</i>	63
17:05 - 17:30	Automatically Exploiting Symmetries in CP <i>Arathi Ramani and Igor L. Markov</i>	79
	Close	

Contents

Invited Talk: Symmetry in Integer Programming	
<i>François Margot</i>	1
Symmetry in Not-Equals Binary Constraint Networks	
<i>Belaïd Benhamou</i>	2
Combining Branch & Bound and SBDD to solve Soft CSPs	
<i>Stefano Bistarelli and Barry O'Sullivan</i>	9
Comparing the Use of Symmetry in Constraint Processing and Model Checking	
<i>Alastair Donaldson, Alice Miller and Muffy Calder</i>	18
Approaches to Conditional Symmetry Breaking	
<i>Ian P. Gent, Iain McDonald, Ian Miguel and Barbara M. Smith</i>	26
On the Extraction of Disjunctive Landmarks from Planning Problems via Symmetry Reduction	
<i>Peter Gregory, Stephen Cresswell, Derek Long and Julie Porteous</i>	34
A Note on the Compatibility of Static Symmetry Breaking Constraints and Dynamic Symmetry Breaking Methods	
<i>Warwick Harvey</i>	42
Breaking Weak Symmetries	
<i>Roland Martin and Karsten Weihe</i>	48
Combining SBDS and SBDD	
<i>Karen Petrie</i>	55
Exploiting Dominance in Three Symmetric Problems	
<i>Steven Prestwich and J. Christopher Beck</i>	63
Breaking Symmetries in All Different Problems	
<i>Jean-François Puget</i>	71
Automatically Exploiting Symmetries in CP	
<i>Arathi Ramani and Igor L. Markov</i>	79
Symmetry-Breaking and Local Search: A Case Study	
<i>Andrea Roli</i>	88
Conditional Interchangeability and Substitutability	
<i>Yuanlin Zhang and Eugene C. Freuder</i>	95

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.

Belaïd.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 lost 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 approach 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: $\sigma : \cup_{i \in [1, n]} D_i \longrightarrow \cup_{i \in [1, n]} D_i$, 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 $[\forall R_{ij} \in R, < d_i, d_j > \in \text{tuples}(R_{ij}) \Rightarrow < \sigma(d_i), \sigma(d_j) > \in \text{tuples}(R_{ij})]$.

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.

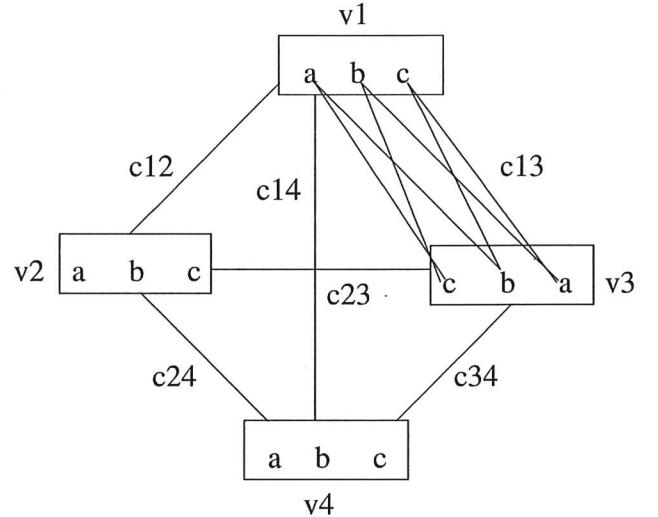


Figure 1: Pigeon-hole problem for 4 pigeons and 3 holes.

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 interchangeabilities. 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 $\sigma(t_1) = t_2 \in R_{ij}$. 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-instanciated 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 $(a_i, b_i)o(b_i, a_i)$ 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 offigure 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-instanciated variable
        we have
            ( $d_i \in D_k$  and  $a_i \in D_k$ )
            or ( $d_i \notin D_k$  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, Haselböck 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:

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

where v_i is a non-instantiated variable, $|D_i|$ its domain size and $\text{Degree}(v_i)$ 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 $d_i \sim \acute{d}_i$ and d_i doesn't participate in any solution of \mathcal{P} , then \acute{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 $\text{SymClass}(d)$ 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 $\text{SymClass}(d)$ 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( $d_k \in D_k, \text{SymClass}(d_k)$ );
                 $D_k=D_k - \text{SymClass}(d_k)$ ;
                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( $D, I, k + 1$ );
            untill  $D_{k+1} = \emptyset$ 
          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) Cl_{s} 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			
	Cl_{s}	Nodes	Time	Cons	Nodes	Time	Cons
8	2667	0.0	0.0	9	0.0	0.0	0.0
9	48939	0.03	0.0	16	0.0	0.0	0.0
10	2263385	15.55	3.0	66	0.0	3.0	0.0
11	11551093	66.87	73.0	158	0.0	73.0	0.0
12	643068	3.65	99.0	59	0.0	99.0	0.0
13	230	0.0	100.0	57	0.0	100.0	0.0

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

SFC-sym			
Cl_{s}	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 (Cl_{s}), 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			
	Colors	Nodes	Time	Cons?
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	
	Nodes	Time
Pigeons		
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 discreteconstraint 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. Haselböck. Exploiting interchangeability in constraint satisfaction problems. In *Proceedings of IJCAI*, pages 282–287, 1993.
- [16] P. Van Hentenryck, P. Flener, J. Pearson, and M. Argon. Tractable symmetry breaking for csp's 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 Internationnal 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 csp's. 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.

Combining Branch&Bound and SBDD to solve Soft CSPs*

Stefano Bistarelli[†]

Dipartimento di Scienze

Università di Chieti-Pescara, Italy

bista@sci.unich.it

Istituto di Informatica e Telematica, C.N.R.

Pisa, Italy

stefano.bistarelli@iit.cnr.it

Abstract

As constraint processing applications are becoming more widespread in areas such as electronic commerce, configuration, etc., it is becoming increasingly important that we can reason about preferences as efficiently as possible. In this paper we extend some existing results dealing with symmetry in the semiring framework for soft constraints. In particular we extend existing definitions of symmetry to partial instantiations. We also present Soft-SBDD, a generalization of Symmetry Breaking via Dominance Detection, and present theoretical results demonstrating that symmetry breaking in soft constraint satisfaction problems improves the efficiency of search.

1 Introduction

Exploiting symmetry in constraint satisfaction problems has become a very popular topic of research in recent times [Backofen and Will, 1999; Benhamou, 1994; Flener *et al.*, 2002; Gent and Smith, 2000; McDonald and Smith, 2002; Puget, 2002; Fahle *et al.*, 2001; Focacci and Milano, 2001]. The existence of symmetry in a problem has the effect of artificially increasing the size of the search space that is explored by search algorithms. Therefore, a typical approach is to break the symmetries in the problem so that only unique solutions are returned (i.e. that only one exemplar of each symmetric equivalence class of solutions is returned). The complete set of solutions can be trivially computed using the symmetry in the problem. The significant advantage is that not only do we return fewer solutions, but we also reduce the search effort required to find these solutions by eliminating symmetric branches of the search tree.

Another significant topic of research in the constraint processing community is the ability to reason about prefer-

*This work has received support from Enterprise Ireland under their Basic Research Grant Scheme (Grant Number SC/02/289) and their International Collaboration Programme (Grant Number IC/2003/88). It has also received support from Science Foundation Ireland (Grant Number 00/PI.1/C075).

[†]Part of this research was carried out while this author was visiting the Cork Constraint Computation Centre, University College Cork, Ireland.

Barry O'Sullivan

Cork Constraint Computation Centre

Department of Computer Science

University College Cork

Ireland

b.osullivan@cs.ucc.ie

ences [Bistarelli *et al.*, 1999; Junker, 2001]. It has been shown how preferences can be modeled as constraints [Bistarelli *et al.*, 1997; Domshlak *et al.*, 2003]. As constraint processing applications are becoming more widespread in areas such as electronic commerce, configuration, etc., it is becoming increasingly important that we can reason about preferences in as efficient a manner as possible. One obvious avenue to be explored here are notions of symmetry in preferences. For example, we might seek to find a “diverse” set of solutions to a soft CSP, where diversity might be interpreted as the presentation of a set of solutions which are members of different symmetric equivalence classes.

In this paper we extend the approach to dealing with symmetry in the semiring framework for soft constraints [Bistarelli *et al.*, 1997; 2002; Bistarelli, 2004] presented in [Bistarelli *et al.*, 2003b], giving new important results. In particular we extend existing definitions of symmetry to partial instantiations. We present Soft-SBDD, a generalization of Symmetry Breaking via Dominance Detection, and present theoretical results demonstrating that symmetry breaking in soft constraint satisfaction problems improves the efficiency of search.

The remainder of the paper is structured as follows. Section 2 presents a review of soft constraints and of symmetry breaking in crisp and soft CSPs. We present the new theoretical results of our approach to symmetry breaking in soft CSPs in Section 3. Theoretically we demonstrate the utility of symmetry breaking in SCSPs in Section 4. Some concluding remarks are made in Section 5.

2 Background

Before recalling our approach to dealing with symmetry in soft CSPs [Bistarelli *et al.*, 2003b], we present a review of the state-of-the-art in symmetry breaking (Section 2.1) and in soft constraints (Section 2.2).

2.1 Symmetry Breaking

There is significant interest within the constraint programming community in exploiting symmetry when solving constraint satisfaction problems. As a consequence, a growing number of techniques are being reported in the literature. Benhamou [Benhamou, 1994] presented an early analysis of symmetry-breaking and placed it in the context of Freuder’s work on interchangeability, a special case of sym-

metry [Freuder, 1991].

A common approach to symmetry breaking involves carefully modeling the problem so that symmetries have been removed. For example, Crawford *et al.* [Crawford *et al.*, 1996] have demonstrated how constraints can be added to the model in order to break symmetries. Puget [Puget, 1993] has presented a formal approach to symmetry breaking that involves the addition of ordering constraints to break symmetries. Flener *et al.* [Flener *et al.*, 2002] adopt a similar approach by adding ordering constraints to break symmetries in matrix models. Flener *et al.* [Flener *et al.*, 2002] also remind us that symmetry detection is graph-isomorphism complete in the general case, pointing to the work of Crawford [Crawford, 1992].

Brown *et al.* [Brown *et al.*, 1988] have presented a modified backtracking algorithm that breaks symmetry by pruning branches of the search tree dynamically. This is done by ensuring that only one solution from each symmetric equivalence class is computed. Similarly, a general method for eliminating symmetries, known as symmetry breaking during search (SBDS), has been proposed by Gent and Smith [Gent and Smith, 2000]. The SBDS approach is based on earlier work by Backofen and Will [Backofen and Will, 1999]. Both of these methods can be regarded as examples of a class of approaches to handling symmetries that involve the addition of constraints during search to avoid symmetrical states in the search space. An implementation of SDBS based on the GAP computational abstract algebra system has been presented by Gent *et al.* [Gent *et al.*, 2002].

Meseguer and Torras [Meseguer and Torras, 2001] have reported the use of search ordering heuristics to avoid symmetries during search. However, the method is less general than SBDS [Gent and Smith, 2000].

The notion of partial symmetry breaking has been explored by McDonald and Smith [McDonald and Smith, 2002]. They show that there is a break-even point to be considered when breaking symmetries during search; there is a point where the benefit in reducing search from removing more symmetries is outweighed by the extra overhead incurred. By breaking a subset of the possible symmetries in a problem, rather than breaking all of them, significant savings in runtime can be accomplished. It is worth noting that most static symmetry-breaking schemes (e.g. those of Flener *et al.* [Flener *et al.*, 2002]) are partial.

Finally, symmetry breaking based on nogood recording methods have been presented by Fahle *et al.* [Fahle *et al.*, 2001] and Focacci and Milano [Focacci and Milano, 2001]. The approach presented by the former is known as symmetry-breaking via dominance detection (SBDD) and it has been shown to compare well with SBDS; the latter approach is known as the global cut framework. Puget has presented an improvement on these approaches by using an auxiliary CSP for performing dominance checks based on nogood recording [Puget, 2002].

Recently, inspired by the development of the global cut framework/SBDD, Focacci and Shaw showed that dominance detection cannot only be used to prune under symmetric dominance, but under any dominance relation, especially under dominance of objective cost (they used this idea to exploit

local search in a complete solver for the TSP with time windows) [Focacci and Shaw, 2002].

However, dominance has also been exploited in other contexts. For example, the pure literal rule used in SAT solvers can be regarded as a form of dominance exploitation.

2.2 Soft CSPs

Several formalizations of the concept of *soft constraints* are currently available. In the following, we refer to the one based on c-semirings [Bistarelli, 2001; Bistarelli *et al.*, 1995; 1997; 2002], which can be shown to generalize and express many of the others [Bistarelli *et al.*, 1999]. A soft constraint may be seen as a constraint where each instantiations of its variables has an associated value from a partially ordered set which can be interpreted as a set of preference values. Combining constraints will then have to take into account such additional values, and thus the formalism has also to provide suitable operations for combination (\times) and comparison ($+$) of tuples of values and constraints. This is why this formalization is based on the concept of c-semiring, which is just a set plus two operations.

Semirings. A semiring is a tuple $\langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ such that: 1. A is a set and $\mathbf{0}, \mathbf{1} \in A$; 2. $+$ is commutative, associative and $\mathbf{0}$ is its unit element; 3. \times is associative, distributes over $+$, $\mathbf{1}$ is its unit element and $\mathbf{0}$ is its absorbing element. A c-semiring is a semiring $\langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ such that: $+$ is idempotent, $\mathbf{1}$ is its absorbing element and \times is commutative. Let us consider the relation \leq_S over A such that $a \leq_S b$ iff $a + b = b$. Then it is possible to prove that (see [Bistarelli *et al.*, 1997]): 1. \leq_S is a partial order; 2. $+$ and \times are monotone on \leq_S ; 3. $\mathbf{0}$ is its minimum and $\mathbf{1}$ its maximum; 4. $\langle A, \leq_S \rangle$ is a complete lattice and, for all $a, b \in A$, $a + b = lub(a, b)$ (where *lub* is the *least upper bound*). Moreover, if \times is idempotent, then: $+$ distributes over \times ; $\langle A, \leq_S \rangle$ is a complete distributive lattice and \times its *glb* (*greatest lower bound*). Informally, the relation \leq_S gives us a way to compare semiring values and constraints. In fact, when we have $a \leq_S b$, we will say that *b is better than a*. In the following, when the semiring will be clear from the context, $a \leq_S b$ will be often indicated by $a \leq b$.

Constraint Problems. Given a semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ and an ordered set of variables V over a finite domain D , a *constraint* is a function which, given an assignment $\eta : V \rightarrow D$ of the variables, returns a value of the semiring. By using this notation we define $\mathcal{C} = \eta \rightarrow A$ as the set of all possible constraints that can be built starting from S, D and V .

Note that in this *functional* formulation, each constraint is a function (as defined in [Bistarelli *et al.*, 2002]) and not a pair (as defined in [Bistarelli *et al.*, 1995; 1997]). Such a function involves all the variables in V , but it depends on the assignment of only a finite subset of them. So, for instance, a binary constraint $c_{x,y}$ over variables x and y , is a function $c_{x,y} : V \rightarrow D \rightarrow A$, but it depends only on the assignment of variables $\{x, y\} \subseteq V$. We call this subset the *support* of the constraint. More formally, consider a constraint $c \in \mathcal{C}$. We define its support as $supp(c) = \{v \in V \mid \exists \eta, d_1, d_2. c\eta[v :=$

$d_1] \neq c\eta[v := d_2]\}$, where

$$\eta[v := d]v' = \begin{cases} d & \text{if } v = v', \\ \eta v' & \text{otherwise.} \end{cases}$$

Note that $c\eta[v := d]$ means $c\eta'$ where η' is η modified with the assignment $v := d$ (that is the operator $[]$ has precedence over application). Note also that $c\eta$ is the application of a constraint function $c : V \rightarrow D \rightarrow A$ to a function $\eta : D \rightarrow A$; what we obtain, is a semiring value $c\eta = a$.

A soft constraint satisfaction problem is a pair $\langle C, con \rangle$ where $con \subseteq V$ and C is a set of constraints: con is the set of variables of interest for the constraint set C , which may concern also variables not in con . Note that a classical CSP is a SCSP where the chosen c-semiring is: $S_{CSP} = \langle \{\text{false}, \text{true}\}, \vee, \wedge, \text{false}, \text{true} \rangle$. Fuzzy CSPs [Bowen et al., 1992; Schiex, 1992] can instead be modeled in the SCSP framework by choosing the c-semiring $S_{FCSP} = \langle [0, 1], \max, \min, 0, 1 \rangle$. Many other “soft” CSPs (Probabilistic, weighted, ...) can be modeled by using a suitable semiring structure ($S_{prob} = \langle [0, 1], \max, \times, 0, 1 \rangle$, $S_{weight} = \langle \mathcal{R}, \min, +, +\infty, 0 \rangle, \dots$).

Figure 1 shows the graph representation of a fuzzy CSP. Variables and constraints are represented respectively by nodes and by undirected (unary for c_1 and c_3 and binary for c_2) arcs, and semiring values are written to the right of the corresponding tuples. The variables of interest (that is the set con) are represented with a double circle. Here we assume that the domain D of the variables contains only elements a and b and c .

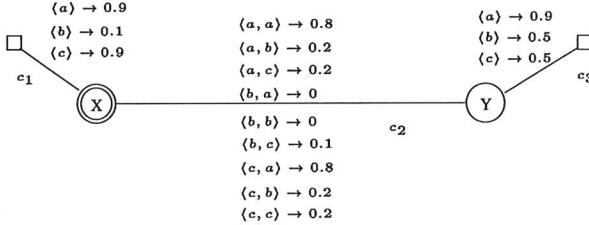


Figure 1: A fuzzy CSP.

Combining and projecting soft constraints. Given the set \mathcal{C} , the combination function $\otimes : \mathcal{C} \times \mathcal{C} \rightarrow \mathcal{C}$ is defined as $(c_1 \otimes c_2)\eta = c_1\eta \times_S c_2\eta$. Informally, combining two constraints means building a new constraint whose support involves all the variables of the original ones, and which associates with each tuple of domain values for such variables a semiring element which is obtained by multiplying the elements associated by the original constraints to the appropriate sub-tuples. It is easy to verify that $\text{supp}(c_1 \otimes c_2) \subseteq \text{supp}(c_1) \cup \text{supp}(c_2)$.

Given a constraint $c \in \mathcal{C}$ and a variable $v \in V$, the projection of c over $V - \{v\}$, written $c \Downarrow_{(V - \{v\})}$ is the constraint c' s.t. $c'\eta = \sum_{d \in D} c\eta[v := d]$. Informally, projecting means eliminating some variables from the support. This is done by associating with each tuple over the remaining variables a semiring element which is the sum of the elements associated by the original constraint to all the extensions of this tuple

over the eliminated variables. In short, combination is performed via the multiplicative operation of the semiring, and projection via the additive one.

Solutions. A solution of an SCSP $P = \langle C, con \rangle$ is the constraint $\text{Sol}(P) = (\bigotimes C) \Downarrow_{con}$. That is, we combine all constraints, and then project over the variables in con . In this way we get the constraint with support (not greater than) con which is “induced” by the entire SCSP. Note that when all the variables are of interest we do not need to perform any projection.

For example, the solution of the fuzzy CSP of Figure 1 associates a semiring element to every domain value of variable x . Such an element is obtained by first combining all the constraints together. For instance, for the tuple $\langle a, a \rangle$ (that is, $x = y = a$), we have to compute the minimum between 0.9 (which is the value assigned to $x = a$ in constraint c_1), 0.8 (which is the value assigned to $\langle x = a, y = a \rangle$ in c_2) and 0.9 (which is the value for $y = a$ in c_3). Hence, the resulting value for this tuple is 0.8. We can do the same work for tuple $\langle a, b \rangle \rightarrow 0.2$, $\langle a, c \rangle \rightarrow 0.2$, $\langle b, a \rangle \rightarrow 0$, $\langle b, b \rangle \rightarrow 0$, $\langle b, c \rangle \rightarrow 0.1$, $\langle c, a \rangle \rightarrow 0.8$, $\langle c, b \rangle \rightarrow 0.2$ and $\langle c, c \rangle \rightarrow 0.2$. The obtained tuples are then projected over variable x , obtaining the solution $\langle a \rangle \rightarrow 0.8$, $\langle b \rangle \rightarrow 0.1$ and $\langle c \rangle \rightarrow 0.8$.

When solving a crisp CSP we refer only to either finding one solution or finding all solutions. In the context of Soft CSP, solving can assume several meanings. Specifically, depending on the application and context, we may want to find:

1. all best solutions;
2. one from amongst all best solutions;
3. all best solutions amongst all solutions whose semiring value is greater than a given bound α ;
4. one best solution from amongst all solutions whose semiring value is greater than a given bound α ;
5. all best solutions, given that we know their semiring level α ;
6. one best solution, given that we know their semiring level α ;

Mapping the solution process for crisp CSPs into the above classification results in the two last categories (using *true* as the threshold level α). In Section 4 we will study how removing symmetries in each of the above categories of problems affects search.

2.3 Symmetry in Soft CSPs

Using an approach similar to [Benhamou, 1994], in [Bistarelli et al., 2003b] we defined two notions of *semantic symmetry*: *symmetry for satisfiability* and *symmetry for all solutions*.

Informally, two domain values a and b are *symmetrical for satisfiability* if whenever the assignment $v := a$ ($v := b$) leads to a solution with semiring value α , we can also obtain a solution with the same value α using the assignment $v := b$ ($v := a$), i.e. we say that b and a are *symmetrical for satisfiability* ($a \approx b$) if and only if

$$\forall \alpha, \exists \eta, \eta' : \bigotimes C\eta[v := a] = \alpha \iff \bigotimes C\eta'[v := b] = \alpha$$

When we want to indicate that a and b are symmetrical for satisfiability via η and η' we will write $a \approx^{\eta, \eta'} b$.

Informally, two domain values a and b are instead *symmetrical for all solutions (w.r.t. the constraints C)* if whenever we have the assignment $\eta[v := a]$ with semiring value α , there is also an assignment $\eta'[v := b]$ with the same semiring value, where $\eta'[v := b] = \phi(\eta[v := a])$ (for some bijective mapping ϕ), and vice versa. Therefore, we say that b and a are *symmetrical (w.r.t. the constraints C) for all solutions ($a \simeq b$)* if and only if

$$\begin{aligned} & \exists \phi, \eta', \eta'': \\ & \forall \eta : \phi(\eta[v := a]) = \eta'[v := b] \\ & \quad \wedge \phi(\eta[v := b]) = \eta''[v := a] \\ & \quad \wedge \bigotimes C\eta[v := a] = \bigotimes C\phi(\eta[v := a]) \\ & \quad \wedge \bigotimes C\eta[v := b] = \bigotimes C\phi(\eta[v := b]). \end{aligned}$$

When we want to indicate that a and b are symmetrical for all solutions via the specific symmetry function ϕ we will write $a \simeq^\phi b$.

Clearly symmetry for all solutions implies symmetry for satisfiability [Bistarelli *et al.*, 2003b].

Threshold Symmetries

Symmetries in SCSPs are rarer than in classical CSPs. For this reason (using a notion of threshold similar to that defined by Bistarelli *et al.* [Bistarelli *et al.*, 2003a]) in [Bistarelli *et al.*, 2003b] we defined an approximate notion of symmetry. We say that b and a are α -symmetrical for satisfiability ($a \approx_\alpha b$) if and only if

$$\forall \bar{\alpha} \geq \alpha, \exists \eta, \eta' : \bigotimes C\eta[v := a] = \bar{\alpha} \iff \bigotimes C\eta'[v := b] = \bar{\alpha}$$

When we want to indicate that a and b are α -symmetrical for satisfiability via η and η' we will write $a \approx_\alpha^{\eta, \eta'} b$. Informally, two domain values a and b are α -symmetrical for satisfiability if whenever the assignment $v := a$ ($v := b$) leads to a solution with value $\bar{\alpha} \geq \alpha$, then, there is also a way to obtain a solution with the same value $\bar{\alpha}$ using the assignment $v := b$ ($v := a$).

We say that b and a are α -symmetrical for all solutions ($a \simeq_\alpha b$) if and only if

$$\begin{aligned} & \exists \phi, \eta', \eta'', \bar{\alpha}, \bar{\alpha}' \geq \alpha : \forall \eta : \\ & \phi(\eta[v := a]) = \eta'[v := b] \\ & \quad \wedge \phi(\eta[v := b]) = \eta''[v := a] \\ & \quad \wedge \bigotimes C\eta[v := a] = \bar{\alpha} \\ & \quad \wedge \bigotimes C\phi(\eta[v := a]) = \bar{\alpha} \\ & \quad \wedge \bigotimes C\eta[v := b] = \bar{\alpha}' \\ & \quad \wedge \bigotimes C\phi(\eta[v := b]) = \bar{\alpha}'. \end{aligned}$$

When we want to indicate that a and b are α -symmetrical for all solutions via the mapping ϕ we will write $a \simeq_\alpha^\phi b$. Informally, two domain values a and b are α -symmetrical for

all solutions if whenever the assignment $\eta[v := a]$ leads to a solution whose semiring value is $\alpha' \geq \alpha$, there is also a solution $\eta'[v := b]$ with the same semiring value, where $\eta'[v := b] = \phi(\eta[v := a])$ (for some bijective mapping ϕ), and vice versa.

We proved [Bistarelli *et al.*, 2003b] that the number of symmetries increases when we increase the threshold level. By using that result we also easily have¹:

Corollary 1 *Given two domain elements a and b and a threshold α , then,*

- *if $a \approx b$, then $a \approx_\alpha b$;*
- *if $a \simeq b$, then $a \simeq_\alpha b$.*

3 Extending Symmetry for Soft CSPs

All of the previous definitions of symmetry in terms of a single variable can be easily extended to assignments of more than a single variable. This notion will be useful in Section 4 where we will show that symmetry breaking is indeed very useful in solving soft CSPs.

Note that the semiring projection operator can also be used to compute the semiring value associated with partial instantiations. If we have a partial instantiation $\eta' : V' \rightarrow D$, and a constraint c s.t. $V' \subset \text{supp}(c)$, the semiring value associated with cn' is computed by first projecting c over the variables V' and then computing the semiring value of the resultant constraint. That is $cn' = c \Downarrow_{V'} \eta'^2$.

Definition 1 (Symmetry for Partial Instantiations)

Consider two partial assignments η_1 and η_2 over the same set of variables $\bar{V} \subseteq V$ and the set of constraints C :

- *we say that η_1 and η_2 are symmetrical for satisfiability ($\eta_1 \approx \eta_2$) if and only if*

$$\forall \alpha, \exists \eta, \eta' : \bigotimes C\eta[\eta_1] = \alpha \iff \bigotimes C\eta'[\eta_2] = \alpha$$

- *we say that η_1 and η_2 are symmetrical for all solutions (w.r.t. the constraints C) ($\eta_1 \simeq \eta_2$) if and only if*

$$\begin{aligned} & \exists \phi, \eta', \eta'': \\ & \forall \eta : \phi(\eta[\eta_1]) = \eta'[\eta_2] \\ & \quad \wedge \phi(\eta[\eta_2]) = \eta''[\eta_1] \\ & \quad \wedge \bigotimes C\eta[\eta_1] = \bigotimes C\phi(\eta[\eta_1]) \\ & \quad \wedge \bigotimes C\eta[\eta_2] = \bigotimes C\phi(\eta[\eta_1]). \end{aligned}$$

Since finding the mapping ϕ is one of the most important and difficult steps in order to exploit symmetry, it could be useful to give some equivalent (sometimes easier) conditions to check.

¹Proofs for all theorems can be found in [Bistarelli and O'Sullivan, 2004].

²The definition of how to compute semiring values for partial instantiations was not defined in [Bistarelli *et al.*, 1997; 2002]. However, this is one of the most natural ways to compute them.

Proposition 1 *Symmetry for all solutions ($\eta_1 \simeq \eta_2$) (equation 1) holds iff equation 2 holds iff equation 3 holds:*

$$\exists \phi_1, \eta', \eta'': \quad (1)$$

$$\begin{aligned} \forall \eta : \phi_1(\eta[\eta_1]) &= \eta'[\eta_2] \\ \wedge \phi_1(\eta[\eta_1]) &= \eta''[\eta_2] \\ \wedge \bigotimes C\eta[\eta_1] &= \bigotimes C\phi_1(\eta[\eta_1]) \\ \wedge \bigotimes C\eta[\eta_2] &= \bigotimes C\phi_1(\eta[\eta_2]); \end{aligned}$$

$$\begin{aligned} \exists \phi_2, \forall \eta : \bigotimes C\eta[\eta_1] &= \bigotimes C\phi_2(\eta[\eta_1])[\eta_2] \wedge \quad (2) \\ \bigotimes C\eta[\eta_2] &= \bigotimes C\phi_2(\eta[\eta_2])[\eta_1]; \end{aligned}$$

$$\begin{aligned} \exists \phi_3, \forall \eta : \bigotimes C\eta[\eta_1] &= \bigotimes C\phi_3(\eta)[\eta_2] \wedge \quad (3) \\ \bigotimes C\eta[\eta_2] &= \bigotimes C\phi_3(\eta)[\eta_1]; \end{aligned}$$

We can prove some interesting properties when adding/removing assignments to two partial instantiations η_1 and η_2 .

Theorem 1 (Extended symmetry for satisfiability) *Given two partial instantiations η_1, η_2 over the same set of variables $\bar{V} \subseteq V$, and a variable $v \in V - \bar{V}$, we have*

$$\eta_1 \cup \{v := a\} \approx^{\eta, \eta'} \eta_2 \cup \{v := b\} \iff \eta_1 \approx^{\eta[v:=a], \eta'[v:=b]} \eta_2.$$

Theorem 2 (Extended symmetry for all solutions) *Given two partial instantiations η_1, η_2 over the same set of variables $\bar{V} \subseteq V$, and a variable $v \in V - \bar{V}$. If ϕ is decomposable³ we have*

$$\eta_1 \simeq^\phi \eta_2 \implies \eta_1 \cup \{v := a\} \simeq^\phi \eta_2 \cup \phi(\{v := a\}).$$

Notice that threshold symmetries can also be defined over partial instantiations in a manner similar to Definition 1. Relating thresholds and symmetries over partial instantiations leads to some interesting theorems that will be used to prune the search space in Section 4.

Corollary 2 (Extended α -symmetries for satisfiability)

Given two partial instantiations η_1, η_2 over the same set of variables $\bar{V} \subseteq V$, and a variable $v \in V - \bar{V}$, we have

$$\begin{aligned} \{\eta_1 \cup \{v := a\}\} \approx_\alpha^{\eta, \eta'} \{\eta_2 \cup \{v := b\}\} \\ \iff \eta_1 \approx_\alpha^{\eta[v:=a], \eta'[v:=b]} \eta_2. \end{aligned}$$

and

$$\eta_1 \simeq_\alpha^\phi \eta_2 \implies \eta_1 \cup \{v := a\} \simeq_\alpha^\phi \eta_2 \cup \phi(\{v := a\}).$$

Theorem 3 (Symmetries and SCSP solution levels (1))

Given a constraints problem over the constraint set C and given two partial instantiations η_1, η_2 over the same set of variables $\bar{V} \subseteq V$ s.t. $C\eta_1 < \alpha$, and $C\eta_2 < \alpha$ then we have $\eta_1 \approx_\alpha \eta_2$ and $\eta_1 \simeq_\alpha \eta_2$.

³A mapping $\phi_v : D \rightarrow D$ is decomposable if any $\phi(V \rightarrow D) \rightarrow (V \rightarrow D)$ is defined as the composition of several ϕ_{v_i} , with $v_1, \dots, v_i, \dots, v_n \in V$ s.t. $\phi(v_1 := a_1, \dots, v_n := a_n) = \phi_{v_1}(a_1), \dots, \phi_{v_n}(a_n)$.

Corollary 3 *Given a constraints problem over the constraint set C . If for all η we have $\bigotimes \bar{C}\eta < \alpha$, then for any η_1, η_2 over the same set of variables $\bar{V} \subseteq V$ we have $\eta_1 \approx_\alpha \eta_2$ and $\eta_1 \simeq_\alpha \eta_2$.*

Theorem 4 (Symmetries and SCSP solutions levels (2))
Given two partial instantiations η_1, η_2 over the same set of variables $\bar{V} \subseteq V$, and a variable $v \in V - \bar{V}$, we have

$$\forall a_i, (\eta_1 \cup v := a_i) \simeq_\alpha (\eta_2 \cup \phi(v := a_i)) \implies \eta_1 \simeq_\alpha \eta_2$$

The previous theorem proves to be very important in the next section where it will be used to cut computation branches during search.

4 Exploiting Symmetry Breaking in Branch&Bound

In this section we show how symmetry breaking can improve the amount of pruning performed by a classical Branch&Bound algorithm. In particular, we will define Soft-SBDD extending the classical definition [Fahle *et al.*, 2001].

4.1 Soft-SBDD

SBDD uses the notion of dominance amongst partial instantiations of variables. Using our notation we can say that a partial instantiation η_1 is dominated by a partial instantiation η_2 if $\eta_1 \subseteq \eta_2$ (that is if η_2 extends the instantiation η_1). When given a symmetry ϕ , the dominance relation can be extended, by noting that if η_1 is dominated by η_2 ($\eta_1 \subseteq \eta_2$) and η_3 is symmetric with η_2 ($\eta_3 = \phi(\eta_2)$), then η_1 is dominated under the symmetry ϕ by η_3 .

More precisely, we can define two partial orders on partial instantiations, based on the notions of symmetry for satisfiability (\approx) and symmetry for all solutions (\simeq) defined in Section 3.

Definition 2 (Dominance under symmetry) *Consider three partial assignments η_1, η_2, η_3 :*

- we say that η_1 is dominated under symmetry for satisfiability by η_3 ($\eta_1 \lesssim \eta_3$) if and only if $\eta_1 \subseteq \eta_2$ and $\eta_2 \approx \eta_3$;
- we say that η_1 is dominated under symmetry for all solutions by η_3 ($\eta_1 \lesssim \eta_3$) if and only if $\eta_1 \subseteq \eta_2$ and $\eta_2 \simeq \eta_3$;
- we say that η_1 is α -dominated under symmetry for satisfiability by η_3 ($\eta_1 \lesssim_\alpha \eta_3$) if and only if $\eta_1 \subseteq \eta_2$ and $\eta_2 \approx_\alpha \eta_3$;
- we say that η_1 is α -dominated under symmetry for all solutions by η_3 ($\eta_1 \lesssim_\alpha \eta_3$) if and only if $\eta_1 \subseteq \eta_2$ and $\eta_2 \simeq_\alpha \eta_3$;

We can easily show that \lesssim , \lesssim_α , \lesssim_{alpha} and \lesssim_{alpha} are partial orders.

Theorem 5 \lesssim , \lesssim_α , \lesssim_{alpha} and \lesssim_{alpha} are partial orders.

The nodes in the search tree can be represented as partial instantiations. At each step SBDD checks if there exists an already visited node η' s.t. $\eta \lesssim \eta'$, i.e. we use the threshold definition of symmetry. If this is the case the algorithm backtracks and prunes all the branches rooted at η .

If we want to use SBDD to solve Soft CSPs, we have to modify it in order to be able to perform pruning with respect a threshold α . We will refer to this modified version of SBDD as Soft-SBDD.

Using Soft-SBDD, a node of the search space can be represented by partial instantiations plus the semiring value for each instantiation. As in SBDD, Soft-SBDD checks at each step if there exists an already visited node η' s.t. $\eta \lesssim \eta'$. If this is the case the algorithm backtracks and prunes all the branches rooted at η .

Moreover, when a complete assignment η' is found, with associated semiring value α , this is used as a bound. For each new visited node η , we check if $\eta \lesssim_\alpha \eta'$. When the result of this check is positive, the algorithm can backtrack and prune all the branches rooted at η .

We now consider an example to show that symmetry breaking can prune branches of the search tree that are not pruned by a Branch&Bound algorithm.

Example 1 Consider an SCSP over the semiring $S_{weight} = \langle \mathcal{R}, min, +, +\infty, 0 \rangle$ with variables $V = \{x, y, z\}$, each with the same domain $D = \{a, b, c\}$, there are the following unary and binary constraints, $C = \{c_x, c_y, c_z, c_{yz}\}$, defined as follows:

$$\begin{aligned} c_x &= \{\langle a \rangle \rightarrow 4, \langle b \rangle \rightarrow 5, \langle c \rangle \rightarrow 4\}; \\ c_y &= \{\langle a \rangle \rightarrow 3, \langle b \rangle \rightarrow 3, \langle c \rangle \rightarrow 10\}; \\ c_z &= \{\langle a \rangle \rightarrow 6, \langle b \rangle \rightarrow 8, \langle c \rangle \rightarrow 5\}; \\ c_{yz} &= \{\langle a, a \rangle \rightarrow 0, \langle a, b \rangle \rightarrow 0, \langle a, c \rangle \rightarrow 0, \\ &\quad \langle b, a \rangle \rightarrow \infty, \langle b, b \rangle \rightarrow \infty, \langle b, c \rangle \rightarrow 0, \\ &\quad \langle c, a \rangle \rightarrow 0, \langle c, b \rangle \rightarrow 0, \langle c, c \rangle \rightarrow 0\} \end{aligned}$$

We assume that there are other constraints in the problem giving the following symmetry ϕ s.t. $\{x := a, y := a\} \simeq_{12} \{x := a, y := b\}$. Let's also assume that we will use a variable ordering heuristic that considers the variables in the order x , y , and z . Domain elements are selected in the order a , b , and c . Assume also that the estimate of the cost of a partial instantiation is the minimum of the sum of the complete assignment using the projection operator defined at the beginning of Section 3. We are seeking all best solutions.

Let's consider the situation once we have reached the state represented in Figure 2. In the figure, the grey nodes represent fully explored branches, while the white ones are nodes yet to be fully explored. The search at this point has found three complete instantiations $\{x := a, y := a, z := a\}$, $\{x := a, y := a, z := b\}$ and $\{x := a, y := a, z := c\}$ with associated semiring levels 13, 15 and 12, respectively. Therefore, at this stage, 12 is the current bound used by the Branch&Bound algorithm to prune branches of the search tree.

Suppose we now backtrack and make the instantiation $y := b$ leading to the state represented by the partial assignment $\{x := a, y := b\}$. The approximation computed for this instantiation by the Branch&Bound algorithm for this node is 12 (computed by summing the cost for the two instantiated constraints $c_x = a \rightarrow 4$ and $c_y = b \rightarrow 3$ and the best possible value for z which has cost 5). Since the approximation is

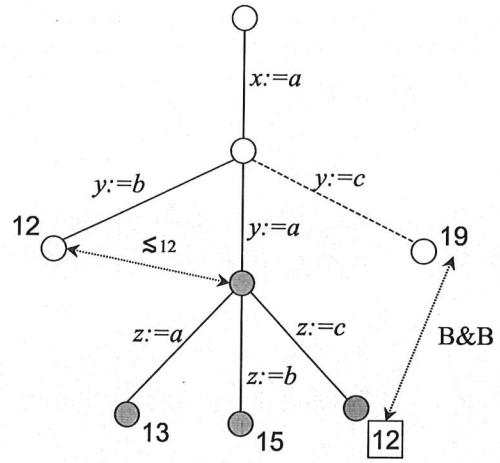


Figure 2: Search tree after some steps.

not worse than the actual bound (remember we are minimizing and trying to find all best solutions, we cannot prune this branch at this stage).

However, if we use SBDD, we can check if the current node is dominated by some of the nodes that have already been expanded (that is we have to check if there exists a (partial) instantiation η s.t. $\{x := a, y := b\} \lesssim \eta$). This is what classical SBDD does.

In our case, since we have a bound (12) we can use it in order to perform more pruning. To do that we have to check if there exists a (partial) instantiation η s.t. $\{x := a, y := b\} \lesssim_{12} \eta$. By hypothesis we have a symmetry ϕ with $\{x := a, y := b\} \lesssim_{12} \{x := a, y := a\}$. Since the node $\{x := a, y := b\}$ is completely explored, we can prune node $\{x := a, y := b\}$ because we are sure that all the nodes in this branch represent either a solution with semiring level worse than 12 or a solution with semiring level 12, symmetric to $\{x := a, y := a, z := c\}$ (and easily computed from $\phi(\{x := a, y := a, z := c\})$).

At the next step, when instantiating $y := c$ we move to the state $\{x := a, y := c\}$. Since the approximation of the cost of this branch is 19 and since $19 > 12$, we can prune this node using Branch&Bound. \triangle

Therefore, this example shows how the pruning of Branch&Bound can be improved by using SBDD. Notice that SBDD and Branch&Bound have two different partial orders for pruning. Soft-SBDD uses \lesssim , while Branch&Bound uses the \sqsubseteq order induced by the semiring order \leq . Both partial orders rely on the fact that instantiating more variables leads to a solution which is no better from the perspective of consistency; for SBDD this is due to its definition, and for Branch&Bound from the fact that constraints are monotonic (the more variables we instantiate, the more constraints are defined, and the worse will be their combination).

Using Soft-SBDD, we combine together Branch&Bound and symmetry breaking. We will see in the following how using \lesssim_α will prune more than classic Branch&Bound.

However, even if we have a perfect heuristic for Branch&Bound, i.e. a heuristic that would compute the true best semiring value for a given search tree node, does it perform better than Soft-SBDD? We can prove the following:

Theorem 6 *Branch&Bound with a “perfect” heuristic cannot prune all of the branches pruned by a “perfect” Soft-SBDD.*

We can also prove something stronger. In Figure 2 the branch $\{x := a, y := c\}$ is pruned by Branch&Bound because the current estimated cost was worse than the bound. However, can that node be removed by symmetry?

In general the answer to this question is no. Finding symmetry functions in a problem is fact one of the main drawbacks of applying symmetry breaking. We usually need to have a deep understanding of the problem in order to identify all its symmetries. However, if all the symmetries of a problem were known, Soft-SBDD could perform better than classical Branch&Bound.

Theorem 7 *If all the α -symmetries of a problem are known, Soft-SBDD can prune more than a classical Branch&Bound.*

Example 2 As an example consider Figure 3. Node $\{x := a, y := c\}$ has been extended to $\{x := a, y := c, z := a\}$, $\{x := a, y := c, z := b\}$ and $\{x := a, y := c, z := c\}$, with the associated cost of 20, 22 and 19. By definition of α -symmetry we have: $\{x := a, y := a, z := c\} \simeq_{12} \{x := a, y := c, z := a\}$, $\{x := a, y := a, z := c\} \simeq_{12} \{x := a, y := c, z := b\}$, $\{x := a, y := a, z := c\} \simeq_{12} \{x := a, y := c, z := c\}$ ⁴. By using the results of Theorem 4, we can also say that $\{x := a, y := a\} \simeq_{12} \{x := a, y := c\}$, so, in the case where we know all the symmetries in a problem, we can prune at node $\{x := a, y := c\}$ using Soft-SBDD, because $\{x := a, y := c\} \lesssim_{12} \{x := a, y := a\}$. \triangle

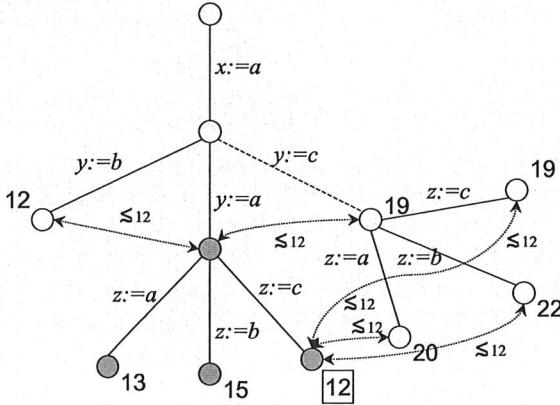


Figure 3: Soft-SBDD can perform better than B&B.

⁴In fact, the definition of α -symmetry takes into account the semiring level of the solution. If the solution is worse than 12, they are by definition $_{12}$ -symmetric.

4.2 Applying Soft Symmetry Breaking

Symmetry breaking is used to remove symmetrical instantiations in order to reduce the search space. Many of the methodologies described in Section 2.1 remove solutions symmetric to those already found (we will call this *symmetry breaking on success*). Others, instead, remove branches of the search tree corresponding to non-solutions already found (we will call this *symmetry breaking on failure*).

In the following we will consider each of the six optimization problems in SCSPs (see end of Section 2.2) and we will indicate how symmetry breaking can be used to reduce the size of the search space in each case. We will also highlight when the amount of pruning performed by Soft-SBDD (that can be seen as symmetry breaking plus Branch&Bound) improves that of the classic Branch&Bound algorithm, that can be used to solve an SCSP.

Proposition 2 *When looking for one best solution, given that we know its semiring level α , soft symmetry breaking performs more pruning than classic Branch&Bound due to symmetry breaking on failure only.*

Essentially, if we know the semiring level of the best solution, once we have found it we are done. However, as we search we find solutions below the desired threshold. The symmetric equivalents of these solutions can be pruned using symmetry breaking, thus reducing the amount of redundant work that Branch&Bound has to perform.

Proposition 3 *When looking for one best solution from amongst all solutions greater than a given bound α , soft symmetry breaking performs more pruning than classic Branch&Bound due to both symmetry breaking on success and failure.*

Since the best semiring level of the solution is unknown we have to explore the entire search space to decide if the best solution that has been found to date is the best one. Symmetry breaking on success can be used to exclude from the search space equivalent solutions that we do not want to collect. However, such pruning will usually be quite weak in comparison to the pruning that can be performed by symmetry on failure.

Proposition 4 *When looking for one solution among all best solutions, soft symmetry breaking performs more pruning than classic Branch&Bound due to both symmetry breaking on success and failure.*

In this case an approximation of the semiring level of the best solution is not known. This implies that it is not possible to perform initial symmetry breaking on failure. However, as soon as we have found a solution with semiring level α we can use this threshold to perform symmetry breaking on success and failure.

Proposition 5 *When looking for all best solutions, either with or without specified bounds on the semiring level, soft symmetry breaking performs more pruning than classic Branch&Bound due to both symmetry breaking on success and failure.*

If we wish to find all the best solutions, symmetry breaking on success becomes more useful. It is important to notice that in this, and all the previous cases, symmetry breaking on failure is much more useful whenever the heuristic used in Branch&Bound is not perfect. In the theoretical case of dealing with a perfect heuristic, symmetry breaking on failure is not useful, but, when all the solutions are needed, symmetry breaking on success remains useful.

5 Conclusions and Future Work

While symmetry breaking has been studied widely in the context of crisp constraint satisfaction, it has received very little attention in the context of soft constraints. We make contributions to this topic in this paper.

One of the most powerful techniques used in symmetry breaking is based on dominance detection. In this paper we have extended an existing approach to symmetry breaking for soft constraints in order to exploit dominance amongst partial instantiations. This provides a basis for a generalization of Symmetry Breaking via Dominance Detection for soft constraint satisfaction problems called Soft-SBDD, which we have theoretically shown to be beneficial when solving soft CSPs.

Soft-SBDD provides a basis for exploiting symmetry amongst preferences, which has applications in a number of fields such as preference-based configuration and e-commerce. As part of our future work we plan to implement Soft-SBDD in the context of a branch & bound solver, following the approach of [Gent *et al.*, 2002; 2003] in order to fully evaluate it on real-world problems.

References

- [Backofen and Will, 1999] R. Backofen and S. Will. Excluding symmetries in concurrent constraint programming. In *Proceedings of CP-99*, LNCS 1520, pages 72–86, 1999.
- [Benhamou, 1994] B. Benhamou. Study of symmetry in constraint satisfaction problems. In *Proc. CP'94*, 1994.
- [Bistarelli and O'Sullivan, 2004] S. Bistarelli and B. O'Sullivan. Combining Branch&Bound and SBDD to solve Soft CSPs, May 2004. Available from:
<http://www.cs.ucc.ie/~osullb/pubs/ercim2004proofs.pdf>.
- [Bistarelli *et al.*, 1995] S. Bistarelli, U. Montanari, and F. Rossi. Constraint Solving over Semirings. In *Proc. IJCAI95*, pages 624–630, San Francisco, CA, USA, 1995. Morgan Kaufman.
- [Bistarelli *et al.*, 1997] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based Constraint Solving and Optimization. *Journal of the ACM*, 44(2):201–236, Mar 1997.
- [Bistarelli *et al.*, 1999] S. Bistarelli, H. Fargier, U. Montanari, F. Rossi, T. Schiex, and G. Verfaillie. Semiring-based CSPs and Valued CSPs: Frameworks, properties, and comparison. *Constraints*, 4(3), 1999.
- [Bistarelli *et al.*, 2002] S. Bistarelli, U. Montanari, and F. Rossi. Soft concurrent constraint programming. In *Proc. ESOP, April 6 - 14, 2002, Grenoble, France*, LNCS, pages 53–67. Springer-Verlag, 2002.
- [Bistarelli *et al.*, 2003a] S. Bistarelli, B. Faltings, and N. Neagu. A definition of interchangeability for soft csp. In *Recent Advances in Constraints*, LNAI 2627. Springer, 2003.
- [Bistarelli *et al.*, 2003b] S. Bistarelli, J. Kelleher, and B. O'Sullivan. Symmetry breaking in soft csp. In *Proceedings of AI-2003, the Twenty-third SGAI International Conference on Knowledge-Based Systems and Applied Artificial Intelligence*, 2003.
- [Bistarelli, 2001] S. Bistarelli. *Soft Constraint Solving and programming: a general framework*. PhD thesis, Dipartimento di Informatica, Università di Pisa, Italy, mar 2001. TD-2/01.
- [Bistarelli, 2004] S. Bistarelli. *Semirings for Soft Constraint Solving and Programming*, volume 2962 of *LNCS*. Springer, 2004.
- [Bowen *et al.*, 1992] J. Bowen, R. Lai, and D. Bahler. Lexical imprecision and fuzzy constraint networks. In *Proceedings of AAAI-92*, pages 616–621, July 1992.
- [Brown *et al.*, 1988] C.A. Brown, L. Finkelstein, and P.W. Purdon Jr. Backtrack searching in the presence of symmetry. In T. Mora, editor, *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, volume 357 of *LNCS*, pages 99–110. Springer-Verlag, 1988.
- [Crawford *et al.*, 1996] J. Crawford, G. Luks, M. Ginsberg, and A. Roy. Symmetry breaking predicates for search problems. In *Proc. KR-96*, pages 148–159, 1996.
- [Crawford, 1992] J. Crawford. A theoretical analysis of reasoning by symmetry in first-order logic. In *Proceedings of the AAAI-92 Workshop on Tractable Reasoning*, 1992.
- [Domshlak *et al.*, 2003] C. Domshlak, F. Rossi, B. Venable, and T. Walsh. Reasoning about soft constraints and conditional preferences: complexity results and approximation techniques. In *Proceedings of IJCAI-2003*, August 2003.
- [Fahle *et al.*, 2001] T. Fahle, S. Schamberger, and M. Sellmann. Symmetry breaking. In *Proceedings of CP-01*, LNCS 2239, pages 93–107, 2001.
- [Flener *et al.*, 2002] P. Flener, A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Breaking row and column symmetries in matrix models. In *Proceedings of CP-02*, LNCS 2470, pages 462–476, 2002.
- [Focacci and Milano, 2001] F. Focacci and M. Milano. Global cut framework for removing symmetries. In *Proceedings of CP-01*, LNCS 2239, pages 75–92, 2001.
- [Focacci and Shaw, 2002] Filippo Focacci and Paul Shaw. Pruning sub-optimal search branches using local search. In Narendra Jussien and François Laburthe, editors, *Proceedings of CP-AI-OR'02*, pages 181–189, Le Croisic, France, March, 25–27 2002.
- [Freuder, 1991] E.C. Freuder. Eliminating interchangeable values in constraint satisfaction problems. In *Proceedings of the AAAI*, pages 227–233, 1991.

- [Gent and Smith, 2000] I.P. Gent and B.M. Smith. Symmetry breaking in constraint programming. In W. Horn, editor, *Proceedings of ECAI-2000*, pages 599–603. IOS Press, 2000.
- [Gent *et al.*, 2002] I.P. Gent, W. Harvey, and T. Kelsey. Groups and constraints: Symmetry breaking during search. In *Proceedings of CP-02*, LNCS 2470, pages 415–430, 2002.
- [Gent *et al.*, 2003] I.P. Gent, W. Harvey, T. Kelsey, and S. Linton. Generic SBDD using computational group theory. In *Proc. CP2003*, volume 2833 of *LNCS*, pages 333–347. Springer, 2003.
- [Junker, 2001] U. Junker. Preference programming for configuration. In *Proceedings of the 4th International Workshop on Configuration (IJCAI-01)*, pages 50–56, August 2001.
- [McDonald and Smith, 2002] I. McDonald and B. Smith. Partial symmetry breaking. In *Proceedings of CP-02*, LNCS 2470, pages 431–445, 2002.
- [Meseguer and Torras, 2001] P. Meseguer and C. Torras. Exploiting symmetries within constraint satisfaction search. *Artificial Intelligence*, 129(1–2):133–163, 2001.
- [Puget, 1993] J.-F. Puget. On the satisfiability of symmetrical constrained satisfaction problems. In *Proceedings of ISMIS-93*, LNAI 689, pages 350–361, 1993.
- [Puget, 2002] J.-F. Puget. Symmetry breaking revisited. In *Proceedings of CP-02*, LNCS 2470, pages 446–461, 2002.
- [Schiex, 1992] T. Schiex. Possibilistic constraint satisfaction problems, or “how to handle soft constraints?”. In *Proc. 8th Conf. of Uncertainty in AI*, pages 269–275, 1992.

Comparing the use of symmetry in constraint processing and model checking

Alastair Donaldson, Alice Miller, Muffy Calder

Department of Computing Science

University of Glasgow, Scotland

{ally,alice,muffy}@dcs.gla.ac.uk

Abstract

Both *model checking* and *constraint processing* involve the searching of graphs: in model checking to establish the truth of a temporal logic formula; in constraint processing to determine whether or not solutions to a problem exist which satisfy a set of constraints. In both fields, the presence of symmetry in the model or problem can result in redundant search over equivalent areas of the graph or search space. Recently there has been much interest in *symmetry reduction* in model checking, and *symmetry breaking* in constraint satisfaction problems (CSPs). The *n*-queens problem is a CSP to which symmetry breaking has been applied. To illustrate the approaches to exploiting symmetry in both model checking and CSPs we show how the *n*-queens problem can be solved using model checking, and how symmetry reduction can be used to make larger instances of the problem tractable. Through this example we highlight the similarities and differences between the approaches.

1 Introduction

Searching for *all* solutions of a constraint satisfaction problem (CSP) can take a prohibitively long time due to symmetry inherent in the problem [14]. Most of the search effort is dedicated to exploring variable assignments which are symmetrically equivalent to previous assignments, and are thus redundant. Similarly in model checking, the state space of a model may be intractably large, but consist of many symmetrically equivalent components which are indistinguishable in terms of the global behaviour of the system being modelled [7].

Recently there has been much research into *symmetry breaking* in constraint processing—adding constraints to a problem either before or during search to avoid the exploration of symmetrically equivalent assignments [12; 13; 14]. The goal of this approach is to quickly obtain only a subset of all solutions to the problem; the set of *all* solutions is obtained from this subset by applying symmetries. Similarly in model checking there has been significant interest in *symmetry reduction* [7; 10; 19]. If symmetries of a model can be identified then they can be exploited during model checking

to construct a *quotient model*. The quotient model is generally smaller than the original model, but they are behaviourally equivalent in the sense that they satisfy the same set of (symmetrically invariant) properties. Thus model checking can be performed over the smaller quotient model.

The similarity between model checking and constraint processing has been investigated, and constraints technology has been applied to model checking problems [9; 11]. Model checking has been used to solve the *rehearsal problem* [16], generally accepted as a constraint satisfaction problem [21]. In this paper we illustrate the relationship between symmetry breaking in constraint programming and symmetry reduction in model checking, using the *n*-queens problem as a case study. We do not suggest that model checking is more suitable than constraints technology for solving this problem—we use the example as a way to demonstrate the theory of symmetry in model checking to the constraint programming community.

1.1 Overview of results

We present the *n*-queens problem using the Promela modelling language, and show how the SPIN model checker [17] can be used to find all solutions. We then show how symmetry reduction can be applied to ensure that only symmetrically distinct solutions are found. Through this example we show that model checking over a quotient Kripke structure is similar both to the method of symmetry breaking during search [13], and to the approach of finding the smallest solution in each equivalence class under an appropriate ordering [4] in constraint processing.

For all verification runs we used a PC with a 2.4GHz Intel Xeon processor, 3Gb of available main memory, running Linux (2.4.18), with SPIN version 4.0.7.

2 Preliminaries

We give a brief description of model checking in section 2.1, and of the SPIN model checker and the Promela modelling language in section 2.2. For a thorough introduction to model checking see e.g. [6], and for the definitive reference on SPIN and Promela see [17].

2.1 Model checking, Kripke structures and temporal logic

Model checking is a technique whereby temporal logic properties of a system can be checked by building an abstract

model of the system and checking whether the model satisfies the properties. The model is constructed using a specification language, and checked using an automatic model checker. Failure of the *model* to satisfy a property of the system indicates either that the model does not accurately reflect the behaviour of the system, that the property has been inaccurately specified in temporal logic, or that there is an error (bug) in the original system. Examination of counter-examples provided by the model checker enable the user to either refine the model, refine the property, or, more importantly, to debug the original system.

To reason about a model, we refer to its underlying Kripke structure, which is defined over a set of atomic propositions. Atomic propositions are logical statements representing the values of variables in the model.

Definition 1 Let AP be a set of atomic propositions. A Kripke structure \mathcal{M} over AP is a quadruple $\mathcal{M} = (S, R, L, s_0)$ where:

- S is a non-empty, finite set of states,
- $R \subseteq S \times S$ is a total transition relation, that is for each $s \in S$ there exists $t \in S$ such that $(s, t) \in R$,
- $L : S \rightarrow 2^{AP}$ is a mapping that labels each state in S with the set of atomic propositions true in that state,
- $s_0 \in S$ is an initial state.

In model checking, properties of models are usually specified in the temporal logic CTL*, or in one of its sub logics, LTL (linear temporal logic) or CTL (computation tree logic). In this paper we make use the SPIN model checker, which allows verification of LTL properties. The properties we are interested in use the operators \Box (always), and \Diamond (eventually). For further details on temporal logic see e.g. chapter 3 of [6].

2.2 Promela and SPIN

Promela is the specification language for the SPIN model checker [17]. Although similar in syntax to the C programming language, Promela includes constructs to specify nondeterminism and concurrency in a model, but not many of the advanced features of the C language such as memory management functions. SPIN is a bespoke model checker for the Promela language. Given a Promela model and a temporal logic formula, SPIN attempts to show that the model *does not* satisfy the formula by finding a counter example path through the model which violates the formula. In SPIN, LTL properties are converted into Büchi automata [23], expressed in the form of *never-claims*. A never-claim can be thought of as an additional process which makes a transition every time one of the other process in the model has made a transition. If no such transition is possible (if the current path cannot possibly lead to an error for example) the current path is blocked, and the search backtracks.

Usually when a single error is found, model checking terminates so that the error can be resolved. In the approach we use here, we require *all* errors to be reported before model checking terminates. Conveniently SPIN provides an option which supports this and provides a separate counterexample for each error.

3 Symmetry in CSPs

A symmetry α for a constraint problem C is a function that maps any assignment A which is a solution of C to a symmetric assignment $\alpha(A)$ which is also a solution of C [1]. The set of all symmetries for a problem C forms a group, which partitions the set of all possible variable assignments into equivalence classes. In any class, either every assignment is a solution, or none is.

According to Gent and Smith [14], the three aims for a symmetry-exclusion method are:

- to guarantee that we never allow search to find two symmetrically equivalent solutions,
- to respect heuristic choice as much as possible, and
- to allow arbitrary forms of symmetry.

One approach to symmetry breaking in CSPs involves adding symmetry breaking constraints to the CSP before search, converting it into a less symmetric (ideally asymmetric) problem [8; 20]. Another approach, known as *symmetry breaking during search* (SBDS) involves adding symmetry breaking constraints to the problem as the search proceeds, by detecting symmetries which remain unbroken when the search tries a fresh branch on backtracking [13; 14]. A third approach [4] defines an ordering on the set of assignments and finds only the smallest solution in each equivalence class under this ordering.

4 Symmetry in model checking

While there are several approaches to symmetry breaking in CSPs, the vast majority of work on symmetry reduction in model checking centres around one approach. This approach uses symmetries of a Kripke structure to automatically construct a *quotient* structure.

Let $\mathcal{M} = (S, R, L, s_0)$ be a Kripke structure. An *automorphism* of \mathcal{M} is a bijection $\alpha : S \rightarrow S$ which satisfies the following condition:

$$\forall s, t \in S, (s, t) \in R \Rightarrow (\alpha(s), \alpha(t)) \in R.$$

In a model of a concurrent system with many replicated processes, Kripke structure automorphisms may involve the permutation of process identifiers throughout all states of the model. On the other hand, a model may include a data structure which has geometrical symmetry. In this case Kripke structure automorphisms involve applying the geometrical symmetries throughout all states of the model. All Kripke structure automorphisms arising in this paper are geometrical.

The set of all automorphisms of the Kripke structure \mathcal{M} forms a group under composition of mappings. This group is denoted $Aut(\mathcal{M})$. A subgroup G of $Aut(\mathcal{M})$ induces an equivalence relation \equiv_G on the states of \mathcal{M} by the rule

$$s \equiv_G t \Leftrightarrow s = \alpha(t) \text{ for some } \alpha \in G.$$

The equivalence class under \equiv_G of a state $s \in S$, denoted $[s]$, is called the *orbit* of s under the action of G . The orbits can be used to construct a *quotient* Kripke structure \mathcal{M}_G as follows:

Definition 2 The quotient Kripke structure \mathcal{M}_G of \mathcal{M} with respect to G is a quadruple $\mathcal{M}_G = (S_G, R_G, L_G, [s_0])$ where:

- $S_G = \{[s] : s \in S\}$ (the set of orbits of S under the action of G),
- $R_G = \{([s], [t]) : (s, t) \in R\}$,
- $L_G([s]) = L(\text{rep}([s]))$ (where $\text{rep}([s])$ is a unique representative of $[s]$),
- $[s_0] \in S_G$ (the orbit of the initial state $s_0 \in S$).

In general \mathcal{M}_G is a smaller structure than \mathcal{M} , but \mathcal{M}_G and \mathcal{M} are equivalent in the sense that they satisfy the same set of logic properties which are *invariant* under the group G (that is, properties which are “symmetric” with respect to G). For a proof of the following theorem, together with a description of the temporal logic CTL*, and sub logics LTL and CTL, see [6].

Algorithm 1 Algorithm to construct a quotient Kripke structure

```

reached := {rep(s0)}
unexplored := {rep(s0)}
while unexplored ≠ ∅ do
    remove a state s from unexplored
    for all successor states q of s do
        if rep(q) is not in reached then
            append rep(q) to reached
            append rep(q) to unexplored
        end if
    end for
end while

```

Theorem 1 Let \mathcal{M} be a Kripke structure, G be a subgroup of $\text{Aut}(\mathcal{M})$ and f be a CTL* formula. If G is an invariance group for all the atomic propositions p occurring in f , then

$$\mathcal{M}, s \models f \Leftrightarrow \mathcal{M}_G, [s] \models f$$

where \mathcal{M}_G is the quotient structure corresponding to \mathcal{M} .

Thus by choosing a suitable symmetry group G , model checking can be performed over \mathcal{M}_G instead of \mathcal{M} , often resulting in considerable savings in memory and verification time [7].

Of course it must be possible to construct the structure \mathcal{M}_G without first constructing \mathcal{M} . Assuming that symmetries of the model are known in advance, this construction can be achieved using algorithm 1, adapted from [19].

5 The n -queens problem

The n -queens problem [2] is as follows: “Given an $n \times n$ board of squares, place n -queens on the board so that no queen can attack any other queen”. The problem is a generalisation of the 8-queens problem (the case where $n = 8$). A solution to the problem is a configuration of the board which satisfies the requirement that no two queens can attack one another. Figure 1 shows two solutions to the 4-queens problem. Figure 3 on the other hand shows two configurations of the 4-queens problem which are not solutions.

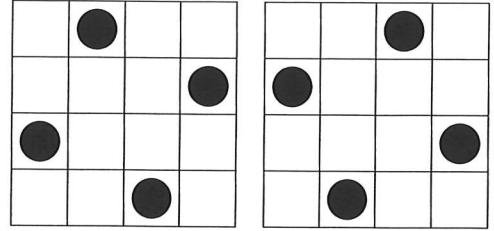


Figure 1: Symmetric solutions to the 4 queens problem.

5.1 Representing n -queens as a CSP

The standard formulation of n -queens as a constraint satisfaction problem uses n variables, q_1, q_2, \dots, q_n , where the position of the queen on row i of the board is represented by q_i . The domain of each of the variables q_1, q_2, \dots, q_n is $\{1, 2, \dots, n\}$. For all $i, j = 1, 2, \dots, n$, $i \neq j$, the following constraints ensure that no queen can attack any other:

$$q_i \neq q_j \quad (1)$$

$$q_i + i \neq q_j + j \quad (2)$$

$$q_i - i \neq q_j - j \quad (3)$$

Constraint 1 ensures that the positions of queens in different rows must be different, so that queens cannot attack one another by moving vertically on the board. The requirement that no two queens can appear on the same diagonal is specified by constraints 2 and 3. No constraint is required to ensure that only one queen appears on each row: formulating the problem using one variable per row means that this requirement is implicitly satisfied.

5.2 Representing n -queens in Promela

We follow an approach similar to the one presented in [16] for solving a constraint satisfaction problem using model checking. Recall from section 2.1 that model checking is useful for finding errors in models of systems. To find all solutions to a CSP using model checking we write a model of the problem, and assert that no solutions to the problem exist. The model checker then regards a solution as an error, and reports it.

The approach involves first writing a verification model which non-deterministically assigns values to each variable in the problem, so that the state space of the model includes all possible assignments to variables. A temporal logic formula asserting that the constraints of the problem are *not* satisfied in *any* state of the model must then be specified. Finally the model checker must be instructed to verify the formula, reporting all errors paths. All solutions to the problem will thus be reported as errors by the model checker.

Our model of the n -queens problem uses a byte array of length n , indexed from 0 to $n - 1$. Each entry of this array represents a row of the board, and is set to u (unassigned) if no queen has been placed on that row. Setting entry $j - 1$ to the value $i - 1$ implies that a queen has been placed at row j , column i ($1 \leq i, j \leq n$). Here is the Promela code for our model with $n = 4$:

```

#define u 255
byte q[4]=u;
bit result=0;

```

```

init {
  do
    :: result==1 -> skip
    :: else ->
      atomic { if
        :: q[0]==u ->
          if :: q[0]=0 :: q[0]=1 :: q[0]=2 :: q[0]=3 fi
        :: q[1]==u && q[0]!=u ->
          if :: q[1]=0 :: q[1]=1 :: q[1]=2 :: q[1]=3 fi;
          diagscheck(1,result);
          alldifferentcheck(1,result)
        :: q[2]==u && q[1]!=u && q[0]!=u ->
          if :: q[2]=0 :: q[2]=1 :: q[2]=2 :: q[2]=3 fi;
          diagscheck(2,result);
          alldifferentcheck(2,result)
        :: q[3]==u && q[2]!=u && q[1]!=u && q[0]!=u ->
          if :: q[3]=0 :: q[3]=1 :: q[3]=2 :: q[3]=3 fi;
          diagscheck(3,result);
          alldifferentcheck(3,result)
      fi }
  od
}

```

The keyword *atomic* ensures that the code enclosed in the following curly braces is treated as one step, so that a single state change occurs on execution of the enclosed block. The *do :: Seg od* construct causes the code segment *Seg* to be unconditionally repeated, and the construct *if Seg₁ :: ... :: Seg_m fi* allows one of the code segments *Seg_i* to be executed nondeterministically if it is executable ($1 \leq i \leq m$, $m > 0$).

To make the problem tractable our code uses two simple optimisations. The rows are assigned in order: $q[j]$ cannot be set until $q[k]$ has been set for all $k < j$ ($0 \leq j < n$). This is achieved by the boolean guards of the form

```
q[j]==u && q[j-1]!=u && ... && q[0]!=u
```

This optimisation ensures that there is exactly *one* path through the Kripke structure leading to each total variable assignment—without this optimisation there would be multiple paths to each solution as illustrated by figure 2, so the model checker would report the same solution many times. Note that the rows could be assigned in any fixed order.

The second optimisation is that on assigning each variable, a check is made to see whether or not the partial assignment satisfies the diagonal constraints and the column constraint. This optimisation makes use of two functions, *diagscheck(k,result)* and *alldifferentcheck(k,result)*. The function *diagscheck(k,result)* sets *result* to 1 if the assignment just made to row *k* breaks the diagonal constraints. This forces the search to backtrack if a partial assignment cannot be a solution. The function *alldifferentcheck(k,result)* works similarly for the column constraint. This optimisation dramatically reduces the state space of the model.

The following LTL formula is used to find solutions to the problem:

$$\neg(\diamond(\square(\text{finished} \wedge \text{alldifferent} \wedge \text{updiag} \wedge \text{downdiag})))$$

Here *alldifferent*, *updiag* and *downdiag* correspond to constraints 1, 2 and 3 in section 5.1 respectively. The proposition *finished* is used to ensure that only total assignments are con-

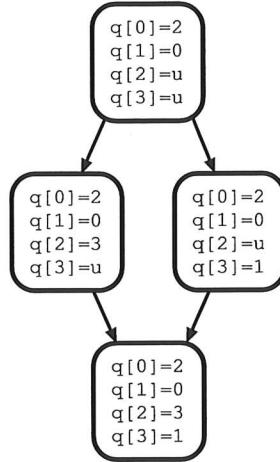


Figure 2: Two paths to the same solution. Our model ensures that only the left hand path is explored.

sidered as solutions.

$$\begin{aligned}
 \text{alldifferent} &= \bigwedge_{i \neq j} (q[i] \neq q[j]) \\
 \text{updiag} &= \bigwedge_{\substack{0 \leq i, j < n \\ i \neq j}} (q[i] + i \neq q[j] + j) \\
 \text{downdiag} &= \bigwedge_{\substack{0 \leq i, j < n \\ i \neq j}} (q[i] - i \neq q[j] - j) \\
 \text{finished} &= \bigwedge_{0 \leq i < n} (q[i] \neq u)
 \end{aligned}$$

Recall from section 5.1 that in modelling the problem using one variable for each row we eliminate the need for a constraint specifying that there should be no more than one queen on each row. In English, the above LTL formula asserts that “at every state in the model, for all paths, it is not true that the constraints eventually always hold”. A violation of this property clearly must be a solution to the problem. The \square operator is necessary to ensure that duplicates of solutions are not reported, due to the way SPIN verifies LTL formulae.¹

We have written a PERL script which generates a Promela model and never claim for the *n*-queens problem, given $n > 0$. The code for this script is available on our website [5]. Table 1 shows the number of solutions to the *n*-queens problem found by our setup for $n \leq 10$, along with the time and memory requirements. For $n \leq 9$, all solutions are found. It was not possible to find any solutions for $n = 10$ before available main memory was exhausted.² This shows that although the

¹In SPIN all finite paths are extended to infinite paths by the creation of a loop from a final state to itself. As we have asked the model checker to report *all* errors, without the \square operator a different error would be reported depending on how many times this loop is executed, thus the same solution would be returned multiple times.

²We could use the minimised automaton method of SPIN [17] to

Table 1: Verification results for the n -queens Promela model without symmetry reduction

n	solutions found	states searched	time	memory (Mb)
4	2	332	0.01 s	2.3
5	10	875	0.01 s	2.3
6	4	14,850	0.05 s	2.7
7	40	121,282	0.41 s	6.4
8	92	1.11×10^6	4.50 s	43.9
9	352	1.12×10^7	1 m 47 s	428.3
10	0	7.70×10^7	4 m 31 s	3039.0

problem can in principle be solved using model checking, it is not an effective approach.

6 Symmetry in the n -queens problem

Consider the solution to the 4-queens problem shown on the left in figure 1. Observe that flipping the board through the vertical axis results in the solution shown on the right. These solutions are said to be *symmetric*, the symmetry being reflection through the vertical axis.

In general there are 8 symmetries of the n -queens problem for any n . Geometrically these symmetries are: the identity, rotation by 90° , 180° and 270° , reflection through the vertical axis, and reflection through the vertical axis combined with rotation through 90° , 180° or 270° . These symmetries form a group (called the dihedral group of order 8) that preserves solutions to the n -queens problem. The solution preserving property is illustrated by both figure 1 which shows symmetric solutions, and figure 3 which shows two symmetric assignments which are not solutions.

Symmetry in the 8-queens problem was investigated as early as 1874 by Glaisher [15].

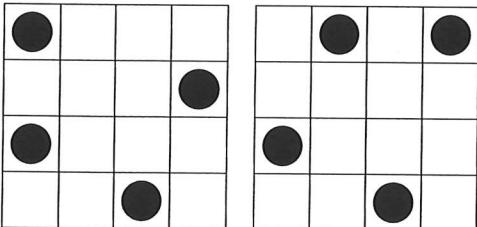


Figure 3: Symmetric configurations of the 4 queens problem which are not solutions.

6.1 Symmetry breaking in the CSP approach

The n -queens problem has been used as an example by various authors investigating symmetry in constraint processing [4; 14]. Three methods of symmetry breaking which have been used are symmetry breaking before search, symmetry

try to verify larger configuration. This method trades memory for time. While it can handle extremely large state spaces, verification can be very slow.

breaking during search, and symmetry breaking by assignment ordering. We briefly summarise the advantages and disadvantages of these techniques for the n -queens problem.

Symmetry breaking before search

It is possible to add symmetry breaking constraints to a CSP representation of the n -queens problem before search to reduce the number of symmetrically equivalent solutions that are reported [14; 22]. For example, to eliminate 180° rotational symmetry we could add constraints in the following manner [14]:

- $q_1 \leq n + 1 - q_n$
- if $q_1 = n + 1 - q_n$ then $q_2 \leq n + 1 - q_{n-1}$
- if $q_1 = n + 1 - q_n$ and $q_2 = n + 1 - q_{n-1}$ then $q_3 \leq n + 1 - q_{n-2}$

etc. This method requires a large number of constraints to be added to the problem if all symmetries are to be broken. Breaking all symmetries in practice can prove difficult [22], and symmetry breaking constraints do not always work well with variable ordering heuristics, especially if only one solution is required [14].

Symmetry breaking during search

In [14], the method of symmetry breaking during search is applied to the n -queens problem, using ILOG Solver. Seven functions are written, one for each non-trivial symmetry of the problem. Run time is shown to be cut by up to 75%, and the number of backtracks is greatly reduced as n increases. All symmetry of the problem is exploited, so that all the solutions found are symmetrically distinct. Since the number of symmetries of the problem is small, and does not change as n increases, the SBDS approach is very effective when applied to the n -queens problem [14].

Symmetry breaking by assignment ordering

A symmetric backtracking algorithm with symmetry is presented in [4]. Given a symmetry group, the algorithm works by imposing an ordering on variable assignments, and uses a function, *Symtest*, to check for each partial assignment whether or not it is equivalent to a smaller one using the symmetry group.

Experimental results using the n -queens problem show that for $n \leq 6$ the overhead of symmetry computations means the algorithm using symmetry takes longer than one without symmetry, but for larger values of n the algorithm with symmetry is 2.6 times faster than without. The algorithm with symmetry could also handle cases of the problem which were not tractable without symmetry.

6.2 Symmetry reduction in the model checking approach

We show how the approach to symmetry reduction using quotient structures can be applied to the Promela model described in section 5.2. Definition 2 of a quotient Kripke

structure, and algorithm 1 for constructing a quotient Kripke structure, both make use of a function *rep* which, given a state *s*, computes a unique representative from the equivalence class $[s]$ such that for all $t \in [s]$, $\text{rep}(t) = \text{rep}(s)$. It would be possible to modify the SPIN search algorithm to store a unique representative of each state, and this is done in [3] for total symmetries. However, such a modification to SPIN would be quite tricky, and would tie the symmetry reduction technique to the current version of the model checker. For this reason we alter our model so that once a state has been reached, but *before* SPIN has stored the state, the state is *canonicalised*, i.e. it is converted into a unique representative from its equivalence class. This has *exactly* the same effect as altering the SPIN search according to algorithm 1. The Promela code for the symmetry reduced model is the same as that given in section 5.2, with a call to the function *rep* added as follows:

```
init {
  do
    :: result==1 -> skip
    :: else ->
      atomic { if
        :: q[0]==u ->
          ....
        fi;
        rep();
      }
  od
}
```

The function *rep* applies each symmetry of the problem to the array *q* in turn, and selects (as a representative) the assignment which is smallest using the natural lexicographical ordering on the array. This is similar to the approach used in [4] of assigning an ordering on assignments and returning only the smallest solution in each equivalence class under this ordering. Here is an excerpt of the Promela code:

```
inline rep() {
  d_step {
    copy(q,min); wipe(temp); r90(q,temp);
    if
      :: lt(temp,min) -> copy(temp,min) :: else -> skip
    fi;
    wipe(temp); r180(q,temp);

    if
      :: lt(temp,min) -> copy(temp,min) :: else -> skip
    fi;
    wipe(temp); r270(q,temp);
    ....
    copy(min,q);
  }
}
```

As can be seen from the code, there is a function for each symmetry in the problem. The function *r90(a,b)*, which turns *b* into a configuration of *a* rotated through 90°, is implemented (for the case $n = 4$) as follows:

```
inline r90(a,b) {
  if
    :: a[0]==u
    :: else b[a[0]]=n-1
  fi;
  if
    :: a[1]==u
    :: else b[a[1]]=n-2
  fi;
  if
    :: a[2]==u
    :: else b[a[2]]=n-3
  fi;
```

Table 2: Verification results for the n -queens problem with symmetry reduction

n	solutions found	states searched	time	memory (Mb)
4	1	137	0.01 s	2.3
5	2	742	0.01 s	2.3
6	1	4,901	0.01 s	2.4
7	6	37,994	0.18 s	3.5
8	12	337,070	1.61 s	14.5
9	46	3.35×10^6	23.06 s	129.1
10	92	3.70×10^7	17 m 20 s	1478.7
11	46	6.93×10^7	6 m 33 s	3039.0

```
fi;
if
  :: a[3]==u
  :: else b[a[3]]=n-4
fi;
}
```

The others functions are defined in a similar way. The function *copy(a,b)* makes copies the array *a* into the array *b*. This has been implemented since Promela does not support the operation of assigning one array to another. Similarly, *wipe(a)* sets each element of the array *a* to zero. The function *lt(a,b)* returns true if array *a* is lexicographically smaller than array *b*.

Providing a function for each nontrivial symmetry of the problem is similar to the way SBDS is applied to the problem in [14].

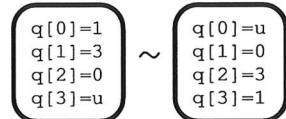


Figure 4: These assignments are symmetrically equivalent, but the one on the right is not reachable in the original model.

An important point to note is that our model is only partially symmetric. We assign values to the array positions $q[0], q[1], \dots, q[n - 1]$ in a fixed order, and so the configuration shown on the left in figure 4 is a reachable configuration, but the one on the right is not. However, the two configurations are symmetrically equivalent. This means that the original model is not *closed* under the symmetries of the problem: given a state and a symmetry it may be that the state resulting from applying the symmetry is not a reachable state of the model. The work of Emerson and Tretter [10] on *rough symmetry* in model checking ensures that in this case the quotient model obtained by applying symmetries of the problem is still behaviourally equivalent to the original model for symmetric properties even though the symmetry is only partial. The justification for this is that each variable $q[i]$ ($0 \leq i < n$) can be seen as a prioritised process which assigns itself a value, such that process $q[i]$ is blocked until process $q[j]$ has assigned itself a value for each $j < i$. For more details of symmetry in prioritised systems see [10].

We have extended our PERL script to allow the inclusion of symmetry reduction in the generated Promela model. Table 2 shows the number of solutions to the n -queens problem found by our model using symmetry reduction for $n \leq 11$, along with the time and memory requirements. Since all symmetries of the problem have been exploited, only symmetrically distinct solutions are returned. For $n \leq 10$, all symmetrically distinct solutions are found. Despite the overhead of the *rep* function, using symmetry reduction is always at least as fast as not using it. For the case $n = 8$, using symmetry reduces the number of explored states by a factor of more than 3, and search is 4.7 times faster. We were also able to find all (symmetrically distinct) solutions for the case $n = 10$, which was not possible without exploiting symmetry. For the case $n = 11$, it was possible to find 46 of the 341 symmetrically distinct solutions before memory was exhausted.

Recall from section 6.1 the three aims of a symmetry exclusion method proposed by Gent and Smith [14]. We have demonstrated that symmetry reduction using quotient structures fulfills the first and last of these aims—this approach has avoided finding symmetrically equivalent solutions, and we could deal with different forms of symmetry in other problems by coding the *rep* function differently. In fact it would be possible to automatically generate the *rep* function given generators of a symmetry group, using a similar approach to [4]. The second aim, that heuristic choice should be represented, is not applicable in this situation since search heuristics are not usually used in model checking (this is one of the reasons that model checking performs so poorly in solving the n -queens problem). Although symmetry reduction has made larger instances of the problem tractable, the largest instance we can handle using model checking is still much smaller than the largest instances successfully handled by other approaches [4; 14].

7 Related work

Symmetry reduction using SPIN is applied to a model of the game *Tic Tac Toe* in [18]. The symmetries of this model are the same as the rotation and reflection symmetries of the n -queens problem. Symmetry reduction is implemented in a way similar to our approach, by incorporating a canonicalisation function into the model rather than changing the SPIN model checker. In [16], model checking with SPIN is used to solve the *rehearsal problem*, a constraint satisfaction problem. Symmetry is exploited in the model by adding additional constraints rather than by building a quotient structure of unique representatives. This is analogous to the technique of symmetry reduction before search in CSPs, discussed in section 6.1

In order to exploit the partial symmetry in our Promela verification model we use the results of Emerson and Trefler [10] on systems of identical processes which are “nearly” symmetric. Emerson and Trefler have also worked on the problem of exploiting symmetry at the *source code level*, translating a symmetric program into a reduced program *before* model checking, then model checking the reduced program. It would be interesting to compare this with the approach of converting a symmetric CSP into an asym-

metric one before search to avoid duplicate solutions [20; 8].

8 Conclusions and future work

We have presented a Promela verification model to find all solutions of the n -queens problem, and have applied symmetry reduction techniques to speed up search, and to make larger instances of the problem tractable. We have also summarised approaches to symmetry breaking for this problem in constraint processing. Through this example it is clear that symmetry reduction in model checking shares similarities with SBDS in constraint programming, in that functions must be provided for each nontrivial symmetry in the problem, and these functions are used during search to avoid the exploration of symmetrically equivalent portions of the search space. The method of symmetry breaking by assignment ordering [4], which returns only the smallest solution in each equivalence class under a defined ordering, is also similar to symmetry reduction using quotient structures. The states of a quotient structure consist of a unique representative from each equivalence class of the original structure under the symmetry group, and we have followed the common approach of using the lexicographically smallest state from each class as a representative [3].

In this paper we have concentrated on the concepts of symmetry reduction and symmetry breaking rather than on the theoretical details. However, we intend to formalise our observations at a theoretical level, and try to establish exactly what the correspondence is between symmetry breaking in constraint processing and symmetry reduction in model checking.

It would be possible to apply the approach of symmetry breaking *before* search to our verification model. Although this technique has been used in model checking for specific examples [16], to our knowledge applying this idea to model checking has not been studied for arbitrary forms of symmetry. Emerson and Trefler [10] have investigated symmetry reduction at the source code level for *fully* symmetric systems. We intend to investigate this further.

One of the reasons that model checking performs so poorly in solving constraint satisfaction problems is that it stores the entire search space in main memory. This is because Kripke structures typically have cycles leading back to old states. However, in our formulation of this problem it is clear that the underlying Kripke structure is a tree (apart from the stuttering extension of final states), so it would be possible to use stateless search to find solutions. Although a stateless variation of the SPIN search algorithm is discussed in [17], it is not currently implemented in SPIN.

An important problem in using symmetry with model checking and constraint processing is *symmetry detection*, that is determining symmetries of a models or problems from their specifications. The challenge is to allow symmetry to be exploited without requiring knowledge of group theory on the part of the modeller. Ip and Dill proposed a new data type called *scalarset* which makes the specification and detection of symmetries trivial for models of concurrent systems [19]. It would be interesting to try to apply the notion of a scalarset

to constraint satisfaction problems as a way to specify problem symmetries easily.

References

- [1] Rolf Backofen. Symmetries: To break or not to break, that's the question. (extended abstract). In *Proceedings of the Joint Annual Workshop of the ERCIM Working Group on Constraints and the CoLogNET area on Constraint and Logic Programming*, Budapest, Hungary, 2003.
- [2] R.A. Bosch. Peaceably coexisting armies of queens. *Optima (Newsletter of the Mathematical Programming Society)*, 62:6–9, 1999.
- [3] D. Bosnacki, D. Dams, and L. Holenderski. Symmetric spin. *International Journal on Software Tools for Technology Transfer*, 4(1):65–80, 2002.
- [4] C.A. Brown, L. Finkelstein, and P.W. Purdom. Backtrack searching in the presence of symmetry. In T. Mora, editor, *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, volume 357 of *Lecture Notes in Computer Science*, pages 99–110, 1988.
- [5] M. Calder and A. Miller. Veriscope publications website:
<http://www.dcs.gla.ac.uk/research/veriscope/publications.html>.
- [6] Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.
- [7] E.M. Clarke, R. Enders, T. Filkorn, and S. Jha. Exploiting symmetry in temporal logic model checking. *Formal Methods in System Design*, 9(1–2):77–104, 1996.
- [8] J. Crawford, M. Ginsberg, E. Luks, and A. Roy. Symmetry-breaking predicates for search problems. In *Proceedings of Fifth International Conference on Principles of Knowledge Representation and Reasoning*, pages 149–159, 1996.
- [9] G. Delzanno and A. Podelski. Model checking in CLP. In *tacas99*, pages 223–239, 1999.
- [10] E. Allan Emerson and Richard J. Trefler. From asymmetry to full symmetry: New techniques for symmetry reduction in model checking. In Laurence Pierre and Thomas Kropf, editors, *Proceedings of the 10th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME '99)*, volume 1703 of *Lecture Notes in Computer Science*, pages 142–156, Bad Herrenalb, Germany, September 1999. Springer-Verlag.
- [11] L. Fribourg. Constraint Logic Programming Applied to Model Checking. In Annalisa Bossi, editor, *Logic Programming Synthesis and Transformation, 9th International Workshop, LOPSTR'99*, volume 1817 of *Lecture Notes in Computer Science*, pages 30–41, Venezia, Italy, September 2000. Springer.
- [12] I. Gent, W. Harvey, T. Kelsey, and S. Linton. Generic SBDD using computational group theory. In Francesca Rossi, editor, *Principles and Practice of Constraint Programming 9th International Conference - CP 2003*, volume 2833 of *Lecture Notes in Computer Science*, pages 333–347, Kinsale, Ireland, September 2003. Springer.
- [13] I.P. Gent, W. Harvey, and T. Kelsey. Groups and constraints: Symmetry breaking during search. In Pascal Van Hentenryck, editor, *Principles and Practice of Constraint Programming 8th International Conference - CP 2002*, volume 2470 of *Lecture Notes in Computer Science*, pages 415–430, Ithaca, NY, USA, September 2002. Springer.
- [14] I.P. Gent and B. Smith. Symmetry breaking in constraint programming. In W. Horn, editor, *Proceedings of the 14th European Conference on Artificial Intelligence, ECAI 00*, pages 599–603, Berlin, Germany, August 2000. John Wiley and Sons, Chichester.
- [15] J.W.L. Glaisher. On the problem of the eight queens. *Philosophical Magazine*, 4(48):457–467, 1874.
- [16] P. Gregory, A. Miller, and P. Prosser. Solving the rehearsals problem with planning and with model checking. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI 2004), workshop W14: Modelling and Solving Problems with Constraints*, Valencia, Spain, 2004. to appear.
- [17] Gerard J. Holzmann. *The SPIN model checker: primer and reference manual*. Addison Wesley, Boston, 2003.
- [18] G.J. Holzmann and R. Joshi. Model-driven software verification. In Susanne Graf and Laurent Mounier, editors, *Proceedings of the 11th International SPIN Workshop (SPIN 2004)*, volume 2989 of *Lecture Notes in Computer Science*, pages 76–91, Barcelona, Spain, April 2004. Springer-Verlag.
- [19] C.Norris Ip and D. Dill. Better verification through symmetry. *Formal Methods in System Design*, 9:41–75, 1996.
- [20] J.F. Puget. On the satisfiability of symmetrical constrained satisfaction problems. In J. Komorowski and Z.W. Ras, editors, *Methodologies for Intelligent Systems (Proceedings of ISMIS '93)*, volume 689 of *Lecture Notes in Artificial Intelligence*, pages 350–361, 1993.
- [21] B. Smith. Constraint programming in practice: Scheduling a rehearsal. Technical Report APES-67-2003, APES research group, 2003.
- [22] Barbara Smith. Personal communication, 2004.
- [23] Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *Proceedings, Symposium on Logic in Computer Science*, pages 332–344. IEEE Computer Society, June 1986.

Approaches to Conditional Symmetry Breaking*

Ian P. Gent¹, Iain McDonald¹, Ian Miguel², Barbara M. Smith³

¹School of Computer Science, University of St Andrews, St Andrews, Fife, KY16 9SS, UK

²Department of Computer Science, University of York, YO10 5DD, UK

³Cork Constraint Computation Centre, Department of Computer Science,
University College Cork, Cork, Ireland

{ipg,iain}@dcs.st-and.ac.uk, ianm@cs.york.ac.uk, b.smith@4c.ucc.ie

Abstract

Conditional symmetry arises in a sub-problem of a constraint satisfaction problem, where the sub-problem satisfies some *condition* under which additional symmetries hold. Typically, the condition is a set of assignments of values to variables, i.e. a *partial assignment* reached during systematic search. Like unconditional symmetry, conditional symmetry can cause redundancy in a systematic search for solutions. Breaking this symmetry, in addition to breaking unconditional symmetry, is therefore an important part of solving a constraint satisfaction problem effectively. This paper examines three ways in which this can be done: by adding conditional symmetry-breaking constraints, by reformulating the problem to remove the symmetry, and by augmenting the search process to break the conditional symmetry dynamically.

1 Introduction

Constraint programming has been used with great success to tackle a wide variety of combinatorial problems in industry and academia. In order to apply constraint programming tools to a particular domain, the problem must be *as a constraint program. However, constraints provide a rich language, so typically many alternative models exist for a given problem, some of which are more effective than others. Constructing an effective constraint program is a difficult task.*

One important aspect of modelling is dealing with *symmetry*. Symmetry is a solution-preserving transformation; symmetry in a model can result in a great deal of wasted effort when the model is solved via systematic search. Hence, it is necessary to ensure that symmetry

*Most of this work was done while the 4th author was employed at the University of Huddersfield. We thank Alan Frisch, Chris Jefferson, Karen Petrie and Steve Prestwich for useful discussions, and our anonymous reviewers for their insightful comments. Ian Gent is supported by a Royal Society of Edinburgh SEELLD/RSE Support Research Fellowship. Ian Miguel is supported by a UK Royal Academy of Engineering/EPSRC Post-doctoral Research Fellowship.

is broken effectively. The majority of research in symmetry in constraint models considers only the symmetry present in a model before search begins. However, as we will discuss, it is commonly the case that symmetry can form during search. We refer to this as *conditional* symmetry, since its formation depends on the choices made during search. In order to avoid redundant search, it is important to break this symmetry in addition to unconditional symmetry breaking.

All symmetry breaking trades the reduction in search gained versus the cost of breaking the symmetry. The detection of the condition for the symmetry to arise adds a further cost when creating a conditional symmetry-breaking scheme. Therefore, the reduction in search generally has to be significant to make the effort worthwhile. A further consideration is how frequently during search the condition is likely to be satisfied. The greater the proportion of search that is within sub-problems where conditional symmetry arises, the greater the impact conditional symmetry breaking is likely to have.

This paper discusses three ways to deal with conditional symmetry. First, to add constraints to a model to detect and break the symmetry as it arises. Second, to reformulate the problem such that the new model does not have the conditional symmetry. Finally, we discuss how conditional symmetry can be broken during search.

2 Background

The finite domain *constraint satisfaction problem* (CSP) consists of a triple $\langle X, D, C \rangle$, where X is a set of variables, D is a set of domains, and C is a set of constraints. Each $x_i \in X$ is associated with a finite domain $D_i \in D$ of potential values. A variable is *assigned* a value from its domain. A constraint $c \in C$, constraining variables x_i, \dots, x_j , specifies a subset of the Cartesian product $D_i \times \dots \times D_j$ indicating mutually compatible variable assignments. A *constrained optimisation problem* is a CSP with some objective, which is to be optimised.

A *partial assignment* is an assignment to one or more elements of X . A solution is a partial assignment that includes all elements of X and satisfies all the constraints. This paper focuses on the use of systematic search through the space of partial assignments to find such solutions. A sub-CSP, P' , of a CSP P is obtained

from P by adding one or more constraints to P . Note that assigning a value v to a variable x is equivalent to adding the constraint $x = v$.

A *symmetry* in a CSP is a bijection mapping solutions to solutions and non-solutions to non-solutions. A *conditional symmetry* of a CSP P holds only in a sub-problem P' of P . The condition for the symmetry to arise is the conjunction of constraints necessary to generate P' from P . Conditional symmetry is a generalisation of unconditional symmetry, since unconditional symmetry can be seen as a conditional symmetry with an empty condition. For the most part, we focus herein on conditions in the form of partial assignments.

As with unconditional symmetry, we can distinguish between *instance-independent* conditional symmetry, which has the potential to arise in all elements of a problem class, and *instance-dependent* conditional symmetry, which can arise in some instances of a problem class, but not in others. We will see examples of both types in the following section.

Symmetry breaking is made difficult by the interaction of the various symmetries in a problem. Breaking some or all of one symmetry might break some, all or none of another. This remains true of conditional symmetry, as will be discussed. Furthermore, we will see how breaking an unconditional symmetry can simplify the condition of a conditional symmetry, and therefore reduce the cost of conditional symmetry breaking. Symmetry can often be broken in a variety of ways. Hence, it might be preferable to choose a scheme whose consequences are beneficial to conditional symmetry breaking in this way.

3 Conditional Symmetry-breaking Constraints

One method of breaking conditional symmetries is to add symmetry-breaking constraints of the form:

$$\text{condition} \rightarrow \text{symmetry-breaking constraint}$$

where *condition* is a conjunction of constraints, for instance a partial assignment such as $x = 1 \wedge y = 2$, that must be satisfied for the symmetry to form. As in unconditional symmetry breaking [1], the symmetry-breaking constraint usually takes the form of an ordering constraint on the conditionally symmetric objects. This section discusses two case studies of breaking conditional symmetry in this way.

3.1 Graceful Graphs

The first case study is of conditional symmetry in finding a graceful labelling [5] of a class of graphs. As noted in [5], labelled graphs have a variety of applications, ranging from coding theory to radar astronomy. A labelling f of the vertices of a graph with e edges is *graceful* if f assigns each vertex a unique label from $\{0, 1, \dots, e\}$ and when each edge xy is labelled with $|f(x) - f(y)|$, the edge labels are all different. (Hence, the edge labels are a permutation of $1, 2, \dots, e$.)

Finding a graceful labelling of a given graph, or proving that one does not exist, can easily be expressed as a constraint satisfaction problem. The CSP has a variable for each vertex, x_1, x_2, \dots, x_n each with domain $\{0, 1, \dots, e\}$ and a variable for each edge, d_1, d_2, \dots, d_e , each with domain $\{1, 2, \dots, e\}$.

The constraints of the problem are:

- if edge k joins vertices i and j then $d_k = |x_i - x_j|$
- x_1, x_2, \dots, x_n are all different
- d_1, d_2, \dots, d_e are all different (and form a permutation)

Figure 1 shows an instance of a class of graphs listed in Gallian's survey of graceful graphs [5] as $C_n^{(t)}$: they consist of t copies of a cycle with n nodes, with a common vertex. For $n = 3$, these graphs are graceful whenever $t \equiv 0$ or $1 \pmod{4}$, so the graph shown is graceful.

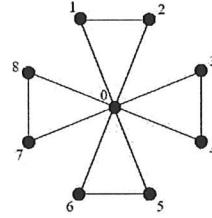


Figure 1: The windmill graph $C_3^{(4)}$

The nodes in Figure 1 are numbered to show the numbering of the variables in the CSP model, i.e. node 0 is the centre node and is represented by the variable x_0 .

The CSP has three instance-independent symmetries:

- swap the labels of the nodes other than the centre node in any triangle, e.g. swapping the labels of nodes 1 and 2;
- permute the triangles, e.g. swap the labels of nodes 1 and 2 with those of nodes 3 and 4;
- change every node label x_i for its complement $e - x_i$.

The centre node cannot have a label > 1 and $< e - 1$. Since there must be an edge connecting two nodes labelled 0 and e , if the centre node's label is not 0 or e , then two other nodes in a triangle, e.g. nodes 1 and 2, must be labelled 0 and e . But then, unless the centre node is labelled 1 or $e - 1$ there is no way to label an edge $e - 1$, given that the largest node label is e . The labels 0, 1, $e - 1$ and e are possible for the centre node, however, if there is a graceful labelling.

Relabelling a Triangle

Consider a graceful labelling of a graph in this class, with the centre node labelled 0. In any triangle, where the other two nodes are labelled a and b , with $a < b$, we can replace a with $b - a$ to get another solution. Note that $b - a$ was unused: otherwise, since the centre is labelled 0, there would have been two edges labelled $b - a$. Figure 2 shows how the edge labels in the triangle are permuted.

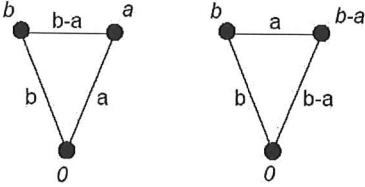


Figure 2: Relabelling a triangle in a graceful labelling with centre node labelled 0.

Any graceful labelling of $C_3^{(t)}$ with centre node labelled 0 has 2^t equivalent labellings by changing or not changing the labels within each of the t triangles in this way. However, for a specific instance of this symmetry, on nodes 0, 1, 2, say, in order to describe its effect, if any, we need two pieces of information. We need to know both whether node 0 is labelled 0 and which of nodes 1 and 2 has the smaller label; hence, we need to know the assignments to these three variables.

A graceful labelling with the centre node labelled 1 can be transformed into an equivalent labelling similarly: a triangle labelled 1, a , b , with $a < b$ can be relabelled 1, $b - a + 1$, b . Again, this is conditional on the three assignments. There are equivalents for the other possible labels for the centre node, i.e. $e - 1$ and e .

Labellings with Centre Node Labelled 1

As already mentioned, there are graceful labellings where the centre node is labelled 1. In such a labelling, there must be a triangle labelled 1, 0, e , since there must be an edge whose endpoints are labelled 0 and e . The remaining nodes have labels in the range 3, .., $e - 1$. (There is no edge labelled 2, since it has to be connected to the centre node, giving a second edge labelled 1.)

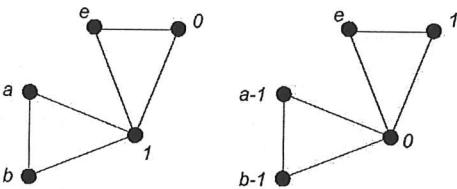


Figure 3: Transforming a labelling with centre node labelled 1.

Figure 3 (left) shows the 1, 0, e triangle and another representative triangle. We can transform the labels of all the nodes as shown on the right. If the original labelling is graceful, so is the transformed labelling, and the centre node is now labelled 0. Hence, any labelling with centre node labelled 1 is equivalent to one with centre node labelled 0. The reverse is true *only* if there exists a triangle labelled 0, 1, e . The condition for this symmetry is thus a conjunction of constraints specifying that a 0, 1, e triangle exists. Similarly, if the centre node is labelled $e - 1$, we can transform any resulting graceful

labelling into one with the centre node labelled e , and vice versa if there exists a triangle labelled 0, $e - 1$, e .

We can also view these particular conditional symmetries as *dominance* relations [10]. Labelling the centre node with 0 dominates labelling it with 1, and labelling the centre node with e dominates labelling it with $e - 1$. These relations are typically exploited by disallowing the dominated value, which, as we will see, corresponds to the way we break the conditional symmetry here. Exploring the connection between dominance and conditional symmetry is an important item of future work.

Symmetry-Breaking Constraints

Ignoring the conditional symmetries for now, the symmetries of the CSP can easily be eliminated by adding constraints to the model.

1. In each triangle, we can switch the labels of the nodes that are not the central node. Constraints to eliminate this are: $x_{2i-1} < x_{2i}, i = 1, 2, \dots, t$
2. We can permute the triangles. Given the previous constraints, we can add the following to eliminate this: $x_{2i-1} < x_{2i+1}, i = 1, 2, \dots, t - 1$
3. To eliminate the complement symmetry, we can post: $x_0 < e/2$.

The conditional symmetry where the central node is 0 can be eliminated easily. It also requires knowing which of the two other nodes in each triangle has the smaller label. Notice that, because of symmetry-breaking constraint 1 above, this condition is simplified: the node with the smaller label has the smaller index. Hence, we can add a conditional constraint: if $x_0 = 0$, then $2x_{2i-1} < x_{2i}$, for $i = 1, 2, \dots, t$. In terms of Figure 2, we choose the labelling 0, a , b for the triangle and want this to be lexicographically smaller than 0, $b - a$, b .

Given symmetry-breaking constraint 3 to eliminate the complement symmetry, 0 and 1 are the only possible labels for the central node. We have shown that the labellings with the central node labelled 1 are equivalent to some of the labellings with node 0 labelled 0. Hence, we can simply add $x_0 = 0$ to eliminate the conditional symmetry where there exists a triangle labelled 0, 1, e . This, in turn, simplifies the conditional constraints given earlier: if we know that $x_0 = 0$, then we can drop the condition from the earlier constraints and just have $2x_{2i-1} < x_{2i}$, for $i = 1, 2, \dots, t$.

Hence, in this example, all the symmetries, including the conditional symmetry, can be eliminated by simple constraints.

Results

Using symmetry-breaking constraints (1-3) to eliminate the graph and complement symmetries, the graph in Figure 1 has 144 graceful labellings. Eliminating the conditional symmetries reduces these to 8. The resulting reduction in search would be greater still for larger graphs in the same class, $C_3^{(t)}$. This case study demonstrates that conditional symmetry can sometimes be eliminated with little overhead and reduce the search effort enormously.

3.2 Steel Mill Slab Design

Our second case study is the steel mill slab design problem [4] (problem 38 at www.csplib.org). Steel is produced by casting molten iron into slabs. A finite number, σ , of *slab sizes* is available. An order has two properties, a *colour* corresponding to the route required through the steel mill and a *weight*. The problem is to pack the d input orders onto slabs such that the total slab capacity is minimised. There are two types of constraint:

1. **Capacity constraints.** The total weight of orders assigned to a slab cannot exceed the slab capacity.
2. **Colour constraints.** Each slab can contain at most p of k total colours (p is usually 2). This constraint arises because it is expensive to cut the slabs up to send them to different parts of the mill.

A Matrix Model

It is natural to use a matrix model to represent this problem. Under the assumption that the largest order is smaller than the largest slab, at most d slabs are required to accommodate all the input orders. Hence, a one-dimensional matrix of size d , $slab_M$, can be used to represent the size of each slab used, with a size of zero indicating that there is no corresponding slab in the solution. In addition, a $d \times d$ 0-1 matrix, $order_M$, can be used to represent the assignment of orders to slabs, where a ‘1’ entry in the i th column and j th row indicates that the i th order is assigned to the j th slab. Constraints on the rows ensure that the slab capacity is not exceeded:

$$\forall j \in \{1..d\} : \sum_{i \in \{1..d\}} weight(i) \times order_M[i, j] \leq slab_M[j]$$

where $weight(i)$ is a function mapping the i th order to its weight. Constraints on the columns ensure that each order is assigned to one and only one slab:

$$\forall i \in \{1..d\} : \sum_{j \in \{1..d\}} order_M[i, j] = 1$$

A second 0-1 matrix, $colour_M$ with dimensions $k \times d$, is used to represent the relation between the slabs and the colours. A ‘1’ entry in the i th column and j th row indicates that the i th colour is present on the j th slab. Constraints link $order_M$ and $colour_M$:

$$\begin{aligned} \forall i \in \{1..d\} \forall j \in \{1..d\} : \\ order_M[i, j] = 1 \rightarrow colour_M[colour(i), j] = 1 \end{aligned}$$

where $colour(i)$ is a function mapping the i th order to its colour. Constraints on the rows of $colour_M$ ensure that orders with at most p colours are assigned to each slab:

$$\forall j \in \{1..d\} : \sum_{i \in \{1..k\}} colour_M[i, j] \leq p$$

Symmetry and Conditional Symmetry

In this initial model, $slab_M$ has instance-independent column symmetry: a (non-)solution can be transformed into a (non-)solution by permuting the values assigned to

each element of $slab_M$ and permuting the corresponding rows of $order_M$. This symmetry can be broken simply by ordering the elements of $slab_M$ as follows:

$$slab_M[1] \geq slab_M[2] \geq \dots slab_M[d]$$

Furthermore, $order_M$ has instance-dependent partial column symmetry. When two orders have equal weight and colour, a (non-)solution can be mapped to a (non-)solution by exchanging the columns associated with these two orders. This symmetry can be broken by combining symmetric orders into a single column. The sum of that column is then constrained to be equal to the number of orders it represents.

Symmetry on the rows of $order_M$ is, however, conditional: only if two slabs have equal size can their contents be exchanged in a (non-)solution to obtain a (non-)solution. This symmetry is instance-independent. Notice also that the unconditional symmetry-breaking on $slab_M$ simplifies breaking the conditional slab symmetry, since it constrains rows of $order_M$ representing equal-sized slabs to be adjacent. Hence, conditional slab symmetry can be broken statically in a straightforward manner using lexicographic ordering:

$$\begin{aligned} \forall i \in \{1..d-1\} : \\ (slab_M[i] = slab_M[i+1]) \rightarrow (order_M[i] \geq_{lex} order_M[i+1]) \end{aligned}$$

There is a further instance-dependent symmetry conditional on the way that orders are assigned to slabs. Consider 3 ‘red’ orders, order a of weight 6 and two instances of order b , with weight 3 (the last two are represented by a single column). Consider the following partial assignments to $order_M$:

$$\begin{array}{ccc} a & b & \dots \\ slab_1 & \left(\begin{array}{ccc} 1 & 0 & \dots \end{array} \right) & slab_1 & \left(\begin{array}{ccc} 0 & 2 & \dots \end{array} \right) \\ slab_2 & \left(\begin{array}{ccc} 0 & 2 & \dots \end{array} \right) & slab_2 & \left(\begin{array}{ccc} 1 & 0 & \dots \end{array} \right) \\ \dots & \dots & \dots & \dots \end{array}$$

These assignments are symmetrical. Note that the symmetry is conditional on both instances of b being assigned to the same slab, effectively creating a single ‘super’ order symmetrical to a . This is the simplest case of *compound* order symmetry, where individual orders combine to become symmetrical to single larger orders (such as the instance of a in the example — we will call these *unit* compounds) or other compounds.

The conditional lexicographic ordering of the contents of the slabs interacts with the compound order symmetry. In the above example, if the size of the first slab is the same as that of the second then only the first partial assignment is allowed. This is not, however, the case in general. We now describe how compound order symmetry can be detected and broken effectively.

Formation of Compound Order Symmetry

To break compound order symmetry, we must know when and where the symmetry forms. For simplicity, we consider only compound orders composed from multiple instances of the same order. The encoding described

$$a) \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} b) \begin{pmatrix} 3 \\ 3 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} c) \begin{pmatrix} 0 \\ 0 \\ 6 \\ 0 \\ 0 \\ 0 \end{pmatrix} d) \begin{pmatrix} 1 \\ 1 \\ 3 \\ 1 \\ 0 \\ 0 \end{pmatrix} e) \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} f) \begin{pmatrix} 3 \\ 2 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Figure 4: Conditional formation of two compound orders of size 3.

here can be extended straightforwardly to support compounds formed from orders of different sizes. Consider an instance with 6 red orders of size 1. The assignment of these orders to slabs is represented by a single column of $order_M$, whose sum is constrained to be six. Consider now the formation of a red compound order of size three. Up to two such compounds can form from the six red orders. Figure 4 presents example cases for which we must cater. In every example all the orders have been assigned to a slab, but in some cases one (Fig.4d, Fig.4e, Fig.4f) or both (Fig.4a) compounds have not formed.

It is useful to consider a first compound (formed from the first three orders, counting down the column) and a second compound (formed from the second three). Notice that, counting from the top of each column, a compound can form only when a sufficient number of orders have been assigned. In the example, this is three and six orders for the first and second compounds, respectively. To exploit this observation, for each column on which compound orders may appear, we introduce a column of variables, $subsum_M$, which record the cumulative sum of assigned orders read down the column. Figure 5 presents the $subsum_M$ variables for our examples.

Given the $subsum_M$ variables, we can introduce a *position* variable for each compound, whose domain is the set of possible slab indices, constrained as follows:

$$\begin{aligned} subsum_M[position - 1] &< compoundSize \times instanceNo \\ subsum_M[position] &\geq compoundSize \times instanceNo \end{aligned}$$

where *compoundSize* indicates the number of orders necessary to form the compound in question, and *instanceNo* denotes which of the compounds of *compoundSize* on this column that *position* is associated with. This pair of constraints ensure that *position* indicates a unique slab when the corresponding column of $order_M$ is assigned.

The remaining question, given some partial assignment, is whether the compound order associated with *position* has formed on the slab indicated by *position*. This is recorded in a 0/1 variable, *switch*, paired with each *position* variable and constrained as follows:

$$switch = (order_M[column][position] \geq compoundSize)$$

where *column* is the column of $order_M$ on which the compound may form.

Breaking Compound Order Symmetry

Consider n symmetrical compound orders. We order these compounds ascending by the column on which they

$$a) \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{pmatrix} b) \begin{pmatrix} 3 \\ 6 \\ 6 \\ 6 \\ 6 \\ 6 \end{pmatrix} c) \begin{pmatrix} 0 \\ 6 \\ 6 \\ 6 \\ 6 \\ 6 \end{pmatrix} d) \begin{pmatrix} 1 \\ 3 \\ 6 \\ 6 \\ 6 \\ 6 \end{pmatrix} e) \begin{pmatrix} 1 \\ 5 \\ 6 \\ 6 \\ 6 \\ 6 \end{pmatrix} f) \begin{pmatrix} 3 \\ 4 \\ 6 \\ 6 \\ 6 \\ 6 \end{pmatrix}$$

Figure 5: Assignments to $subsum_M$ variables corresponding to order variable assignments in Figure 4.

appear, breaking ties by ordering the ‘first’, ‘second’, …, ‘nth’ compounds in a column, as defined in the previous section, ascending. We denote the *switch* and *position* variables of the i th compound under this ordering as $switch_i$ and $position_i$. The conditional symmetry can be broken straightforwardly as follows:

$$\forall i < j \in \{1, \dots, n\} : \\ (switch_i = 1 \wedge switch_j = 1) \rightarrow position_i \leq position_j$$

We have been careful to ensure that these ordering constraints are compatible with the slab conditional symmetry breaking constraints given above. Consider the following example, where slabs one and two have equal size, and the compound formed from two instances of order three is symmetrical to an instance of order two:

$$\begin{array}{ccc} order_1 & order_2 & order_3 \\ slab_1 & \begin{pmatrix} 1 & 1 & 0 \end{pmatrix} \\ slab_2 & \begin{pmatrix} 1 & 0 & 2 \end{pmatrix} \end{array}$$

The lexicographic ordering constraints used to break the conditional symmetry on the slabs, and the compound order symmetry-breaking constraints, as given above, are both satisfied. If, however, the compound orders were ordered in the reverse direction, in this example they would clash with the lexicographic ordering constraints, potentially pruning solutions.

Note that we post the transitive closure of the ordering constraints. This is contrary to unconditional symmetry breaking; given a set of symmetrical objects, it is usually only necessary to order adjacent elements in the enumeration of the set [3]. However, we cannot be certain that any particular conditional symmetry will form.

Experimental Results

As noted, conditional slab symmetry is instance-independent, while compound order symmetry is instance-dependent. To experiment with the effects of both slab and compound order symmetry breaking, we constructed 12 instances where compound order symmetries were highly likely to form by using only one colour for all orders, and choosing the size and number of the smaller orders such that multiple instances of a small order sum to the size of one of the larger orders. In constructing these instances, we made use of the following observation: if the steel mill is able to create slabs whose size is small, the likelihood that individual orders will be assigned to a slab alone is increased, and therefore the likelihood that compounds will form is decreased.

Problem	No Conditional Symmetry Breaking		Slab Conditional Symmetry Breaking		Compound Order Conditional Symmetry Breaking		Slab & Compound Order Conditional Symmetry Breaking	
	Choices	Time(s)	Choices	Time(s)	Choices	Time(s)	Choices	Time(s)
1	18,014,515	1120	79,720	5.64	-	-	68,717	36.4
2	6,985,007	439	15739	1.45	-	-	13,464	6.79
3	7,721	0.741	1,798	0.26	6,461	3.48	1,472	0.971
4	155,438	8.86	60,481	4.10	49,234	31.0	30,534	16.2
5	146,076	7.48	56,590	3.45	46,599	23.4	27,921	12.4
6	117,240	6.01	49,098	2.82	39,411	17.7	24,112	9.70
7	147,148	7.1	60,035	3.34	70,881	36.3	37,672	18.0
8	171,781	8.02	77,187	4.13	80,557	37.1	45,293	19.3
9	206,138	9.52	92,526	4.87	97,072	44.9	53,666	23.0
10	348,716	16.6	140,741	7.55	178,753	94.8	84,046	41.5
11	313,840	15.7	130,524	7.21	164,615	98.5	79,621	44.4
12	266,584	13.9	110,007	6.19	138,300	82.5	68,087	37.8

Table 1: Steel Mill Slab Design: Experimental Results. Times to 3 significant figures. A dash indicates optimal solution not found within 1 hour. Hardware: PIII 750MHz, 128Mb. Software: Ilog Solver 5.3 (Windows version).

The results (Table 1) show that both types of conditional symmetry breaking reduce search significantly. In the case of slab symmetry breaking, the overhead of the symmetry-breaking constraints is negligible, hence there is also a reduction in time. The overhead of compound order symmetry breaking is more significant. Although this technique clearly reduces search — in the instances tested a further reduction of as much as 50% is gained by adding compound order symmetry breaking to slab symmetry breaking — overall time taken is increased.

These results confirm that if a conditional symmetry can be detected and broken cheaply, then it is probably worthwhile to do so. Another positive indicator is if the symmetry is likely to appear in many sub-problems. In the case of the steel mill, although there is a clear reduction in search gained from breaking compound order symmetry, the challenge is to make the encoding of detection of this symmetry sufficiently lightweight that it can be used without fear of increasing the overall effort.

4 Breaking Conditional Symmetry by Reformulation

While reformulation is very important, we discuss it only briefly in this paper as our major success in this area is discussed fully in another paper [8].

Formulation of constraint problems can be essential to success in solving them, affecting solution times by many orders of magnitude. So an appropriate reformulation of a constraint problem can turn an insoluble problem into a soluble one in practical terms.

Formulation and reformulation are equally, or even more, important for symmetry breaking. Different formulations of the same problem can have different numbers of symmetries. Also, one formulation can have symmetries which are easier to deal with than in another formulation. For example, in one formulation symmetries might be the full permutation group S_n , which is usually easy to deal with, while another formulation with fewer symmetries may yield a group that is more difficult to deal with. Thus, reformulation of a problem into a different model may be the critical step in dealing with symmetries. Unfortunately, there is no general technique for suggesting reformulations for breaking symmetry, and to

date it has been achieved only by the insight of the particular constraint programmer.

If anything, conditional symmetry intensifies the problems inherent in reformulation to break symmetry. This can be seen in the case of all-interval series problem, reported by Gent, McDonald and Smith [8]. They achieved a speedup of a factor of 50 on the state-of-the-art by reformulating the problem based on identifying a conditional symmetry. The problem itself has 4 symmetries, and the conditional symmetry doubles this to 8 where it occurs. In fact, one of n conditional symmetries *always* occurs. The reformulation in fact changed the problem slightly, so goes beyond what can be achieved by changing the constraint model of the original problem. In the new problem, the conditional symmetry has become unconditional, and indeed all n of the possible conditional symmetries apply in each case. So we have both increased the number of symmetries, and used formulation to make conditional symmetries unconditional: both of these tricks have the aura of a rabbit pulled out of the hat rather than a generalisable technique. The reason this works so well is that it is extremely easy to break all symmetry in the new problem, leading to the excellent runtimes, and from solutions to the new problem we can read off solutions to the original very easily. So this example shows how effective reformulation can be, without apparently suggesting how to do it in other cases of conditional symmetry.

In summary, there is little we can say in general about reformulating to break conditional symmetry. To achieve this seems to require considerable insight on a case-by-case basis, so general techniques for reformulation that could be useful even in families of constraint problems would be highly desirable, but remain in the future.

5 A Generic Method of Breaking Conditional Symmetries

It is preferable for breaking conditional symmetries — as it is for ordinary, non-conditional symmetries — to have a generic method where the symmetries and conditions can be described easily and broken efficiently. Hence, we examine how previous methods of breaking symmetries could be modified to cope with conditional symmetries.

Gent, McDonald and Smith provided two implementations of SBDS [9] modified to work for conditional symmetries [8]. These implementations provided proof of concept only as both had serious problems. In both methods the efficiency of constraint solving was reduced by its introduction. The first method required a different symmetry function for each *possible* conditional symmetry, and naturally there will always be many more than the unconditional symmetries. For example, the all-interval series problems has 4 unconditional symmetries but $4(n-1)$ conditional ones. The second modification of SBDS removed this problem, but the implementation was grounded heavily in the specific CSP. Thus no general purpose method proposed to date for conditional symmetries can be regarded as satisfactory.

In this section we show what the main disadvantage of using SBDS like approaches (such as GHK-SBDS [6]) is when dealing with conditional symmetry. We also explain how SBDD [2] can be modified to effectively deal with generic conditional symmetries, although implementing this modification remains future work.

5.1 The problem with using SBDS to break conditional symmetry

SBDS adds constraints to the local subtree. These constraints are discarded upon backtracking from the root node of a the subtree. However, this means that we must have an SBDS constraint for each *possibly* applicable symmetry. In the case of conditional symmetry, this is a particularly high overhead where, as in the example of all-interval series, there are many more conditional symmetries than unconditional ones.

An alternative is to check at a node whether or not a condition holds, and only to add the SBDS constraints in a local subtree where the condition is known to hold (Figure 6). Unfortunately, this approach fails. We might backtrack from this point and therefore discard the SBDS constraint, going back up the tree to a node where the condition is no longer true. Since the condition is not true, no conditional symmetry will be posted. Unfortunately, the condition could become true again on further backtracking and reassignment of variables. Thus, this approach is untenable because it will miss duplicate (non-)solutions (Figure 7). In order to solve this problem to use SBDS, we need to either post global constraints, or search the failed subtrees for conditional symmetries that *were* true. Even then, the constraints posted for breaking conditional symmetries would be of larger arity, since they contain the condition as well as the symmetry breaking constraints. Thus, we do not yet have a satisfactory approach for using a variant of SBDS as a generic method for breaking conditional symmetries.

5.2 Using SBDD to break conditional symmetries

In contrast to SBDS, SBDD should adapt very naturally to the conditional case. This is because the check is performed at a node about to be explored. At this point, we can calculate which conditional symmetries

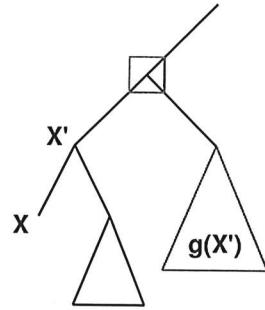


Figure 6: Upon backtracking to the highlighted node from X' SBDS posts a constraint to forbid $g(X')$ in the local subtree.

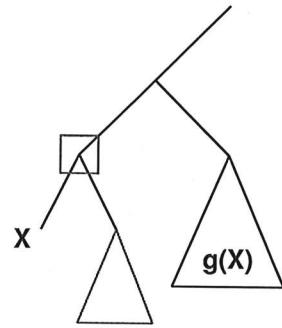


Figure 7: The symmetrical variant $g(X)$ of the nogood X , does not exist in the local subtree. SBDS can cope with such a situation by forbidding $g(X')$ (as shown in Figure 6). The **condition** that makes a bijective mapping h a symmetry, for example, may not hold at the node X' , hence the nogood $h(X')$ will not be ruled out.

are known to hold. We can then calculate the resulting group, and check this against previously visited nodes. Unlike SBDS, when we backtrack from a node, we do not need to know what conditional symmetry holds in some future node. We can maintain the database of nodes visited in the same way as conventional SBDD: that is, we need merely to record the nodes at the roots of fully explored search trees. At a search node of depth d there are at most d such roots to store.

In the case of conditional-SBDD, we need to check whether the current node is dominated by some previously visited node. That is, does some conditional symmetry hold which maps one of the roots of a failed tree into the current node? In the terms of this paper, does it map a previous node representing a CSP P into another CSP P' , such that the current node P'' is a sub-CSP of P' ? It might seem that we have to consider all nodes representing sub-CSP's of P , as different conditional symmetries can occur at different sub-CSP's, and perhaps one of these but not others will dominate P'' . This is the problem that bedevils extending SBDS for conditional symmetries. However, we can solve this apparent problem by a simple reversal: if a conditional symmetry maps a sub-CSP of P into a super-CSP of the

current node P'' , then its inverse must be a conditional symmetry mapping P'' into a sub-CSP of P .

We look at the question the opposite way round because we need to calculate the conditional symmetries that apply only at the current node. Specifically, we can calculate the symmetries *known* to apply at the current node. There may be sub-CSP's of the current node where more conditional symmetries apply, but we cannot deal with this. However, we do not see this as a major problem in general. If some subproblem P' of P maps to a superproblem of P'' , then in most cases the symmetric version of the condition that holds at P' will hold in the superproblem of P'' and so in P'' itself.

This reversal allows us to apply SBDD methods almost unchanged from current implementations. The new feature is that at any node where we check dominance, we have to calculate which conditions apply and therefore which symmetries to check. Having done this, and thus having perhaps a different group at each node, we can use any existing implementation technique for SBDD, adapting it as necessary to allow for a different group holding each time the dominance check is called.

For example, consider using computational group theory methods for SBDD following Gent, Harvey, Kelsey, and Linton [7]. The advantage for conditional SBDD is that we do not need to deal separately with all conditional symmetries. Instead, we need some method for testing the existence of generating symmetries. For example, consider the case of conditional compound order symmetry in the steel mill problem. At a point where we want to perform the SBDD check, it is very easy to examine the problem to see if the symmetry has formed. We just look at each slab to see if it has definitely been assigned two copies of the same order. While the *switch* variable could be used for this purpose, alternatively we could perform a simple check before performing the dominance check. Having done so, we then know which of these conditional symmetries hold. These can be expressed, as required by [7], as a permutation. The group of symmetries that hold is therefore as generated by the conventional symmetries, and the detected conditional symmetries. Passing each detected conditional symmetry as a permutation to the computational algebra system, automatically allows all combinations of symmetries – conditional and unconditional – to be used in the dominance check. The computational algebra system using essentially the same algorithm to check dominance as in the unconditional case [7]. In the example discussed, note that we did not need to have anything stored for conditional symmetries which do not arise. A simple program is written to see which symmetries hold, and for each one we only need to construct one generating permutation on the fly. This does not lead to large overheads compared to an unconditional SBDD implementation. On the algebraic side, the main inefficiency is in having to start with potentially a new group on each check, but it remains to be seen how significant a problem this is.

It thus seems that conditional SBDD has the poten-

tial to give a general and efficient way of dealing with conditional symmetries. We intend to implement our proposed conditional-SBDD in the near future. It thus remains to be seen if this approach can provide effective symmetry breaking with low enough overheads.

6 Conclusions and Future Work

This paper has discussed the phenomenon of conditional symmetry, and methods to exploit this symmetry to reduce search. The first, adding conditional symmetry-breaking constraints to a model, is most effective when the condition for the symmetry to arise is a) simple and therefore easy to check, and b) likely to be satisfied often during search. As with unconditional symmetry breaking, adding constraints to break one conditional symmetry can partially break another. Furthermore, choosing unconditional symmetry-breaking constraints carefully can simplify conditional symmetry-breaking in some cases, as was shown in both the Graceful Graphs and Steel Mill Slab Design problems studied herein. Formulating general rules to guide the modelling of a problem and the choice of symmetry-breaking constraints to take advantage of these insights is a considerable challenge, which we are beginning to explore.

The second method, reformulating the model to remove the conditional symmetry, can give a much improved model, but it is not clear that such a reformulation will always be possible or, if possible, achievable through general methods rather than special purpose reasoning. Finally, we discussed how a generic, dynamic method of breaking conditional symmetries might be constructed based on SBDD. A principal item of future work is in developing this method.

References

- [1] J. Crawford, M. L. Ginsberg, E. Luks, A. Roy. Symmetry-breaking Predicates for Search Problems. *Proc. 5th Int. Conf. on Principles of Knowledge Representation & Reasoning*, pp. 148–159, 1996.
- [2] T. Fahle, S. Schamberger, M. Sellmann. Symmetry Breaking. *Proc. 7th Int. Conf. on Principles & Practice of Constraint Programming*, 93–107, 2001.
- [3] A. M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, T. Walsh. Global Constraints for Lexicographic Orderings. *Proc. 8th Int. Conf. on Principles & Practice of Constraint Programming*, pp. 93–108, 2002.
- [4] A. M. Frisch, I. Miguel, T. Walsh. Symmetry and Implied Constraints in the Steel Mill Slab Design Problem. *Proc. CP'01 Workshop on Modelling & Problem Formulation*, pp. 8–15, 2001.
- [5] J. A. Gallian. Graph Labeling. *The Electronic Journal of Combinatorics*, Dynamic Surveys (DS6), www.combinatorics.org/Surveys, 2003.
- [6] I. P. Gent, W. Harvey, T. Kelsey. Groups and Constraints: Symmetry Breaking During Search. *Proc. 8th Int. Conf. on Principles & Practice of Constraint Programming*, 415–430, LNCS 2470, 2002.
- [7] I. P. Gent, W. Harvey, T. W. Kelsey, S. A. Linton. Generic SBDD Using Computational Group Theory. *Proc. 9th Int. Conf. on Principles & Practice of Constraint Programming*, pp. 333–347, 2003.
- [8] I. P. Gent, I. McDonald, B. M. Smith. Conditional Symmetry in the All-Interval Series Problem. *Proc. 3rd Int. Workshop on Symmetry in Constraint Satisfaction Problems*, 2003.
- [9] I. P. Gent, B. M. Smith. Symmetry Breaking in Constraint Programming. *Proc. 14th European Conf. on AI*, 599–603, 2000.
- [10] S. Prestwich, J. C. Beck. Exploiting Dominance in Three Symmetric Problems. *Proc. 4th Int. Workshop on Symmetry & Constraint Satisfaction Problems*, 2004.

On The Extraction of Disjunctive Landmarks from Planning Problems via Symmetry Reduction *

Peter Gregory & Stephen Cresswell

& Derek Long & Julie Porteous

University of Strathclyde

Glasgow, UK

{firstname.lastname}@cis.strath.ac.uk

Abstract

The exploitation of symmetry in combinatorial search problems has typically focussed on using information about symmetries to control search. This work describes an approach that exploits symmetry to get more detailed domain-analysis rather than as a method of search-control. We describe both the concepts of atomic and disjunctive landmarks in planning (a concept analogous to the backbone of a SAT/CP problem) and algorithms for extracting them. We categorise various types of symmetry encountered in planning, and show examples of each on actual problems. We introduce the Orlando API, a tool that extracts disjunctive landmarks using symmetry-breaking techniques and atomic landmarks analysis in concert. This provides a polynomial-time algorithm for extracting landmarks with respect to any equivalence function. We introduce the standard equivalence functions supplied with Orlando and show situations in which these are useful.

1 Introduction

Many different research communities have seen the attraction of exploiting symmetry in problems. Symmetry seems to be a characteristic of many combinatorial search problems and has been examined in the CP [Joslin and Roy, 1997; Gent and Smith, 2000], planning [Fox and Long, 1999; 2002] and model-checking [Ip and Dill, 1996; Clarke *et al.*, 1996; Emerson and Sistla, 1996] communities. Much effort has been spent in finding ways break symmetries during search. This work, however, details a method of gaining a rich domain-analysis by analysing *symmetrically reduced problems*. Symmetric reduction is performed by casting the properties of all the objects in a symmetric group on to a representative member of that group.

Landmarks are facts that occur in every valid solution to a planning problem [Porteous *et al.*, 2001]. Atomic landmarks are single literals. There are often no atomic landmarks in problems with symmetry in them. Disjunctive landmarks are disjunctions of literals that represent a set of facts, any of

which could occur in a valid plan [Porteous and Cresswell, 2002]. These disjunctions arise because of the symmetry in problems. If the problems are first symmetrically reduced then atomic landmarks can be extracted from the new problem and interpreted as disjunctive landmarks. This method is faster than other ways of extracting disjunctive landmarks, it also attaches more meaning to the landmarks we extract.

In Section 2 we will briefly introduce planning problems. We will discuss atomic landmarks, an algorithm for extracting them and give reasons for their weakness. We discuss how symmetry arises in planning problems, and how it renders atomic landmarks analysis useless. We then discuss disjunctive landmarks, along with an algorithm to extract them.

Section 3 discusses the Orlando API. The approach of extracting disjunctive landmarks using a combination of symmetry reduction and atomic landmarks analysis is detailed. The equivalence functions that are supplied with Orlando as standard are described. The use of landmarks analysis in several planners is discussed in Section 4. Also discussed are planners that make use of symmetry-breaking (although not in concert with landmarks analysis.)

2 Planning Problems

A planning problem is composed of a set of actions, an initial state and a goal formula. An action is itself composed of a condition formula that must be satisfied for successful application of the action, and an effect that modifies the state in some way. In today's planning formalisms [Fox and Long, 2003], it is possible to represent problems with a very rich structure. Problems with temporal structure, numeric resources and conditional effects amongst others are representable. The conditions of an action can be pre- or post-conditions to an action or can be an invariant that must hold over the duration of the action. The initial state is a conjunction of atomic predicates. The goal is a formula that, when satisfied, indicates a valid plan.

This paper, for reasons of clarity, will only consider a subset of this expressivity. We consider actions to have only pre-conditions (and not post-conditions and state-invariants). These pre-conditions will take the form of a conjunction of atomic predicates. Also, the effects are represented as two sets of atomic predicates; the add-list and the delete-list. When an action is applied, the facts in the add-list are added, and those in the delete-list are removed, from the current

*This work is part funded by EPSRC grant GR/S11015/01.

state. The goal formula is also assumed to be a conjunction of atomic predicates.

Predicate	Key
(at parcel location)	□○ ↗
(at truck location)	□○ ↘
(road loc1 loc2)	○ ↗
(in parcel truck)	□○ ↘

(a) The predicates.

:action	load
:parameters	?parcel ?truck ?location
:preconditions	(at ?parcel ?location) (at ?truck ?location)
:add-effects	(in ?parcel ?truck)
:del-effects	(at ?parcel ?location)
:action	unload
:parameters	?parcel ?truck ?location
:preconditions	(in ?parcel ?truck) (at ?truck ?location)
:add-effects	(at ?parcel ?location)
:del-effects	(in ?parcel ?truck)
:action	drive
:parameters	?truck ?loc1 ?loc2
:preconditions	(at ?truck ?loc1) (road ?truck ?location)
:add-effects	(at ?truck ?loc2)
:del-effects	(at ?truck ?loc1)

(b) The Action Schema

Table 1: The domain description of a logistics domain, with diagrammatic key, where appropriate

Table 1 describes a logistics domain used throughout this paper for explanatory purposes. It is intentionally simplistic so that clear examples of important concepts can be shown throughout the paper.

2.1 Atomic Landmarks Analysis

A *landmark* can be informally described as a fact that occurs in every valid solution to a problem [Porteous *et al.*, 2001]. The concept of landmarks in a planning problem is closely related to that of the *backbone* in the general CP/SAT problem. The main difference between the two is that the backbone is described as the set of fixed variables in the *optimal* solution [Slaney and Walsh, 2001]; landmarks are the facts that occur in *all* solutions.

There are algorithms in the literature [Porteous *et al.*, 2001; Zhu and Givan, 2003] for extracting atomic landmarks from a planning problem. The approach we use is described in 2.

Contrary to the method employed in [Porteous *et al.*, 2001] the algorithm does not need to reconstruct an RPG to test each literal. The concept of an RPG (*relaxed plan graph*) is described in [Hoffmann, 2003]. The important aspect of the RPG for this work is that it is constructed in polynomial time.

We associate a set of *ancestors* with each literal and each action. The ancestors of a fact or action are literals necessarily visited in reaching it. These are carried through the process of RPG construction. We obtain the necessary ancestors of an action at a given level by computing the union of the ancestors of its preconditions. We obtain the necessary ancestors of a fact at a given level by computing the intersection of the ancestors of its achieving actions. Trivially, the initial and goal states are landmarks. The algorithm is detailed in Figure 2.

Advance through Level until no change to action ancestors set

```
For each Literal reachable at Level
Ancestors(Literal,Level) :=  
    intersection of ancestors of  
    achievers of Literal  
    reachable at Level-1
```

```
For each Action reachable at Level
Ancestors(Action,Level) :=  
    Union of ancestors of  
    preconditions(Action)  
    reachable at Level
```

Output ancestors of the top-level goals as landmarks.

Figure 2: Algorithm for extracting atomic landmarks.

If the parcel in Figure 1(a) needs to be delivered to location G then atomic landmarks analysis would find the landmarks (at T1 S), (in P1 T1) and (at T1 G) other than the facts in the initial and goal state. These facts, along with their orderings, are almost enough to infer a plan.

Atomic landmarks analysis is weak in situations like Figure 1(b). Because each parcel could be delivered by either T1 or T2, the landmarks analysis counts neither (in P1 T1) nor (in P1 T2) as landmarks. The *symmetry* in the problem means that the algorithm is unable to find any landmarks.

2.2 Symmetries in Planning

The literature identifies several types of symmetry that arise in planning and CSP problems [Fox and Long, 1999; Long and Fox, 2003b; Joslin and Roy, 1997]. Techniques to break symmetry in search have been identified and implemented in several different planners [Fox and Long, 2002; Chen *et al.*, 2004; Edelkamp and Helmert, 2000]. General symmetry-breaking approaches have also been studied in the CP [Joslin and Roy, 1997; Gent and Smith, 2000] and model-checking communities [Ip and Dill, 1996; Clarke *et al.*, 1996; Emerson and Sistla, 1996]. The CP community has also considered static symmetry-breaking constraints. The CP lan-

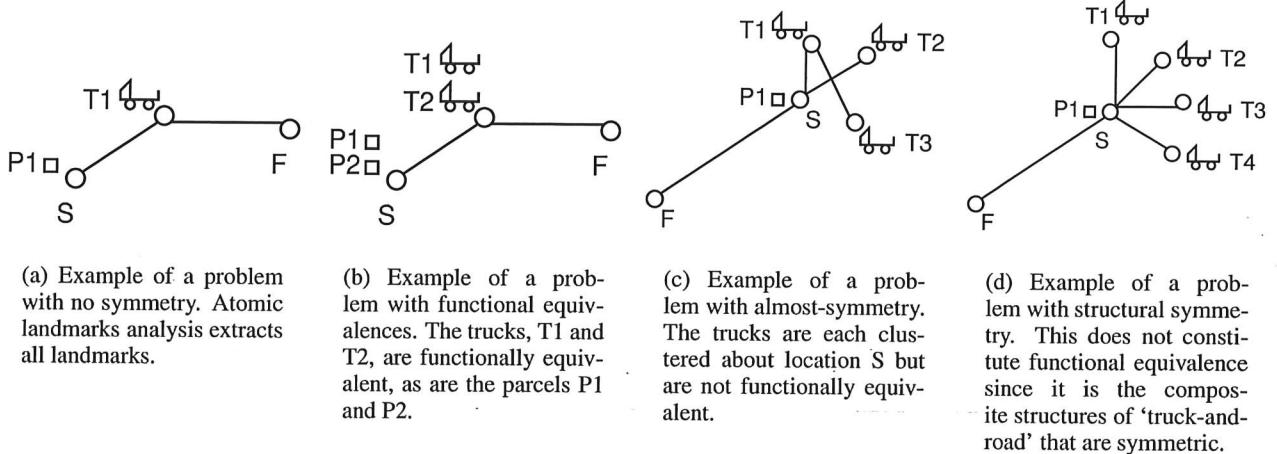


Figure 1: Pedagogical examples of problems from the simple logistics domain defined in Table 1. In all problems, each parcel is initially located at S; the goal is that each parcel is delivered to location F. Atomic landmarks analysis extracts all landmarks in (a) but none from the rest.

guage eclipse is packaged with a SBDS (symmetry-breaking during search) library and the model-checking tool Spin include a symmetry package ‘SymmSpin’.

We now describe four major categories of symmetries that arise in planning problems. Illustrations drawn from the simple logistics domain demonstrate an example of each:

Functional Equivalences: Two objects are functionally equivalent if they play exactly the same role in the initial (or current) and goal states of a planning problem and are not distinguished in any actions. In this case, it is only the names of the objects that distinguishes the objects and they can be freely permuted in any plan to yield a plan that achieves the same goals [Fox and Long, 1999]. Each of the trucks (parcels) in Figure 1(b) are functionally equivalent. In any valid plan, the roles of T1 and T2 could be reversed and the new plan would remain sound and would be equivalent to the original.

Functional identities are the most commonly exploited type of symmetry in planning. Typically, the only exploited identities are those which can be derived from the initial and goal states. [Fox and Long, 2002; Long and Fox, 2003b] used an innovative approach to keep track of the symmetric groups during search to take advantage of symmetries that arise during search.

Plan Permutation: Two plan fragments are symmetric when they achieve the same facts from the same state [Long and Fox, 2003b]. A successful plan for the problem in Figure 1(b) could include the plan fragment `(load P1 T1) ; (load P2 T1)`. It could also include the fragment `(load P2 T1) ; (load P1 T1)`, so these fragments are an example of plan permutation symmetry.

Any two non-interfering actions executed sequentially may be reversed. Any n non-interfering actions executed sequentially can be permuted in any of $n!$ ways. More generally, certain collections of actions will be equivalent under par-

ticular permutation operations (that is, symmetries) and will therefore play equivalent roles in any plans that contain them. As with other types of symmetry, exponential growth in the search-space is possible. Some preliminary results are presented in [Long and Fox, 2003a] for exploitation of this form of symmetry in a Graphplan planner (STAN).

Almost Symmetry: Symmetry arises as a consequence of abstraction in the representation of a problem. In many planning problems, potential symmetries are broken by elements of the problem description that are not sufficiently abstracted. In some cases, it is possible to apply an abstraction to a problem and thereby create opportunities for exploitation of symmetries that are exposed. The key to this approach is to find abstractions that lead to problems for which the solutions remain direct and relevant guides to the solutions of the original problems. A simple abstraction is to remove information from the domain. If the information is irrelevant to the problem then its removal will not prevent a solution from being found and the solution remains (trivially) a solution of the original problem.

A more sophisticated abstraction is to ignore certain initial conditions. An example of this idea can be applied to the example in Figure 1(c). The trucks clustered around the initial location of the parcel are *not* functionally equivalent, but since the goal location is so distant the small differences in location at the clustered side could be abstracted out. A plan prefix or postfix can then be applied to resolve the differences between the solution to the abstracted problem and the solution to the original problem.

There are many unresolved challenges in the exploitation of almost symmetry and it remains a current topic of research.

Structural Symmetry: The trucks in Figure 1(d) are equidistant from location S. They are not functionally equivalent because they are connected to S by different roads. It is clear, however, that the structure of each of the initial (Truck,

Location) pairs is symmetric with respect to any of any other initial (Truck, Location) pairs.

These higher-order structures can be found using NAUTY, a tool that finds graph automorphisms. Joslin and Roy [Joslin and Roy, 1997] have illustrated use of this approach. Donaldson et. al. have also used the approach in model-checking [Donaldson *et al.*, 2004].

Much symmetry in planning is clear to a person but is not as uniform as functional equivalence. This is exemplified in the concept of almost symmetry. If symmetry can be defined as a mapping of a structure onto itself then there must be methods of representing these types of abstractions as mappings.

Can these types of symmetry be generalised to CSP? Yes, many problems exhibit these forms of symmetry. Consider any problem where some ordered schedule has to be found; identical jobs can clearly be permuted, a stronger claim could be that any jobs can be permuted so long as the permutation doesn't affect the cost of the schedule. This exhibits plan permutation symmetry. Also, since none of the jobs need be identical, it exhibits almost-symmetry.

All types of symmetry prevent atomic landmarks analysis from performing well. One of the goals of our work is to extend earlier work on landmarks analysis to exploit information on symmetry.

2.3 Disjunctive Landmarks

Atomic landmarks are incapable of representing the landmarks that are present because of the symmetry in problems. Disjunctive landmarks analysis (referred to as 'resource abstracted landmarks' in [Porteous and Cresswell, 2002]) looks to remedy this situation by representing landmarks as disjunctions of facts. If the reader again refers to Figure 1(b) then he will recall that atomic landmarks analysis fails to recover any landmarks.

There are several recoverable disjunctive landmarks in this problem, three examples of which are given in Table 2.

1. $(in P1 T1) \vee (in P1 T2)$
2. $(in P2 T1) \vee (in P2 T2)$
3. $(in P1 T1) \vee (in P1 T2) \vee (in P2 T1) \vee (in P2 T2)$

Table 2: Disjunctive landmarks from problem in Figure 1(b).

The first two tell us that each of the parcels respectively are required to be in one of the trucks if the goal is to be achieved. The last is more general; it states that either of the parcels must be in either of the trucks. We now introduce an algorithm for extracting disjunctive landmarks based on the algorithm in Figure 2.

2.4 Disjunctive Landmarks via Forwards Propagation

In this section, we extend the forwards algorithm to deal with disjunctive landmarks. The part of the algorithm which changes is the computation of an ancestor set for a literal from

ancestor sets of achieving actions. When we consider landmarks to be simple literals, this computation is an intersection of the ancestor sets. A literal has to occur as an ancestor of every achieving action.

In the case of disjunctive landmarks, we have the option of selecting one condition from each achiever, and forming a disjunctive set. A problem that we now encounter is that there are very many possible disjunctive sets that could correctly be considered to be ancestors, and may turn out to be landmarks. We need some criteria by which to select which disjunctive sets are worth keeping.

Selecting disjunctions

An obvious criterion to use is to select literals which have the same predicate. However, we need to also impose further restrictions for this to work well.

We now give an example where this simple approach works poorly. Suppose we have a landmark which requires that some truck (truck1 or truck2) has a driver (driver1 or driver2). The disjunction of preconditions of achieving actions would include the following:

$$\begin{aligned} &(at(driver1, loc1) \wedge at(truck1, loc1)...) \vee \\ &(at(driver2, loc1) \wedge at(truck1, loc1)...) \vee \\ &(at(driver1, loc1) \wedge at(truck2, loc1)...) \vee \\ &(at(driver2, loc1) \wedge at(truck2, loc1)...) \end{aligned}$$

We would like to get two disjunctive landmarks, requiring that at least one truck and one driver are available.

$$\begin{aligned} &at(driver1, loc1) \vee at(driver2, loc1) \\ &at(truck1, loc1) \vee at(truck2, loc1) \end{aligned}$$

However, if we group according to the common predicate *at*, two disjunctive landmarks become merged into one:

$$\begin{aligned} &at(driver1, loc1) \vee \\ &at(driver2, loc1) \vee \\ &at(truck1, loc1) \vee \\ &at(truck2, loc1) \end{aligned}$$

Hence we need to be able to make a distinction between drivers and trucks which is not obvious from the context. A solution to this problem is to make use of the type data which is generated by analysis of the domain and problem using TIM [Fox and Long, 1998]. Fig. 3 shows part of a type hierarchy that is produced by TIM analysis. Given this information, when we attempt to generalise a disjunctive set, we do so by making the smallest available step towards the root of the type hierarchy. The categorisation of types in TIM makes use of a fingerprint for each object, which is the set of TIM spaces in which the object may participate. TIM spaces group together objects which are functionally equivalent. If we represent types as sets of space identifiers, then for types *Type1* and *Type2*, the subset relation $Type1 \subset Type2$ means that the type *Type2* is more specific than *Type1*, and so *Type2* is subtype of *Type1*. For present purposes, we also regard specific object identities as type information, and where appropriate we add those to the set. As a consequence, we can find a node in the type hierarchy that generalises two types simply by computing the intersection of their signatures.

We represent each ancestor as a pair $\langle Sig, DJset \rangle$, where *Sig* is a signature which specifies the predicate common to

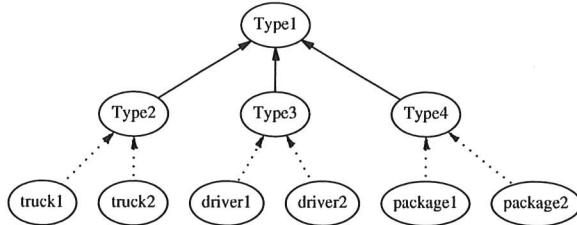


Figure 3: Example type hierarchy

the disjuncts and the types of the arguments common between disjuncts, and $DJset$ is a disjunctive set of literals.

The step in the forwards propagation algorithm in which the intersection of the ancestors of the achievers is computed is replaced by a procedure which uses signatures to derive generalisations where necessary. The generalisation of two pairs is defined as follows:

```

least_generalisation( < $Sig_0, DJset_0$ >, < $Sig_1, DJset_1$ >):
   $Sig_2 := intersect\_signatures(Sig_0, Sig_1)$ 
   $DJset_2 := DJset_0 \cup DJset_1$ 
  return < $Sig_2, DJset_2$ >

intersect_signatures(  $Sig_0, Sig_1$  ):
   $P :=$  predicate of  $Sig_0, Sig_1$ 
   $Sig_2$  is a new signature with predicate  $P$ 
  For each ( $N^{th}$ ) arg of  $Sig_0$ 
     $Arg(Sig_2, N) := Arg(Sig_0, N) \cap Arg(Sig_1, N)$ 
  return  $Sig_2$ 
  
```

We generate disjunctive landmarks such that each achieving action has at least one of the disjuncts as an ancestor. We keep only the most specific disjunctive sets — i.e. we discard any disjunctive set if its signature subsumes the signature of another disjunctive set.

Our algorithm based on these principles is effective at finding disjunctive landmarks in benchmark problems. However the space requirement is high, and under certain circumstances the generalisation algorithm could generate a number of disjunctive sets which is exponential in the number achievers for a literal. Hence the type information is not always sufficient on its own to select useful disjunctive landmarks.

2.5 Other Algorithms

There is another algorithm described in the literature for extracting disjunctive landmarks from a planning problem. [Porteous and Cresswell, 2002] introduce a different approach based on propagating an RPG backwards. No disjunctive landmarks analysis has yet been used in practical planning.

Here we introduce the Orlando API, a program that we have created to extract and manage disjunctive landmarks. Orlando does not use the algorithm from [Porteous and Cresswell, 2002] or the one in Section 2.4. Rather, the authors have discovered a novel way of combining symmetry-breaking techniques and atomic landmarks analysis to discover disjunctive landmarks.

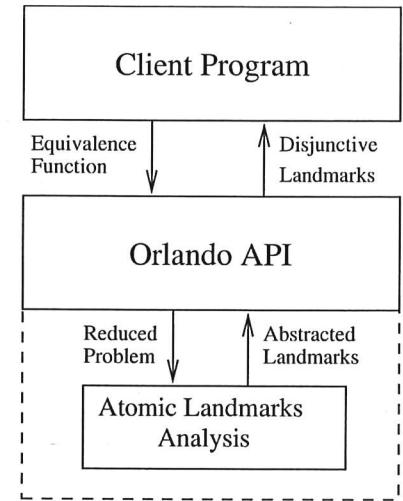


Figure 4: Architecture of Orlando.

3 The Orlando API

Orlando is a tool for extracting and managing disjunctive landmarks. By symmetric reduction, we can reveal disjunctive landmarks using only conventional landmarks analysis. The API takes a planning problem as input; the user can then request landmarks analysis be performed on several abstractions of the original problem. The landmarks obtained from these abstracted problems are equivalent to disjunctive landmarks.

Figure 4 shows the architecture of Orlando. A client program requests that landmarks analysis be performed with respect to some equivalence function. All of the equivalence functions are hard-coded into Orlando and are described below. Orlando reduces the problem and performs atomic landmarks analysis on this reduced problem. A translation of the results are returned to the client program. The Orlando algorithm can be described, in greater detail, in the following steps:

- Find the symmetry groups S of all the objects in the problem with respect to equivalence function F .
- For each symmetry group, choose a representative member object S_r . All occurrences of any member of group S in the problem should be replaced by S_r . This stage is the symmetric reduction stage. It is likely that all members of some symmetry group share common functionality. Therefore the new problem will be smaller, and less complex, than the old one; hence *symmetric reduction*.
- Perform conventional landmarks analysis using the algorithm in Figure 2. The landmarks obtained can now be reinterpreted into the original problem.
- Any atomic predicate including S_r is be expanded into a disjunctive landmark. The number of objects in S gives the number of disjuncts in the landmark. For all m in S , create a new disjunct identical to the atomic landmark replacing S_r with m .

This technique exploits symmetry so that we can generalise a problem. This is a novel use of symmetry. Symmetry has been exploited in planning to prune the search space, but here we are proposing to use symmetries to identify a way to structure the search space before exploring it. It is also a novel way of extracting disjunctive landmarks. Orlando depends on different equivalence functions, and so finds landmarks with respect to different symmetries. It can be said that we are gaining ‘structural analysis from symmetry’.

This is in juxtaposition to the current algorithms, which find disjunctive landmarks in one sweep; there is no indication of what *type* of symmetry the landmarks represent. This can be described as ‘symmetry from structural analysis’. These algorithms can suffer from the fact that they can generate meaningless landmarks [Porteous and Cresswell, 2002]. In contrast to this, the landmarks we generate have a very specific meaning relating to the equivalence function used to reduce the problem. This should aid a user in his decision-making process. Since the algorithm uses only traditional landmarks analysis then the time complexity is better than the older algorithms.

However, the landmarks we find are only ordered with respect to each equivalence function used. The older algorithms order all of the disjunctive landmarks that they find. Orlando requires the user to know something about symmetry, if only very little at this stage. There are good arguments for both types of disjunctive landmarks analysis and a forward-propagating algorithm such as that described in Section 2.4 is seriously considered for the next version of Orlando.

We will now examine the equivalence functions supplied with Orlando. We demonstrate, by example, what type of landmark these reveal. We also discuss the individual merits and shortfalls of each.

Functional equivalences: The functional equivalence functions come in two different flavours:

1. **With respect to all goals.** Often reducing a problem by functional equivalences recovers only a negligible number of new landmarks. Reducing the problem with respect to all of the goals has the problem that the landmarks that are found will not distinguish between goals and the resources associated with those goals.
2. **With respect to individual goals.** Interesting landmarks can be deduced by removing each of the goals, save one, and then reducing the problem by functional identities. This method typically finds more specific, and useful, landmarks than the previous one.

If we only consider one goal at a time, each parameter of that goal predicate is enforced as asymmetric to all objects of the same type. Thus, each object in that goal predicate will appear as an atom. Returning to Table 2, landmarks 1 and 2 are now found but 3 is not.

Relax complete types: Another way of abstracting the prob-

lem is to consider each object of the same type to be symmetric. This abstraction can counter the intuitive, but hard to define, almost-symmetries in a problem. In our logistics example Figure 1(c), for instance, relaxing the truck type would yield the landmark:

$$[(in P1 T1) \vee (in P1 T2) \vee (in P1 T3)]$$

This landmark is useful in the sense that it reveals the resource in the problem (i.e. the trucks). However, it does not give us any *specific* information about the problem. In this particular situation it is a useful landmark (all of the trucks are located close to each other).

It could easily be the case that there was a very distant truck in the problem that should never be considered as a candidate for delivering the parcel. This naïve relaxation would include the distant truck in the disjunction. This is clearly unattractive, but the abstraction is still useful when more specific abstractions fail to discover landmarks.

Reducing the static structure: Static predicates are those that cannot be modified. An example of a static predicate is the *road* predicate in the logistics domain. Since no actions can add or delete a *road* predicate, it is said to be *static*. Sometimes structural symmetry arises because of symmetry in the static structure of the problem.

If only the static structure of Figure 1(d) is considered then by functional identities, we can collapse the structure such that all of the trucks now occupy one location. After reintroducing the dynamic predicates, it is clear that the problem can be reduced further (as the trucks are all now at the same location in the abstraction). Note that two abstractions were performed here. One to remove the symmetry in the static structure, and then one to remove the symmetry which was then introduced.

We would now extract the following landmarks from the problem:

1. $(at T1 S) \vee (at T2 S) \vee (at T3 S) \vee (at T4 S)$
2. $(in P1 T1) \vee (in P1 T2) \vee (in P1 T3) \vee (in P1 T4)$
3. $(at T1 G) \vee (at T2 G) \vee (at T3 G) \vee (at T4 G)$

These landmarks are enough to solve this problem. Any sequence of actions that achieve these landmarks in order is a valid plan.

The previous example shows that the application of compound equivalence functions on a problem can collapse greater amounts of symmetry than one alone. We can imagine the abstractions spanning from the original problem in a hierarchy. At the top levels are those landmarks with the greatest specificity. As we go deeper into the tree, the extracted landmarks are necessarily more general.

4 Future and Related Work

The initial work on atomic landmarks in [Porteous *et al.*, 2001] has demonstrated that the performance of a planner can be improved with use of landmarks. There has remained moderate interest in the field of landmarks analysis.

The planner SGPlan[Chen *et al.*, 2004] uses landmarks (Chen et. al. refer to them as ‘hidden subgoals’, but their definition is the same). SGPlan decomposes the problem by finding goal-wise atomic landmarks, but not disjunctive landmarks. In the recent International Planning Competition it performed exceptionally, coming first and second place in the ‘Suboptimal Metric Temporal Track’ and the ‘Suboptimal Propositional Track’ respectively [EdelKamp and Hoffmann, 2004]. This shows that a planner that utilises landmarks can be successful. Zhu and Givan also entered a planner in the same competition, but performed less well [Zhu and Givan, 2004]. Their planner discriminates between different types of landmarks and then partially orders these to create what they call ‘roadmaps’. [Sebastia *et al.*, 2003] produced a planner that uses landmarks analysis to partition a planning problem so it can be solved in a parallel architecture.

Interesting future work could include creating a planner that uses disjunctive landmarks analysis to guide its search. Analysis of the equivalence functions in different situations will give a deeper understanding of how to automate such a process. Once the landmarks are found, one possible search strategy could be searching across the disjuncts. Also, we could proceed by trying to unify the disjuncts (intersect disjunctive landmarks where facts overlap.) Perhaps it would be better to make the landmarks distinct. Only further experimentation can answer these questions.

5 Conclusions

In this paper we have introduced Orlando, a domain-analysis tool for extracting disjunctive landmarks. This API uses symmetry-breaking techniques to enable disjunctive landmarks analysis to be performed at the cost of atomic landmarks analysis. Several equivalence functions are detailed, these are the standard equivalence functions supplied with Orlando. The functions that rely on functional identities are shown to have special plan-preserving properties. They are however quite brittle. Some structural symmetries can be broken by reducing the static structure of the problem. We have discussed weaknesses of the standard equivalence functions, there is a need for the user to be able to supply custom functions.

In real problems, few objects are functionally identical. The domain engineer knows in which situations objects should be considered symmetric, but sometimes cannot abstract that out when writing the domain. In this circumstance then supplying a custom equivalence function to Orlando should be possible. These functions must depend on structural aspects of a problem known by the user, such as the map in our examples. It seems imperative that, to write reusable functions, we should have to reason about generic types. It is indeed a goal of the authors to create a generic type specification language for this purpose.

The relationship between planning and CSP is very interesting. The comparison between landmarks in planning and the backbone in CSP problems is one that should be studied further. In [Slaney and Walsh, 2001] it is found that for one planning problem (the blocks-world domain) the relationship between size of backbone and difficulty of solving that prob-

lem was slightly negatively-correlated. This was in contradiction to how size of backbone was correlated with difficulty of solving several CSP problems (strongly positive correlation). It would be interesting to find out if this difference is global for all planning/CSP problems.

Landmark-extraction has been seen as a way of making planning easier. Much work has been carried out examining the relationship between the backbone of a CSP and the phase transition of the problem [Singer *et al.*, 2000; Slaney and Walsh, 2001; Culberson and Gent, 2000]. In [Dubois and Dequen, 2001], a heuristic search algorithm for solving SAT problems, based of backbone information, is introduced. Positive results in this paper suggest that heuristics based on backbone information should be studied elsewhere.

Acknowledgments

The authors would like to thank the EPSRC. This work is part funded by EPSRC grant GR/S11015/01. We would also like to thank Alex Coddington for the use of her table format.

The authors would like to thank the reviewers for their insightful input.

References

- [Chen *et al.*, 2004] Y. Chen, C. Hsu, and B.W. Wah. Sgplan: Subgoal partitioning and resolution in planning. International Planning Cometition Booklet, ICAPS 2004, June 2004.
- [Clarke *et al.*, 1996] E.M. Clarke, R. Enders, T. Filkhorn, and S. Jha. Exploiting symmetry in temporal logic model checking. *Formal Methods in System Design*, 9(1–2):77–104, 1996.
- [Culberson and Gent, 2000] J. Culberson and I.P. Gent. Frozen development in graph coloring. *Theoretical Computer Science*, 2000. In press.
- [Donaldson *et al.*, 2004] A. Donaldson, A. Miller, and A. Calder. SPIN-to-GRAPE: a tool for analysing symmetry in promela models. In *Proceedings of the 6th International AMAST Workshop on real-time systems (ARTS 2004)*, Stirling, Scotland, July 2004. To appear.
- [Dubois and Dequen, 2001] Olivier Dubois and Gilles Dequen. A backbone-search heuristic for efficient solving of hard 3-SAT formulae. In *Proceedings of Seventeenth International Joint Conference on Artificial Intelligence*, pages 248–253, 2001.
- [Edelkamp and Helmert, 2000] S. Edelkamp and M. Helmert. the implementation of mips, 2000.
- [EdelKamp and Hoffmann, 2004] S. EdelKamp and J. Hoffmann. Results of the fourth international planning competition. http://ls5-www.cs.uni-dortmund.de/~edelkamp/_ipc-4/, 2004.
- [Emerson and Sistla, 1996] E. Allen Emerson and A. Prasad Sistla. Symmetry and model checking. *Formal Methods in System Design*, 9(1–2):105–131, August 1996.
- [Fox and Long, 1998] M. Fox and D. Long. The automatic inference of state invariants in TIM. *Journal of AI Research*, 9:367–421, 1998.

- [Fox and Long, 1999] M. Fox and D. Long. The detection and exploitation of symmetry in planning problems. In *Proceedings of IJCAI'99*, pages 956–961, 1999.
- [Fox and Long, 2002] M. Fox and D. Long. Extending the exploitation of symmetries in planning. In *Proceedings of AIPS'02*, pages 83–91, 2002.
- [Fox and Long, 2003] M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.
- [Gent and Smith, 2000] Ian P. Gent and Barbara M. Smith. Symmetry Breaking in Constraint Programming. In Werner Horn, editor, *Proceedings ECAI 2000*, pages 599–603. IOS Press, 2000.
- [Hoffmann, 2003] J. Hoffmann. The metric-FF planning system: Translating “ignoring delete lists” to numeric state variables. *Journal of Artificial Intelligence Research*, 20:291–341, 2003.
- [Ip and Dill, 1996] C.Norris Ip and D. Dill. Better verification through symmetry. *Formal Methods in System Design*, 9:41–75, 1996.
- [Joslin and Roy, 1997] David Joslin and Amitabha Roy. Exploiting symmetry in lifted CSPs. In *AAAI/IAAI*, pages 197–202, 1997.
- [Long and Fox, 2003a] D. Long and M. Fox. Plan permutation symmetries as a source of planner inefficiency. In *Proceedings of the 22nd Workshop of the UK Planning and Scheduling Special Interest Group (PLANSIG '03)*, 2003.
- [Long and Fox, 2003b] D.P. Long and M. Fox. Symmetries in planning problems. In *Proceedings of SymCon'03: Third International Workshop on Symmetry in Constraint Satisfaction Problems*, 2003.
- [Porteous and Cresswell, 2002] J. Porteous and S. Cresswell. Extending landmarks analysis to reason about resources and repetition. In *Proceedings of the 21st Workshop of the UK Planning and Scheduling Special Interest Group (PLANSIG '02)*, 2002.
- [Porteous *et al.*, 2001] J. Porteous, L. Sebastia, and J. Hoffmann. On the extraction, ordering, and usage of landmarks in planning. In *Proceedings of European Conference on Planning*, 2001.
- [Sebastia *et al.*, 2003] L. Sebastia, E. Onaindia, and E. Marzal. Concurrent resolution of decomposed planning problems. In *Proceedings of the 22nd Workshop of the UK Planning and Scheduling Special Interest Group*, 2003.
- [Singer *et al.*, 2000] J. Singer, I.P. Gent, and A. Smaill. Backbone fragility and the local search cost peak. *Journal of Artificial Intelligence Research*, 12:235–270, 2000.
- [Slaney and Walsh, 2001] J. Slaney and T. Walsh. Backbones in optimization and approximation. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 254–259, 2001.
- [Zhu and Givan, 2003] Lin Zhu and Robert Givan. Landmark Extraction via Planning Graph Propagation. In *Printed Notes of ICAPS'03 Doctoral Consortium*, June 2003. Trento, Italy.
- [Zhu and Givan, 2004] L. Zhu and R. Givan. Heuristic planning via roadmap deduction. International Planning Competition Booklet, ICAPS 2004, June 2004.

A note on the compatibility of static symmetry breaking constraints and dynamic symmetry breaking methods

Warwick Harvey

IC-Parc, Imperial College London

Exhibition Road, London SW12 9RG, United Kingdom

wh@icparc.ic.ac.uk

Abstract

Adding static constraints (e.g. lexicographic constraints) and modifying a backtracking search procedure to dynamically eliminate the consideration of symmetrically equivalent states (e.g. SBDS and SBDD) are two common methods for breaking symmetry, with different advantages and disadvantages. It is natural to try to combine these in order to try to harness the strengths of each. However, in some cases solutions can be lost if the combination is not done carefully. We examine this problem, giving an example, and show how vulnerable versions of SBDS and SBDD can be modified to make them safe for combining in this way.

1 Introduction

Usually when solving a constraint programming problem which contains symmetry, we are only interested in finding at most one solution from any given equivalence class (doubly so if we are only looking for the first solution!). More than this, we would like to avoid exploring symmetrically equivalent search states; it is the avoidance of redundant search that allows symmetry-aware methods to solve symmetric problems more efficiently than other techniques.

In this paper we consider two popular symmetry-breaking approaches:

- Adding a set of static symmetry-breaking constraints: in particular, inequalities based on the *lex-leader* approach [3].
- Modifying the search to dynamically exclude symmetrical equivalents: in particular, the SBDS (“Symmetry Breaking During Search”) [1; 9] and SBDD (“Symmetry Breaking via Dominance Detection”) [4; 5] approaches (which we will collectively refer to as SBDx).

The standard lex-leader approach can only be applied to variable symmetry, and if the symmetry group is large it is not practical to break all symmetry (one constraint is required for each element of the group), but the addition of a suitable subset (or other simplification) of the lex-leader constraints can be a cheap and efficient way to break a lot of symmetry.

Classic SBDS [9] is also impractical for breaking all symmetry for large symmetry groups (one symmetry function is

required for each element of the group), but a modified version based on computational group theory [7] has been shown to be practical for performing complete symmetry breaking for groups of up to 10^9 elements.

SBDD-based approaches such as [10; 8] have been able to handle much larger groups, but the high cost of dominance checks for large groups has led some to consider sacrificing the completeness of the dominance check for improved overall runtime [4; 10; 2].

Given the efficient but incomplete lex-leader-based approach and the less efficient but complete SBDx approaches, it is natural to consider combining the two, using static constraints to prune much of the symmetric search space cheaply, but using an SBDx-based approach to make the symmetry breaking complete. Indeed, a combination of the two was used in [10], and something similar was used in [2] (with set variables rather than integer variables). However, concerns have been raised as to whether it is possible to combine the techniques without losing correctness: the lex-leader-based approach uses a static ordering on the solutions and only guarantees that the lexicographically least solution will be retained in any equivalence class, while the SBDx approaches usually use a dynamic ordering with the retention of solutions dependent upon the order in which they are found during the search. Any method which combines the two must make sure that they agree upon which solution to retain, or solutions may be lost.

In this paper we examine this issue of compatibility, showing how the obvious but naïve combination can go wrong in some circumstances, identifying which of the SBDx variants might have a problem, and showing how to modify them to make them safe. The simple combination of techniques presented here is not claimed to be a great way to break symmetry, but is merely intended to show that it can be done; the main contribution of the paper is to point out some of the problems that arise when combining these techniques and how to overcome them, as a foundation for further work in this area.

The rest of the paper is organised as follows. In Section 2 we briefly review the important features of the techniques we will be combining. This is followed in Section 3 by the presentation of a straightforward way to combine the techniques, a discussion of the problems that can arise, and how they can be overcome. Finally, in Section 4 we present some experi-

ments backing up the main results of the paper.

2 Review of Relevant Symmetry Breaking Techniques

In this section we briefly review the key features of the relevant symmetry-breaking techniques. The reader is referred to the original papers for more detail if desired.

2.1 Lex-Leader Symmetry Breaking

This approach [3] derives from the observation that one can break symmetry by constraining the problem state to be the smallest (or largest) among all the symmetrically equivalent states according to some suitable measure. Complete symmetry breaking can be achieved if the measure yields different values for two solutions whenever those solutions are not identical. The standard way to do this is to arrange the problem variables into a vector, and constrain this vector to be lexicographically less than or equal to (some approaches use greater than or equal to) every symmetrical variant of the vector (hence making sure it is the *lex leader*).

Since one constraint is needed for each element of the symmetry group in order to ensure complete symmetry breaking, it is not practical to do this for anything but relatively small symmetry groups. However, partial symmetry breaking can be achieved by using just a subset of the constraints (or some other suitable weakening, such as truncation), and many popular sets of symmetry-breaking constraints can be cast in this framework (e.g. “double lex” for matrix models with full row and column symmetry, where each row is constrained to be lexicographically less than or equal to the next, and each column is constrained to be lexicographically less than or equal to the next).

One common way to decide how to choose a suitable subset to use is to take into account the other constraints of the problem. For example, if a set of symmetry-breaking lexicographic constraints have a common prefix $\langle x_1, \dots, x_k \rangle \leq_{\text{lex}} \langle y_1, \dots, y_k \rangle$ and the problem constraints imply that $\bigvee_{i=1}^k x_i \neq y_i$ (i.e. the vectors cannot be identical over this prefix), then we can replace the entire set of constraints by $\langle x_1, \dots, x_k \rangle <_{\text{lex}} \langle y_1, \dots, y_k \rangle$. In particular, if the problem constraints imply that $x_1 \neq y_1$ then the constraints reduce to simply $x_1 < y_1$. Note that this actually *strengthens* the constraints: due to the inequality now being strict, the resulting constraints can propagate more strongly than the originals.

2.2 SBDS

SBDS [1; 9] is a dynamic symmetry breaking technique which works by modifying a standard backtracking search procedure. The idea is simply that whenever a subtree has been completely explored, constraints are added which exclude everything which is symmetrically equivalent to that subtree. Suppose that in the context of a partial assignment A , the assignment $Var = Val$ was tried and is now being backtracked (either because there was no solution, or because we are looking for more solutions). At this point, SBDS in effect imposes the constraint $g(A) \Rightarrow g(Var \neq Val)$ for each

symmetry g of the problem. This ensures that if a state symmetrically equivalent to (or implied by) A is reached, we do not try the equivalent of $Var = Val$.

Note that it is not necessary for A to include all variables which are ground when representing the partial assignment: the ones set by (positive) decisions during the search is sufficient, and this is what most (if not all) implementations use, for efficiency reasons.

2.3 SBDD

SBDD [4; 5] is also a dynamic symmetry breaking technique that works by modifying a standard backtracking search procedure, and is quite closely related to SBDS. Whenever a subtree has been completely explored, a *no-good* is recorded. At every node of the search tree, a *dominance check* is performed, to see whether the current state is dominated by any symmetric variant of a recorded no-good. If it is, then we have already explored a state equivalent to the current one, and may backtrack.

Early versions of SBDD recorded more-or-less the entire state for the no-good and compared these to the current entire state; later versions (e.g. [10; 8]) typically only record the positive decisions made during the search, for comparison with the current entire state, since this significantly improves the efficiency of the dominance checks. Note that the current state is always represented in full in the dominance check, since using just the decisions can lead to incompleteness: in general it is possible to reach the same state via more than one different (and not symmetrically equivalent) set of decisions, and thus a dominance check might fail to detect dominance when it should.

In [4], a variant of SBDD was described where a static solution order is used instead of recording no-goods. In this variant the current state is checked for dominance by any state which comes before it in the ordering. As far as we are aware, this approach has never been implemented.

3 Combining Lex-Leader and SBDx

The real problem with just applying both these techniques without consideration for their compatibility is of course that adding the static symmetry-breaking constraints changes the problem, so that it no longer has all of the symmetries that SBDx is being told it has. The question is whether SBDx can be made to work despite this and continue to retain exactly one solution from each equivalence class of the original problem.

3.1 False Start?

Initially we thought that compatibility could be ensured simply by using the right variable and value order in the search in conjunction with standard SBDx methods. Specifically, the variable order should be the same as that used in deriving the lex-leader-based constraints (most significant variable first), and the value order should be smallest to largest (or whatever the lexicographic constraints use). Applying SBDx in the absence of the static constraints, this would mean that the search would find the lexicographically least solution in any equivalence class first, which means that it is these solutions

that SBDx retains, while the lexicographically greater ones are discarded. Since both approaches when applied individually would keep the lexicographically least solution in each class, it would seem natural for the combination to also keep these solutions.

Here is the proof:

Theorem 1 *Given a static variable and value ordering and a static set of symmetry-breaking constraints which preserve the lexicographically least solution according to this ordering in any equivalence class (e.g. lex-leader constraints or simplifications thereof), one can use SBDS or SBDD symmetry breaking and not lose any (unique) solutions if one uses that variable and value ordering during the search.*

Proof: Consider the complete search tree without any symmetry breaking. Note that the solutions are found in lexicographic order (per the variable/value ordering used) in this tree; in particular, the solutions in any given equivalence class are found in lexicographic order. Note also that the effect on the search tree of adding the static constraints is to prune some subtrees, but if any pruned subtree contained any solutions, these solutions are necessarily not the lexicographically least in their respective equivalence classes.

Thus it remains to show that any solution pruned by SBDx is also not the lexicographically least. This is straightforward. Due to the variable/value search order, the subtree corresponding to an SBDx no-good is lexicographically smaller than the parts of the search tree in which it is applied (since these necessarily come later in the search). Thus if any solution θ to the original problem is pruned by an SBDx no-good, then we know that there is another, symmetrically equivalent solution θ' in the subtree corresponding to the no-good (if not, the no-good could not have pruned θ), and that θ' is necessarily lexicographically smaller than θ . Thus any solution pruned by the no-good is guaranteed to not be the lexicographically least.

Thus, since neither the lexicographic constraints nor the operation of the SBDx method removes the lexicographically least solution in each equivalence class, we are guaranteed to find at least this solution from each class and the result holds. \square

Unfortunately, the theorem is false — or at least misleading without some provisos and caveats. Barbara Smith pointed out¹ that if the symmetry-breaking constraints are strong enough to cause a variable to become fixed and this is not taken into account in some way, solutions can indeed be lost.

3.2 Example: latin squares

Consider the problem of enumerating all possible latin squares, considering only the row and column symmetry, and assume that we are modelling the square as an $n \times n$ array of variables, each taking a value from $\{1 \dots n\}$ representing the colour of that cell (see Figure 1). The problem constraints are then simply that within each row or column, all the entries are different.

$$\begin{array}{cccc} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nn} \end{array}$$

Figure 1: Latin square model

$$\begin{array}{cccc} 1 & 2 & \cdots & n \\ 2 & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & & \vdots \\ n & x_{n2} & \cdots & x_{nn} \end{array}$$

Figure 2: Latin square after symmetry-breaking constraint propagation

Suppose that the variable ordering we are using is row-wise ($x_{11}, x_{12}, \dots, x_{1n}, x_{21}, x_{22}, \dots, x_{2n}, \dots$) and the value order is smallest first. Then the lex-leader symmetry-breaking constraints derived from this ordering can be simplified (losing completeness) down to double lex. (Alternative way of looking at it: double lex is compatible with this ordering because these constraints will never exclude the lexicographically least solution in any equivalence class.)

Note that by taking into account the problem constraints, the lexicographic constraints may be strengthened. Consider for example the row constraint $\langle x_{11}, \dots, x_{1n} \rangle \leq_{lex} \langle x_{21}, \dots, x_{2n} \rangle$. We know that x_{11} cannot be equal to x_{21} , and thus this constraint may be replaced by the simple inequality $x_{11} < x_{21}$. Doing this for all the other lexicographic constraints results in a set of simple inequalities which, when propagated, will set the first row and the first column to $\langle 1, 2, \dots, n \rangle$ (see Figure 2).

Thus when we start searching, the first variable to be labelled is x_{22} . It is first assigned the value 1; call the resulting state \mathcal{A} . On backtracking we impose $x_{22} \neq 1$; call the resulting state \mathcal{B} .

First consider SBDD. SBDD requires that we check whether \mathcal{B} is dominated by \mathcal{A} before we proceed. If we compare whole states, we see that there is no dominance: the first row and first column must stay where they are. However, if we try to map just the decisions made to reach state \mathcal{A} (i.e. $x_{22} = 1$) into the state \mathcal{B} then we can easily achieve dominance by exchanging the first two rows and the first two columns. Thus a standard decision-based SBDD implementation would prune this branch; if it contained any lex-leader solutions, they would be erroneously discarded.

With SBDS, on backtracking the $x_{22} = 1$ assignment we (effectively) impose $g(\mathcal{A}) \Rightarrow g(x_{22} \neq 1)$ for each symmetry g . Since this is the first decision, with standard SBDS \mathcal{A} is empty, and so we are imposing $g(x_{22} \neq 1)$ for all symmetries g . One such g exchanges the first two rows and the first two

¹Personal communication.

columns, so we add the constraint $x_{11} \neq 1$, which obviously causes immediate failure. Again, any lex-leader solutions in this branch are erroneously discarded.

3.3 Where is the problem?

The essence of the problem is that for SBDS and decision-based SBDD dominance checks, in the presence of extra symmetry-breaking constraints, the set of decisions taken to reach a particular search state is no longer a sufficient characterisation of that search state. In the pure SBDx version of the problem, using just the decisions works because any difference between the current search state and the initial configuration is entailed by the set of decisions made to reach that state, and so any other search state which entails a symmetric version of the decisions necessarily implies the symmetric equivalent of the search state derived from those decisions. Thus one can identify a symmetric search state (in order to exclude it) by just looking for the symmetric set of decisions, which is more efficient than comparing the entire state.

In the presence of static symmetry-breaking constraints, however, a symmetrical set of decisions may not lead to a symmetrical search state. In such a case the set of decisions is no longer a sufficient characterisation of a search state in order to identify a symmetrical variant of it. Since SBDS and decision-based variants of SBDD assume that it is, these techniques can therefore go wrong if it is not.

Obviously, variants of the SBDD algorithm where the dominance check is implemented using the entire search state (e.g. [4; 5]) do not have such a problem. For these methods, as long as the search chooses to label the first non-ground variable in the ordering each time, it should work fine. Similarly, if one is using an SBDD variant with a static solution ordering (as mentioned in [4]) then as long as one is using the same lexicographical ordering for this as for the static constraints then the combination will work regardless of the search order.

This naturally leads to the question of whether something less than the entire search state is sufficient characterisation in the presence of static symmetry-breaking constraints. This seems particularly important if one wishes to adapt SBDS for use in combination with static constraints, since all current SBDS algorithms are inherently decision-based.

3.4 Doing better than comparing whole search states

It turns out to be relatively straightforward to adapt SBDS and decision-based SBDD variants to work correctly without taking into account the entire search state. These decision-based methods, when labelling a sequence of variables, usually just skip over any variable that is ground, since there is no decision to make. In the presence of static symmetry-breaking constraints, however, some variables may be ground when they would not have been otherwise, and so some “decisions” are skipped that would have been included in the absence of the static constraints. If these decisions are re-included, then the combination of static and dynamic techniques ought to work fine.

Of course, it is not immediately obvious which of the ground variables encountered during labelling have been set

as a consequence of the static constraints, so we propose simply including all such variables in the decision set used by these techniques. We now show that this modification means that decision-based SBDx techniques will never exclude solutions that are minimal with respect to the lexicographic ordering, regardless of which variables may have been set by (lex-leader compatible) static symmetry-breaking constraints, thus making the combination safe.

Theorem 2 *Given a static variable and value ordering and a static set of symmetry-breaking constraints which preserve the lexicographically least solution according to this ordering in any equivalence class (e.g. lex-leader constraints or simplifications thereof), one can use SBDS or decision-based SBDD symmetry breaking and not lose any (unique) solutions if one uses that variable and value ordering during the search and treats any ground variables encountered as a decision to assign that value to that variable.*

Proof: Let the variable order be $\langle x_1, \dots, x_n \rangle$. We assume w.l.o.g. that the value order is smallest to largest.

Consider a subtree of the search. This subtree is reached by making a sequence of (positive) decisions $\langle x_1 = a_1, \dots, x_j = a_j \rangle$ for some j and some a_i . Upon completing the exploration of this subtree, this set of decisions is treated as a no-good with all symmetrical equivalents excluded from consideration in subsequent parts of the search tree, either by explicit constraints (SBDS) or dominance checks (decision-based SBDD).

Due to the variable and value ordering used, subsequent search states necessarily have a corresponding sequence positive decisions $\langle x_1 = b_1, \dots, x_k = b_k \rangle$ for some k and some b_i such that $\langle a_1, \dots, a_j \rangle <_{lex} \langle b_1, \dots, b_k \rangle$. If a previously recorded no-good mapped by some symmetry g is entailed by the current search state, SBDx prunes the current state.² This pruning is valid if the current state cannot be extended to a solution which is lexicographically least in its equivalence class. Suppose to the contrary that there exists such a lexicographically least solution θ . Since θ is a refinement of the current state, we know that $\theta \models x_i = b_i$ for $1 \leq i \leq k$. Now consider the solution $\theta' = g^{-1}(\theta)$, i.e. θ mapped by the inverse of the symmetry that embeds the no-good in the current state. Then we have that $\theta' \models x_i = a_i$ for $1 \leq i \leq j$. But this means that θ' is lexicographically smaller than θ , contradicting our assumption that θ was lexicographically least in its equivalence class. Hence pruning the current state is valid.

Since the above argument applies for any no-good and makes no assumptions about the presence or absence of any static symmetry-breaking constraints, we have that using the stated labelling approach with SBDx can never exclude the lexicographically least solution in any equivalence class even if static symmetry-breaking constraints are added. Since the static constraints also preserve these lexicographically least solutions, the result follows. \square

For particular problems, one can no doubt do better than including all ground variables encountered during labelling

²We are ignoring here the fact that the no-good can be used to filter domain values as well as causing branches to fail, but the same argument applies.

in the decision set, by exploiting problem-specific information. For instance, if one has a set of variables that must be assigned distinct values from a set of the same cardinality (e.g. $x_1, \dots, x_n \in \{1, \dots, n\}$, $\text{alldifferent}(x_1, \dots, x_n)$), then there is no need to include the last one encountered in the decision set: the other variable-value pairs already included sufficiently represent the state. This kind of optimisation is probably more important for boolean models: if a set of variables must sum to 1, then the only “interesting” assignment is the one that sets some variable to 1; if there is such an assignment (i.e. the set of variables are ground) then including all the 0s in the SBDS partial assignment or an SBDD no-good is a waste of time and space.

3.5 Did it really need fixing?

In the latin square example of Section 3.2, we used the problem constraints to strengthen the static symmetry-breaking constraints, and this was key in demonstrating the loss of solutions. We conjecture that if one uses just natural (unstrengthened) lex-leader constraints, then using standard (unmodified) decision-based SBDx is safe, but this remains to be proven.

3.6 Related work

For SBDS, the extra conditions that appear in the dynamically added constraints as a result of including ground variables in the partial assignment A are similar to the extra conditions added to such constraints when adapting SBDS for *partial symmetries* [12]. However, with a partial symmetry the extra condition c_s added to the constraint is invariant with respect to the symmetry, which is not the case here. Also, the extra condition that ends up included in our approach depends on where in the search tree the constraint is added, whereas with partial symmetries the condition depends only on the (partial) symmetry.

4 Experiments

As an empirical validation of the above results we ran some experiments on the latin square problem, combining lexicographic constraints with SBDS and SBDD implementations. The symmetry-breaking constraint choices tried were:

- **none:** No symmetry-breaking constraints.
- **weak:** Standard double lex constraints.
- **strong:** Strengthened form of the lexicographic constraints; i.e. $<$ constraints between the elements of the first row and between the elements of the first column.

The SBDx choices tried were:

- **none:** Standard search (no SBDS or SBDD).
- **SBDS:** SBDS (the GAP-ECLⁱPS^e version described in [7]).
- **SBDS no-skip:** A modified SBDS search that treats ground variables encountered in the variable order as if they were decisions.
- **SBDD:** Decision-based SBDD (the GAP-ECLⁱPS^e version described in [8]).

- **SBDD no-skip:** A modified SBDD search that treats ground variables encountered in the variable order as if they were decisions.

In each case the labelling order was row-wise.

The experiments were performed using ECLⁱPS^e 5.7 #57 [11] and GAP 4.3.fix5 [6] on a 933MHz Intel Pentium III machine.³

The results of enumerating all 5×5 latin squares (number of solutions found and number of seconds taken) appears in Table 1. The corresponding data for the 6×6 instance are shown in Table 2.

Clearly there are 16 unique solutions to the 5×5 instance, and 1868 for the 6×6 . As expected, the lexicographic constraints are not sufficient to break all symmetry. Also as expected, combining the strong lexicographic constraints with (decision-based) SBDx algorithms results in solutions being lost unless those algorithms are modified appropriately. The results for the weak lexicographic constraints suggest that there is not a problem for this combination — at least on this problem, with only local consistency applied to the lexicographic constraints on an individual basis.

As can be seen, treating all ground variables encountered as though they were decisions does incur an efficiency overhead when no static constraints are added. For SBDD, the benefit that the static constraints bring in reducing the number of dominance checks required more than offsets this extra overhead. On the other hand, SBDS does not seem to particularly benefit from the addition of the static constraints. We have not analysed the reasons for this, but a plausible explanation is that SBDS has already done most (if not all) of the work it would do anyway by the time the static constraints prune the search, in which case there is nothing to gain from adding the static constraints. It is also possible though that we have just done a poor implementation of it, and that better can be done.

5 Conclusions

We have shown that for some combinations of lex-leader-derived static constraints and decision-based SBDx implementations, solutions can be lost. We have also shown how to modify these SBDx implementations in order to make them safe for combining with such constraints. It is still an open question as to exactly when there is a problem; until this is resolved we suggest that any attempt to combine these two approaches should either use a safe variant described here or carefully justify the correctness of the combination.

Acknowledgments

We would like to thank in particular Barbara Smith, Karen Petrie, Ian Gent, Tom Kelsey and Andrew Sadler for interesting and helpful discussions related to this work. We also thank the anonymous reviewers for their helpful comments. This work was supported by EPSRC grant GR/S30658/01.

³The times reported are the result of just a single run and so should be taken with a grain of salt — obtaining accurate timings was not the point of the exercise.

SBDx		none		SBDS		SBDS no-skip		SBDD		SBDD no-skip	
lex		NSols	time	NSols	time	NSols	time	NSols	time	NSols	time
none	161280	191.57		16	1.33	16	2.01	16	1.71	16	1.95
weak	56	0.17		16	1.30	16	1.86	16	1.19	16	1.39
strong	56	0.08		1	0.40	16	1.73	1	0.41	16	1.05

Table 1: 5×5 latin square enumeration: number of solutions and time taken

SBDx		none		SBDS		SBDS no-skip		SBDD		SBDD no-skip	
lex		NSols	time	NSols	time	NSols	time	NSols	time	NSols	time
none	-	-		1868	132.93	1868	232.11	1868	176.15	1868	243.56
weak	9408	26.34		1868	118.01	1868	217.39	1868	138.62	1868	177.01
strong	9408	10.03		3	0.73	1868	186.42	3	0.71	1868	135.93

Table 2: 6×6 latin square enumeration: number of solutions and time taken

References

- [1] R. Backofen and S. Will. Excluding symmetries in constraint-based search. In Joxan Jaffar, editor, *CP'99: Proceedings of the 5th International Conference on Principles and Practice of Constraint Programming*, pages 73–87. Springer, 1999. LNCS 1713.
- [2] Nicolas Barnier and Pascal Brisset. Solving the Kirkman's Schoolgirl Problem in a few seconds. In Pascal Van Hentenryck, editor, *CP 2002: Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming*, volume 2470 of *Lecture Notes in Computer Science*, pages 477–491. Springer-Verlag, 2002.
- [3] J. Crawford, M. Ginsberg, E. Luks, and A. Roy. Symmetry breaking predicates for search problems. In *Proc. KR 96*, pages 148–159. Morgan Kaufmann, 1996.
- [4] Torsten Fahle, Stefan Schamberger, and Meinolf Sellmann. Symmetry breaking. In T. Walsh, editor, *CP 2001: Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming*, pages 93–107, 2001.
- [5] Filippo Focacci and Michaela Milano. Global cut framework for removing symmetries. In T. Walsh, editor, *CP 2001: Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming*, pages 77–92, 2001.
- [6] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.3*, 2002. (<http://www.gap-system.org>).
- [7] Ian P. Gent, Warwick Harvey, and Tom Kelsey. Groups and constraints: Symmetry breaking during search. In Pascal Van Hentenryck, editor, *CP 2002: Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming*, volume 2470 of *Lecture Notes in Computer Science*, pages 415–430. Springer-Verlag, 2002.
- [8] Ian P. Gent, Warwick Harvey, Tom Kelsey, and Steve Linton. Generic SBDD using computational group theory. In Francesca Rossi, editor, *CP 2003: Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming*, pages 333–347. Springer-Verlag, 2003. LNCS 2833.
- [9] I.P. Gent and B.M. Smith. Symmetry breaking in constraint programming. In W. Horn, editor, *Proceedings of ECAI-2000*, pages 599–603. IOS Press, 2000.
- [10] Jean-François Puget. Symmetry breaking revisited. In Pascal Van Hentenryck, editor, *CP 2002: Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming*, volume 2470 of *Lecture Notes in Computer Science*, pages 446–461. Springer-Verlag, 2002.
- [11] Mark G. Wallace, Stefano Novello, and Joachim Schimpf. ECLiPSe : A platform for constraint logic programming. *ICL Systems Journal*, 12(1):159–200, May 1997.
- [12] Sebastian Will and Rolf Backofen. Breaking of partial symmetries in the photo and alignment problem. In Barbara M. Smith, Ian P. Gent, and Warwick Harvey, editors, *Proceedings of SymCon'03: The Third International Workshop on Symmetry in Constraint Satisfaction Problems*, pages 187–194, 2003.

Breaking Weak Symmetries¹

Roland Martin

Darmstadt University of Technology
Algorithms Group
64283 Darmstadt, Germany,
martin@algo.informatik.tu-darmstadt.de

Karsten Weihe

The University of Newcastle
School of Electrical Engineering and Computer Science
Callaghan, NSW 2308 Australia
weihe@cs.newcastle.edu.au

Abstract

In this paper we consider a generalisation of symmetry, which we call *weak symmetry*. A weak symmetry acts only on a subset of the variables and preserves the feasibility state only with respect to a subset of the constraints.

We will consider *weakly decomposable problems* that is, a problem decomposes in two subproblems whereby the whole problem contains weak symmetries. That means that one of the subproblems is proper symmetrical.

We discuss a remodelling concept where we use additional variables which we call *SymVar* (*Symmetry Variable*). These variables enable us to exploit weak symmetries and achieve symmetry breaking on the symmetrical variables of the problem without losing solutions.

Roughly speaking we rearrange the search tree in a way such that all symmetrical variable assignments are arranged under a specific node. This node represents a complete SymVar assignment of the symmetry class. Therefore the symmetrical variable assignments form a “frontier line” in the search tree where each node corresponds exactly to one symmetrical variable assignment of the SymVars.

1 Introduction

Symmetries transform a (partial) solution in a symmetrical (partial) solution and preserve the state of feasibility: no-goods are transformed in symmetrical no-goods while feasible solutions are transformed in symmetrical feasible solutions. Therefore symmetries decompose the search space in classes of symmetrical solutions whereby each class contains either only feasible or infeasible solutions.

When searching a solution to a problem it is sufficient to find only one solution in each class of solutions. If more than one solution is requested the symmetrical equivalents to the found solution can be derived by applying the symmetry function exhaustively. Therefore symmetries should be excluded from the search space to speed up the search.

Various techniques have been proposed for symmetry handling. In general it is done by remodelling the model, excluding the symmetry up-front via constraints, breaking it during the search or by a combination thereof.

Consider now a scenario where the problem itself includes some kind of symmetry that acts only on a subset of the variables and only with respect to a subset of the constraints. They cannot be excluded or broken by the standard methods mentioned above without losing solutions. This is due to the fact that symmetrical variable assignments for the variables affected by this kind of symmetry yield different value assignments for the asymmetrical variables, such that the crucial property of symmetry – full equivalence of symmetrical solutions – is lost.

A good idea would be to decompose the problem in two parts and perform symmetry breaking on the symmetrical subproblem. This implies to solve them successively and use the solutions of the first model as input to the second one. But solving the first model exhaustively is usually too expensive.

SymVars address this problem. SymVars are additional variables which represent the symmetrical solutions of the problem. The symmetry function is then applied via the assignment of the SymVars. In case the symmetry function is a permutation, for example, an assignment of the SymVars represents a permutation of the variables of the symmetrical subproblem.

Using SymVars seems to be particularly useful for tight-fit problems and optimisation within a given time limit (that is just a small amount or a fraction of the time it would take to search the problem exhaustively).

In Section 2 we introduce weak symmetries and state in Section 3 how weak symmetries influence the search tree. Section 4 comprises some variations of standard problems that include weak symmetries. In Section 5 we discuss the different ways of handling weak symmetries and introduce in Section 6 the new modelling concept of SymVars and how it could be used to break weak symmetries. In Section 7 we give examples that use the SymVar concept and in Section 8 we give an outlook to further work with weak symmetries and SymVars.

¹In cooperation with Philips/Assembléon,Netherlands

2 Weak Symmetry Description

2.1 Weak Symmetry Definition

To characterise weak symmetries we first have to define a weakly decomposable problem.

Definition 1 (Weakly Decomposable Problem)

Given a problem description $P = (X, C)$.

$X = \{x_1, \dots, x_n\}$ is the set of variables and

$C = \{c_1, \dots, c_m\}$ is the set of the constraints. The constraints c_1, \dots, c_k are of the form $c(x_1, \dots, x_\ell)$ and the constraints c_{k+1}, \dots, c_m are of the form $c(x_1, \dots, x_n)$.

P is weakly decomposable if it splits in two subproblems P_1 and P_2 .

$P_1 = (X_1, C_1)$ consists of the variables $X_1 = \{x_1, \dots, x_\ell\}$ and the constraints $C_1 = \{c_1, \dots, c_k\}$ and

$P_2 = (X_2, C_2)$ consists of the variables $X_2 = \{x_{\ell+1}, \dots, x_n\}$ and the constraints $C_2 = \{c'_{k+1}, \dots, c'_m\}$ of the form $c(d_1, \dots, d_\ell, x_{\ell+1}, \dots, x_n)$, whereby d_i is a concrete value for the variable x_i , $1 \leq i \leq \ell$.

In that, for $\ell + 1 \leq j \leq m$, c'_j arises from c_j by assigning the variables of X_1 concrete values: $x_i = d_i$, $1 \leq i \leq \ell$.

If $\ell = 0$ then P_1 is empty and if $\ell = n$ then P_2 is empty.

Clearly, if $\ell < n$ then the problem is a proper weakly decomposable problem.

A symmetry that acts on the subproblem P_1 is considered a weak symmetry.

Definition 2 (Weak Symmetry)

Given a weakly decomposable problem with subproblems P_1 and P_2 . A symmetry acting on P_1 but not on P_2 is called a weak symmetry.

If the weakly decomposable problem is proper then the symmetry is considered a proper weak symmetry.

If the weakly decomposable problem is not proper then the symmetry is a real symmetry.

In the following we will only consider proper weakly decomposable problems and proper weak symmetries for convenience.

A weak symmetry splits the problem into two parts: one that is symmetrical and one that is asymmetrical.

The symmetrical part contains all the constraints and variables that are affected by the weak symmetry.

The asymmetrical part contains the rest of the constraints and variables.

Both parts cannot be solved individually to receive a full solution.

Side remark: In our studies so far it is useful to regard the symmetrical part first and then regard the asymmetrical part. This is since the asymmetrical part in our studies is mostly concerned with deriving the objective value (in case of optimisation) or with constraints that evaluates the objective value (in satisfaction problems where a solution must at least achieve a certain objective value to be feasible). \square

2.2 Weak Symmetry as a Constraint Pattern

To characterize the weak symmetry problem we state it following the definition of constraint patterns ([Walsh, 2003]).

- Pattern name: WeakSymmetry
- Context: A symmetry that acts only on a subset of the variables respecting only a subset of the constraints in terms of feasibility. The symmetry splits the problem in a symmetrical and an asymmetrical part intimately related to each other.
- Problem: Breaking the weak symmetry without losing solutions.
- Forces: Breaking the weak symmetry is not possible without losing solutions and not breaking it is highly inefficient.
- Solution: Using SymVars results in a rearrangement of the search-tree that allows all known kinds of standard symmetry breaking for weak symmetries.
- Example: Weighted n-Queens, Multiple Rack Configuration, Automated Manufacturing

3 The Effect of Symmetries in the Search Tree

For convenience, we consider a static variable ordering (where concerning weak symmetries the symmetrical variables are consecutively ordered starting at the root of the tree) but the ideas can also be applied to a dynamical variable ordering.

3.1 Proper Symmetries in the Search Tree

If a problem contains symmetries all paths of an equivalence class of solutions of (partial) solutions can be identified and represented by a single path. Therefore the pruned search tree contains only a fraction of the paths of the straight forward search tree.

In practice, this means that symmetrical (partial) solutions are evaluated as no-goods as soon as possible and the underlying subtree is pruned.

Therefore the explored search tree also excludes all but one path per equivalence class if the symmetry is broken.

3.2 Weak Symmetries in the Search Tree

If a problem contains weak symmetries the symmetry function can only be applied to nodes up to a specific depth in the search tree since up to this depth the symmetric variables of P_1 are arrayed. Therefore only paths up to this specific level could be identified. But the subtrees under each symmetrical subpath of a class are different. Therefore these paths cannot be identified and represented by a single path without losing solutions in the subtree. These solutions **cannot** be derived using the weak symmetry function.

Therefore the search tree contains all symmetrical paths.

To our best knowledge there is no way of excluding weak symmetries without losing solutions in this search tree.

4 Weak Symmetries in Practice

Problems including weak symmetries can be found in optimisation as well as in satisfaction problems. Also several standard problems can be extended to fit in the class of proper weakly decomposable problems. The easiest way to extend a problem to be proper weakly decomposable is to cluster two problems that are familiar in some way like the magic-knight-tour which is a combination of a knight tour and a magic square [Jelliss, 2004], [Stertenbrink, 2004].

4.1 Proper Weakly Decomposable Optimisation Problems

Every optimisation problem can be considered weakly decomposable as follows: The objective value can be represented by a variable and the objective function is a constraint that determines the value of the objective variable. Therefore the original problem forms P_1 and the optimisation part forms P_2 of the weakly decomposable problem.

If the objective function is in the same way symmetrical as the weak symmetry in P_1 then the problem is not proper weakly symmetrical.

There are optimisation problems where P_2 consists of further variables and constraints. In such a case the variable assignment of these further variables depend on the variable assignment of P_1 .

An example of such a problem is the “open warehouse” problem where the number of sold goods is to be maximised (See [Hentenryck, Lustig, 1999] for a detailed description). There are several stores and each can only sell goods if they are open. The variables of P_1 would be the decision variables whether a store is open or not while the variables that state the number of goods sold are a part of P_2 .

This is demonstrated by a short model using ILOG OPL Syntax [Hentenryck, Lustig, 1999]:

```
int+ numberOfStores;
range Stores 1..numberOfStores;

int+ numberOfGoods;
range Goods 1..numberOfGoods;

int+ InStock[Stores, Goods] =...;

var Open[Stores] in 0..1;
var Sold[Stores, Goods] in 0..maxint;

...
forall (s in Stores)
Open[s] = 0 => Sold[s, g in Goods] = 0;
```

4.2 Proper Weakly Decomposable Satisfaction Problems

Satisfaction problems are often self-contained such that they cannot be decomposed reasonably. Nonetheless many can be extended to be proper weakly decomposable. In the case of extension the original problem forms P_1 and the extension

forms P_2 . The variable assignment of the variables in P_1 predefines the feasible domains for the variables in P_2 .

If the problem is weakly symmetrical this could lead to the fact that not all partial symmetrical solutions of a class are feasible with respect to the problem P . So the feasibility state for P_1 is preserved but not in general for P . That means that P_2 contains constraints that evaluates a partial solution infeasible.

Consider a problem where depending on a variable assignment for the symmetrical part variables are to be assigned. The variable assignment of the symmetrical part therefore predefines feasible variable assignments for the latter variables whereby even symmetrical solutions (for the symmetrical part) may yield different feasible variable assignments for the asymmetrical part.

This may even have the consequence that a symmetrical variable assignment yields only infeasible solutions.

4.3 Weak Symmetries in Extensions of Standard Problems

Example of an Optimisation Problem – Weighted N-Queens

Consider the *weighted n-Queen problem* where each field on the chessboard features a weight expressed by an integer. The task now is to find a n-Queen solution with minimal/maximal weight.²

The symmetrical part P_1 is clearly the n-Queen problem while the asymmetrical part P_2 is determining the objective value.

The weak symmetries acting on the variables of P_1 are all kinds of rotation and flips of the chessboard.

In this problem for example the flip around the x-axis delivers a symmetrical solution but this may have a different objective value and is therefore a different solution to the problem.

Example of a Satisfaction Problem – Time-based Rack Configuration

Consider the *time-based rack configuration problem*, a variation of the rack configuration problem (CSPLib problem031 [Gent, Walsh, 2003], [Kiziltan, Hnich, 2003]).

In the rack configuration problem a number of racks of different types are needed to plug in a certain number of different electronic cards. The task is to decide how many racks of each type are needed such that the price for all racks is minimal.

In the extension of the problem the price is modelled as an upper bound, that reflects the budget that could be spent at most for all racks. Any rack configuration that's price is below the budget is a feasible configuration. But additionally to the decision which racks are to be bought the time of purchase has to be considered or more specifically the order in which they are bought. Also for some kind of electronic cards there are constraints that state when they have to be used in the latest.

The symmetrical part P_1 is the rack configuration (configured racks are feasible no matter in which order they appear)

²This problem could also be stated as satisfaction problem when a n-queen solution is wanted that achieves at least a specified value.

while the order of the racks is the asymmetrical part P_2 (since the order defines the feasibility of the racks with respect to the time constraints of the electronic cards).

The weak symmetry acting is the permutation of the racks. Any permutation of the racks is still a feasible rack configuration. But a specific order of the racks may be infeasible since a constraint stating the latest use of a specific card is violated.

Example from Automated Manufacturing

The problem formulation is a variation of a real world problem originating from a cooperation with Philips/Assembléon, Netherlands (see Fotenote 1). For a more specific descriptions see [Gaudlitz, 2004] and [Tazari, 2003].

Given is:

- A mounting machine with several in parallel working, consecutively ordered mounting devices.
- Several tools and container of a specific type for specific component types.
- A PC board layout specified by a set of mounting tasks, each consisting of a specific component type and coordinates where this component type has to be placed on the board.

The task is to find a setup for each mounting device – consisting of a tool and several component container – and – depending on this setup – find an optimal distribution of the mounting tasks to the mounting devices such that the workload is minimised.

The mounting devices are consecutively ordered and do not interfere with each other. The boards are arrayed along the mounting devices. Usually a board is visible on two mounting devices at a time such that both can mount on it, each on a different clipping of the board.

The mounting devices are identical in terms of the possibilities of assigning a setup but each has a different visibility range on the machine. Therefore each device has access to a different subset of mounting tasks of the boards.

Each mounting device receives its own setup that defines which mounting tasks can be processed by this device. Since the devices are symmetrical feasible setups can be swapped to other devices and they are still feasible. A specific setup achieves different performance values in terms of the job processing since the subsets of accessible mounting tasks are different for each mounting device.

The symmetrical part P_1 of the problem is the configuration of the setups for the individual mounting devices while the asymmetrical part P_2 is the distribution of the mounting tasks to the devices.

The weak symmetry acting is the permutation of the setups for the mounting devices since a feasible setup is feasible no matter to which mounting device it is assigned.

5 Handling Weak Symmetries

For the purpose of this paper we focus on the case that the weak symmetry function acts like a permutation.

5.1 Standard Methods

Usually a problem that contains weak symmetries could be decomposed in two independent subproblems whereby the

solution of the first is input to the second. The advantage of this method is that the weak symmetry in this case is a real symmetry in the subproblem and can be broken. All symmetrical solutions can be derived explicitly in post-processing to form the input for the second subproblem. The drawback is that the first problem has to be solved exhaustively to get all solutions before the second problem can be solved. But if one is just interested in one solution that satisfies all constraints (CSP) or the best solution that can be found in a specified time (CSOP) this approach would not be suiting.

A different approach would be to have a framework that supports several models and anytime a solution for the first problem is found it is passed to the second model to find a solution for it like ILOG OPLScript ([Hentenryck, Lustig, 1999]).

Both approaches would lead to a remodelling of the problem and either a special framework has to be used or the solving process is split since the solution of one problem has to be post processed to form the new input.

The approach we suggest to handle weakly decomposable problems is using variable objects called *SymVar*. Using *SymVars* enables us to use standard symmetry breaking methods without solving the first problem exhaustively and without the need of a special framework.

This approach also means to remodel the problem but only in the way of introducing new variables.

5.2 Breaking Weak Symmetries

The main problem in breaking weak symmetries is that the knowledge about the symmetrical structure of P_1 can not be exploited without losing solutions.

Since we are interested in symmetrical equivalents of a feasible solution to P_1 we do not care for infeasible solutions. An infeasible solution to P_1 is also infeasible to P_2 such that the state of infeasibility is preserved throughout the search. Therefore all infeasible classes can be excluded without doubt.

Moreover we are only interested in feasible solutions to P_1 not in feasible partial solutions. Therefore it would be sufficient to retrieve all symmetrical solutions of an investigated feasible solution and exclude the paths to that solutions. That means that we basically have to find only one representative of a feasibility class. That is exactly what symmetry breaking does.

So if we can retrieve all symmetrical solutions we can use all known kinds of symmetry breaking.

To retrieve all symmetrical solutions we use *SymVars* that state which symmetrical equivalent of a solution we consider further for P_2 .

6 The SymVar Concept

The main problem in handling weak symmetries is to receive all symmetrical solutions (once a solution is found) without searching in the symmetrical branches of the search tree.

To achieve this we introduce additional variables that act as an interface to the symmetry of the problem. These variables are the *SymVars*.

Formally the original problem description P is changed to P' which splits in the subproblems P'_1 , P_{sym} and P'_2 .

P_1 is extended to P'_1 by adding the symmetry breaking constraints. P_{sym} contains the SymVars and the constraints for the SymVar assignment. In fact the weak symmetry function is expressed by the constraints of P_{sym} .

P_2 is changed to P'_2 in the way that instead of the values for the variables of P'_1 the symmetrical equivalents expressed by the variables of P_{sym} are considered.

$P_{sym} = (X_{sym}, C_{sym})$ consists of the variables $X_{sym} = \{x_{s_1}, \dots, x_{s_r}\}$ and the constraints $C_{sym} = \{c_{s_1}, \dots, c_{s_t}\}$ of the form $c(x_{s_1}, \dots, x_{s_k})$.

P'_2 consists of the variables $X_2 = \{x_{\ell+1}, \dots, x_n\}$ and the constraints $C'_2 = \{c''_{k+1}, \dots, c''_m\}$, of the form $c(Sym(d_1), \dots, Sym(d_\ell), x_{\ell+1}, \dots, x_n)$.

$Sym(d_1), \dots, Sym(d_\ell)$ is a symmetrical solution – determined by the value assignment of the SymVars – of the concrete values for the variables x_1, \dots, x_ℓ .

That means that the difference between c'_j and c''_j is that the concrete values of the variables of X_1 are changed to a symmetrical value assignment. Which symmetrical values assignment that is, is determined by the values of the variables of X_{sym} .

Definition 3 (Symmetry Variable (SymVar))

The variables of X_{sym} are the SymVars. Each SymVar is a copy of a set of variables of the symmetrical problem P_1 . The set of SymVars represents all variables of the problem P_1 .

A value assignment of the SymVars represents a symmetrical solution for the variables of the problem P_1 .

Therefore the symmetry of the problem acts on the SymVars rather than on the initial variables X_1 such that the symmetry on the initial variables is broken.

An instance of a SymVar corresponds to a set of variables of P_1 forming a specific object in the model.

In the weighted n -Queens problem this object would be a single queen while in the time-based rack configuration problem it would be a whole rack and its included electronic cards. In the last context a value assignment to a SymVar that represents a specific rack would assign this rack to a specific time slot.

In case of the example from automated manufacturing from Page 4 a mounting device would be the corresponding object. A specific SymVar would in this case correspond to a specific setup for a mounting device while the value of the SymVar indicates to which specific mounting device this setup is assigned.

A permutation of the SymVar represents now a permutation of the variable assignments for the mounting devices. Therefore the symmetry of the mounting devices – the permutation of the devices – can be broken since all symmetrical solutions can be derived by the instantiations of the SymVars. The symmetry acts now only via the SymVars and is broken on the initial variables V_1 .

6.1 Ordering Heuristics for P_{sym}

The gain in using SymVars is that once a feasible solution to P'_1 is found all symmetrical solutions can be found just by solving P_{sym} and therefore the symmetry in P'_1 can be broken.

When the SymVars assignment is infeasible for P'_2 or another solution is to be evaluated, a different solution of P_{sym} is considered.

Basically this means that each solution of P_{sym} has to be considered since it may yield a different solution for P'_2 .

But depending on the problem and scenario not all symmetrical solutions have to be considered. Propagation can still take place and therefore ordering heuristics can be used to guide the search. In the automated manufacturing problem (Page 4) it would be a good idea to assign a setup to a mounting device such that as many mounting tasks as possible are visible.

In case of optimisation a good solution may rule out many other such that a great deal a symmetrical solutions can be pruned very early.

Ordering heuristics depend very much on the problem itself and finding a good one is still a crucial task.

But to our best knowledge there are no limitations in terms of using ordering heuristics.

6.2 Trade off in Using SymVars

Since P' contains more variables than the original problem P it means basically that it takes more variable assignments. In fact the height of the tree to make up a solution is greater than the height of the tree in P .

On the other hand, since the weak symmetry is broken in P' the width of its search tree is lesser than the width of P .

Depending on the problem, the instance, the symmetry and the scenario, SymVars may or may not lead to speed up in the search for better solutions.

Disadvantage Scenario

Consider a CSP where the first found solution of P'_1 does not lead to a feasible solution of P' and propagation is unable to prune efficiently. That would mean that all symmetrical solutions of P'_1 will be considered without finding a feasible solution.

The standard approach does not consider the symmetrical solutions necessarily and therefore it could happen that it finds a feasible solution while the SymVar approach still considers the symmetrical solutions.

Advantage Scenario

Consider an optimisation problem with a given computational time limit and consider further that the solution space is rather sparse but propagation can not prune effectively. Consider further that it is rather difficult to find a solution to P_1 but very easy to find a solution to P_2 .

The standard approach would invest much time in searching a feasible solution to P_1 potentially wasting much time with the unbroken weak symmetry. Even if a solution is found it could take a long time until the next is found because of the sparse solution space. Therefore it is likely that this approach does not find many solutions within the given time limit.

In the SymVar approach – once a solution for P'_1 is found – the whole equivalence class to this solution is at hand via the SymVars. Since it is easy to solve P'_2 many solutions are at hand. Therefore this approach is more likely to find many solutions within the given time limit increasing the chance of finding a good solution.

7 Examples using SymVars

In this section we provide insight to the usage and construction of SymVars for some problem descriptions.

7.1 Weighted n-Queen Problem with SymVars

In the weighted n-queen problem a SymVar represents a single queen. Since a geometrical symmetry is more complicated in comparison to a permutation, the SymVars act in a different way: A value assignment for a specific SymVar determines which symmetrical value its corresponding queen has (suppose each queen denotes a column) as well as its column on the chessboard. That means that the row value as well as the column representation are subject to the symmetry.

Example:

Consider the diagonal flip the symmetry function. That means that a queen on column i with row value j is translated to column j and row i .

The problem P' consists of the following parts:

P'_1 is just the n-queen problem extended by the symmetry breaking constraints for the diagonal flip.

P_{sym} consists of determining the symmetrical solutions (which in this case is the identity and the diagonal flip).

P'_2 consists of the problem for determining the objective value whereby the symmetrical values determined by P_{sym} are considered (instead of the values determined by P'_1).

P_{sym} is realised by

$$X_{sym} = \{q_{1_{sym}}, \dots, q_{n_{sym}}\} \text{ and}$$

$$C_{sym} = \{(q_{i_{sym}=q_i}), 1 \leq i \leq n \vee (q_{i_{sym}} = j | q_j = i), 1 \leq i \leq n\}.$$

Since the diagonal flip contains only two different values the constraint in C_{sym} just states these two symmetrical values as feasible.

It's also possible to break the other symmetries of the problem like rotations and any combination thereof. They just have to be incorporated in C_{sym} .

7.2 Automated Manufacturing – Revisited

We consider again the automated manufacturing problem from Page 4 and investigate how SymVars could be used to break the weak symmetry.

P'_1 still consists of the setup configuration part such that the task is to find a feasible setup for the mounting machine. In addition the symmetry of the permutation of the setups is broken.

Each setup for a mounting device is represented by a SymVar, such that its position on the machine is not determined yet. The task for P_{sym} therefore is to find a permutation of the setup delivered by P'_1 on the machine. This can be easily done by stating an alldifferent constraint on the SymVars.

The task of P'_2 is to find a mounting task distribution based on the permuted setup delivered by P_{sym} and compute the processing time for this distribution.

When an optimal distribution for this setup is determined a different permutation is considered for P'_2 .

7.3 Variation of the Problem

Following we will consider a variation of the above problem. The subproblem P_2 itself is hard to solve and finding an optimal distribution comprises a lot of work. There exists an

algorithm that solves the problem by using a network flow as core algorithm but the runtime is also exponential.³ But since we want to investigate the usage of SymVars we focus on solving P_1 and choose a heuristic for P_2 .

Our studies on the problem indicate that good solutions for P_2 can be found for a setup with a high visibility of the mounting tasks. Therefore the problem P is to find a feasible machine setup and maximise the number of visible mounting tasks. That means that the task for P_2 is just to compute the number of visible mounting tasks for the setup.

Given are:

- a variable matrix $A^{m \times n}$ (representing the mounting machine, whereby n represents the number of mounting devices and m represents the number of component container that can be assigned to each device)
- a number t of different types (representing the component types)
- $n \times m$ values $d_1, \dots, d_{n \times m} \in [1, \dots, t]$ that have to be assigned to A (representing the component container)
- a matrix $V^{n \times t}$ (representing the number of mounting tasks for each component type that are processable on each mounting device)
- several sets containing values in $[1, \dots, t]$ (representing the tool types)

Objective: Find a distribution of the values $v_1, \dots, v_{n \times m}$ to the matrix A such that the number of processable mounting tasks is maximised:

$$\max \sum_{(col \in 1..n, row \in 1..m)} V[col, A[row, col]]$$

Constraints:

- several types cannot be assigned simultaneously. This means that all values in a column must be in the same compatibility set
- the values in each column must be all different

Since the order in which the values are assigned in a column does not matter the values can be ordered increasingly breaking the row symmetry.

Using SymVars

P_1 is extended to P'_1 by adding the column breaking constraints.

P_{sym} consists of

$$X_{sym} = \{pos_1, \dots, pos_n\} \in \{1, \dots, n\} \text{ and}$$

$$C_{sym} = \{\text{alldifferent}(pos_1, \dots, pos_n)\}.$$

Each SymVar pos_i , $1 \leq i \leq n$ represents its corresponding column i and therefore the values assigned to this specific column in P'_1 . A value assignment $pos_i = j$ then yields, that the values of column i are permuted to column j .

P'_2 determines the objective value and consists of

$$X_2 = \{obj\} \text{ and}$$

$$C_2 = \{obj = \sum_{(col \in 1..n, row \in 1..m)} V[col, A[pos_{col}, row]]\}$$

³See [Gaudlitz, 2004] and [Tazari, 2003] for an implementation of the algorithm.

Preliminary Results

We generated several small instances of the problem to investigate the power of using SymVars. We used matrices of several sizes ranging from 5×6 to 5×10 and different number of types for the values. We also set a time limit of 20 minutes for the search and compared the results of the naive model with the one that uses SymVars. We used ILOG OPL Studio 3.7 to do the computation on a Pentium 4 with 3.2 GHz and 512 MB RAM.

The results (although preliminary) were very encouraging. In all instances the SymVars approach did find the first solution in less time compared to the naive approach. Even if the objective value was lesser for the SymVars approach on this first solution, another solution could be found (in the same or lesser time that it took the naive approach to compute the first solution) with a objective value that was better (sometimes up to 20 %) than the solution of the naive approach. Even more encouraging was that for all instances the objective value of the SymVar approach was even or much much better than the objective value of the naive approach and mostly the computational time was just a fraction of the solving time for the naive approach.

For example in an instance where A has the dimensions 5×10 the naive approach did find a first solution with objective value 230 after 256.7 seconds and found the best solution with objective value 257 after 580.2 seconds. The SymVar approach found its first solution with objective value 233 after 0.218 seconds, found its seventh solution with objective value 265 after 2.812 seconds and its best solution with objective value 276 after 305 seconds.

This small example shows that the SymVar approach seems to be very promising and that it is especially useful for time bounded optimisation.

8 Outlook

We introduced the weakly decomposable problem definition, the weak symmetry definition and delivered some extensions of well examined standard problems that contain weak symmetries.

We also introduced SymVars that represent symmetrical solutions of a subproblem with the consequence that the symmetry can be broken on the subproblem.

In addition we presented some problems where SymVars are applied and delivered preliminary computational results for on problem that were very encouraging.

An open question is whether there are other scenarios, where weak symmetries play a vital role and whether they can also be handled there in the way we introduced.

If the weak symmetry does not preserve the state of feasibility for the whole problem then there should be ordering heuristics such that not all $n!$ permutations have to be considered (in the case the symmetry is a permutation). Developing such heuristics would also be very interesting.

References

- [Apt, 2003] Krzysztof Apt *Principles of Constraint Programming* Cambridge University Press,2003

- [Backofen, Will, 1999] R. Backofen and S. Will *Excluding Symmetries in Constraint-Based Search* In: Principles and Practice of Constraint Programming, pp. 73-87,1999
- [Flener et al., 2002] P. Flener, A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, T. Walsh *Matrix Modelling: Exploiting Common Patterns in Constraint Programming* In: Proceedings of the International Workshop on Reformulating Constraint Satisfaction Problems, 2002
- [Flener et. al, 2002] P. Flener, A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, T. Walsh *Breaking Row and Column Symmetries in Matrix Models* In: 8th International Conference on Principles and Practices of Constraint Programming (CP-2002), Springer, 2002
- [Gaudlitz, 2004] Rico Gaudlitz *Optimization Algorithms for Complex Mounting Machines in PC Board Manufacturing* Diploma Thesis, TU Darmstadt, To appear Oktober 2004
- [Gent, Smith, 1999] I. Gent, B. Smith *Symmetry Breaking During Search in Constraint Programming* School of Computing Research Report 99.02, University of Leeds, 1999
- [Gent, Smith, 2000] I. Gent, B. Smith *Symmetry Breaking in Constraint Programming* In Horn, W., ed.: Proceedings of the 14th European Conference on Artificial Intelligence, pp. 599-603, 2000
- [Gent, Walsh, 2003] I. Gent, T. Walsh, B. Selman *CSPLib: a problem library for constraints* <http://4c.ucc.ie/tw/csplib>
- [Hentenryck, Lustig, 1999] V. Hentenryck, I. Lustig *The OPL Optimization Programming Language* The MIT Press
- [Jelliss, 2004] Compiled by George Jelliss *Knight's Tour Notes* <http://www.ktn.freeuk.com>
- [Kiziltan, Hnich, 2003] Z. Kiziltan, B. Hnich *Prob031: Rack Configuration Problem* <http://4c.ucc.ie/tw/csplib/prob031>
- [Puget, 2003] J.-F. Puget *Symmetry Breaking Using Stabilizers* In Rossi, F., ed.: Proceedings of 9th International Conference on Principles and Practice of Constraint Programming (CP2003), Springer, 2003
- [Smith, 2000] B. Smith *Modelling a Permutation Problem* In: Proceedings of ECAI 2000 Workshop on Modelling and Solving Problems with Constraints, 2000
- [Smith, Gent, 2001] B. Smith, I. Gent *Reducing Symmetry in Matrix Models: SBDS v. Constraints* Held on SymCon'01, 2001
- [Stertenbrink, 2004] Hosted by Guenter Stertenbrink *Computing Magic Knight Tours* <http://magictour.free.fr>
- [Tazari, 2003] Siamak Tazari *Solving a core scheduling problem in modern assembly-line balancing* Technical Report, TU Darmstadt, Oktober 2003
- [Walsh, 2003] Toby Walsh *Constraint Patterns* In Rossi, F., ed.: Proceedings of 9th International Conference on Principles and Practice of Constraint Programming (CP2003), Springer,2003

Combining SBDS and SBDD

Karen E. Petrie

School of Computing and Engineering
 University of Huddersfield
 Huddersfield, HD1 3DH, UK
 k.e.petrie@hud.ac.uk

Abstract

We introduce two hybrid method of GAP-SBDS and GAP-SBDD; *SBDS+D* and *SBDD+S*. The implementation of these hybrids comes from a comparison of GAP-SBDS and GAP-SBDD, which concludes that combining the methods may produce a more efficient symmetry breaking method than either of the two methods alone. SBDS+D allows the user to change from SBDS to SBDD, at any depth in the search tree, SBDD+S allows the opposite change from SBDD to SBDS. Experimentally we show that SBDD+S is an incomplete symmetry breaking method, which is less efficient than either GAP-SBDS or GAP-SBDD alone. Before presenting SBDS+D, which is a complete symmetry breaking method; that can outperform both GAP-SBDS and GAP-SBDD on a range of problems with different types of symmetry.

1 Introduction

Constraint Satisfaction Problems (CSPs) are often highly symmetric. Symmetries may be inherent in the problem, as in placing queens on a chess board that may be rotated and reflected. Additionally the modelling of a real problem as a CSP can introduce extra symmetry: problem entities which are indistinguishable may in the CSP be represented by separate variables leading to $n!$ symmetries between n variables.

Definition of Symmetry Given a CSP L , with a set of constraints C , a symmetry of L is a bijective function f which maps a representation of a search state α to another search state, so that the following holds:

1. If α satisfies the constraints C , then so does $f(\alpha)$.
2. Similarly, if α is a no-good, then so too is $f(\alpha)$. [13]

Symmetries can give rise to redundant search, while searching for solutions a partial assignments may be considered which is symmetric to one previously examined. If a partial assignment does not lead to a solution, neither will any symmetric assignment, and if it does lead to a solution, the new

solution is symmetrically equivalent to one already found. To avoid this redundant search constraint programmers try to exclude all but one in each equivalence class of solutions. Many methods have been developed for this purpose. Two of these methods for symmetry exclusion which operate during search are, symmetry breaking during search [1; 10], and symmetry breaking via dominance detection [3; 5]. More recently, computational group theoretic versions of these methods have been devised, namely GAP-SBDS [8] and GAP-SBDD [9].

Symmetry breaking during search (SBDS), was developed by Gent and Smith [10], having been introduced by Backofen and Will [1]. The search tree is built from decision points, where a decision point has two possible choices; either assign a value to a variable, or do not assign that value to that variable. When a decision point is first reached during search a value is assigned to a variable; if at a later stage in search the decision point is revisited then a constraint is imposed that the variable should not have the previously assigned value. SBDS operates by taking a list of symmetry functions (provided by the user) and placing related constraints when backtracking to a decision point and taking the second branch.

A feature of SBDS is that it only breaks symmetries which are not already broken in the current partial assignment: this avoids placing unnecessary constraints. A symmetry is broken when the symmetric equivalent of the current partial assignment is not consistent with that assignment. The following expression explains how SBDS works:

$$A \& g(A) \& var \neq val \Rightarrow g(var \neq val)$$

where A is the partial assignment made so far during search, $g(A)$ is the symmetric equivalent of A and $var \neq val$ is the symmetrical equivalent to this failed assignment. If A is the current partial assignment and we have established that $var \neq val$, we need to ensure that we are dealing with an unbroken symmetry, so we check that $g(A)$ still holds. Then to ensure that the symmetrically equivalent subtree to the current subtree will not be explored, the constraint $g(var \neq val)$ is placed. An SBDS library is now available in the ECL*iPS*^e constraint programming system [2]. As previously mentioned, SBDS requires a function for each symmetry in the problem describing its effect on the assignment of a value to a variable. If these symmetry functions are correct and complete, all the symmetry will be broken; as a result of this only

non-isomorphic solutions will be produced. Although SBDS has been successfully used with a few thousand symmetry functions, many problems have too many symmetries to allow a separate function for each.

To allow SBDS to be used in situations where there are too many symmetries to allow a function to be created for each, Gent *et. al.* [8] have linked SBDS in ECLⁱPS^e with GAP (Groups, Algorithms and Programming) [7], a system for computational algebra and in particular *computational group theory* (CGT). Group theory is the mathematical study of symmetry. GAP-SBDS allows the symmetry group rather than its individual elements to be described. GAP is used when a value is assigned to a variable, at a decision point, to find the *stabiliser* of the current partial assignment, i.e. the subgroup which leaves it unchanged. Then if the decision point is revisited on backtracking, the constraints are dynamically calculated from the stabiliser and placed accordingly. GAP-SBDS allows the symmetry to be handled more efficiently than in SBDS; the elements of the group are not explicitly created which is akin to what the symmetry functions represent in SBDS. However, there is an overhead in communication necessitated between GAP and ECLⁱPS^e.

Symmetry Breaking via Dominance Detection (SBDD) [3; 5] performs a check at every node in the search tree to see if it is dominated by a symmetrically equivalent subtree already explored, and if so prunes this branch. In SBDD, the dominance detection function is based on the problem symmetry and is hard-coded for each problem. This means in practice SBDD can be difficult to implement, as the design of the dominance detection function may be complicated; the user has to ensure that all the symmetry of the problem is incorporated within the function to enforce full symmetry breaking.

Gent *et. al.* [9] have recently developed GAP-SBDD, a generic version of SBDD that uses the symmetry group of each problem rather than an individual dominance detection function and links SBDD (in ECLⁱPS^e) with GAP. At each node in the search tree, ECLⁱPS^e communicates the details of that node to GAP, and GAP returns false if dominance has been detected and that branch can be pruned, or true otherwise. Occasionally full dominance is not detected but there are variable/value pairs which are easily detected as being eligible for domain deletion; at which point GAP returns true followed by a list of variable/value pairs for which this is the case. ECLⁱPS^e removes these values from the corresponding variables domains before search continues.

In this paper we compare GAP-SBDS and GAP-SBDD using ‘Graceful Graphs’ as our motivating example. This study leads us to an analysis of exactly how the methods differ. We use this analysis to create hybrid methods of GAP-SBDS and GAP-SBDD. Before comparing our new SBDS+D and SBDD+S methods to both GAP-SBDS and GAP-SBDD, producing favourable results in the case of SBDS+D.

2 Comparison of SBDS and SBDD

Limited past work has been conducted comparing GAP-SBDS and GAP-SBDD. Harvey [11] studied the algorithms theoretically to draw the conclusion that SBDS and SBDD are closely related; the difference is where in the search tree

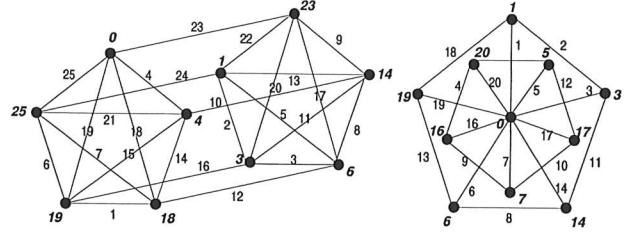


Figure 1: Graceful Labellings of $K_5 \times P_2$ and the Double Wheel DW_5

and how, symmetry breaking is enforced. Gent *et. al.* [9] compared GAP-SBDS and GAP-SBDD applied to instances of the balanced incomplete block design (BIBD) problem and showed that GAP-SBDD could solve much larger problems, and was faster than GAP-SBDS on the smaller problems which both could solve. They surmised this was due to the communication overhead between GAP and ECLⁱPS^e, the overhead in GAP-SBDD in only returning a true or false answer, is less than that of GAP-SBDS where fairly complicated constraints have to be placed. Petrie and Smith [15] investigated symmetry breaking in the *Graceful Graphs* problem, where they found GAP-SBDS to outperform GAP-SBDD on all instances. In this section we perform further analysis on this problem.

2.1 Graceful Graphs

The *Graceful Graphs* problem: A labelling f of the vertices of a graph with q edges is *graceful* if f assigns each vertex with a unique label from $\{0, 1, \dots, q\}$ and each edge xy is labelled with $|f(x) - f(y)|$ the edges are all different [6]. (Hence, the edge labels are a permutation of $1, 2, \dots, q$.) Figure 1 shows two examples. Lustig and Puget [12] give a constraint model for finding a graceful labelling of the graph $K_4 \times P_2$.

A basic CSP model has a variable for each node x_1, x_2, \dots, x_n , each with domain $\{0, 1, \dots, q\}$ and a variable for each edge d_1, d_2, \dots, d_q , each with domain $\{1, 2, \dots, q\}$. The constraints of the problem are:

1. If edge k joins nodes i and j then $d_k = |x_i - x_j|$.
2. x_1, x_2, \dots, x_n are all different.
3. d_1, d_2, \dots, d_q are all different.

ECLⁱPS^e provides two different levels of propagation for the *alldifferent* constraint. It can either be treated as a set of binary \neq constraints or as a *global alldifferent* which has higher propagation. We use the *global alldifferent* on the edge variables and the binary \neq version on the node variables. They are treated differently because the values assigned to the edge variables form a permutation and hence give more scope for domain pruning than the node variables, which have more possible values than variables. The node variables are used as the search variables as they are the simplest set to consider symmetry breaking over. More information on the modelling of this problem and the symmetry group can be found in [15].

The graph $K_5 \times P_2$, shown in figure 1, consists of two copies of K_5 , with corresponding vertices in the two cliques forming the vertices of path P_2 . The symmetries of $K_5 \times P_2$

	BT	ECL ⁱ PS ^e time	GAP time	Total time
GAP-SBDD	$K_3 \times P_2$	13	0.23	0.50
	$K_4 \times P_2$	173	7.18	2.72
	$K_5 \times P_2$	4402	337.69	88.20
GAP-SBDS	$K_3 \times P_2$	9	0.20	0.33
	$K_4 \times P_2$	165	7.15	1.35
	$K_5 \times P_2$	4390	352.10	36.61
				388.71

Table 1: Comparison of GAP-SBDS and GAP-SBDD showing backtracks (bt) and the time (in seconds) for finding all graceful labellings of $K_3 \times P_2$, $K_4 \times P_2$, $K_5 \times P_2$.

are first any permutation of the 5-cliques which act on both in the same way. Second, inter-clique symmetry: all the node labels in the first clique can be interchanged with the labels of the adjacent nodes in the second. Third, complement symmetry: we can replace every vertex label x_i by its complement $n - x_i$. These two graph symmetries and the complement symmetry can be combined with each other. Hence, the size of the symmetry group is $5! \times 2 \times 2$. In general $K_m \times P_2$ graphs have a symmetry group of size $m! \times 2 \times 2$. In this study we will concentrate on symmetry breaking in 3 such graphs, namely $K_3 \times P_2$, $K_4 \times P_2$ and $K_5 \times P_2$. The results of finding all graceful labellings of these graphs using either GAP-SBDS or GAP-SBDD can be found in table 1. All experiments throughout this paper are run on a 1.6GHz Pentium 4 processor with 512MB of memory, using ECLⁱPS^e version 5.7 and GAP version 4.2. Analysing the results in table 1 we can see that GAP-SBDD is slower than GAP-SBDS for all instances.¹ Experiments have shown these results to be general for all graphs, these results are documented in [14].

2.2 Analysis

We have analysed the difference between GAP-SBDS and GAP-SBDD for the three graphs $K_3 \times P_2$, $K_4 \times P_2$ and $K_5 \times P_2$. The reason for the different times is consistent, but for reasons of simplicity only the results for $K_3 \times P_2$ are presented here. A diagram of $K_3 \times P_2$ can be seen in figure 2, for ease of reference the node variables are named in capital letters, and the edge variables with a letter pairing corresponding to their attached nodes.

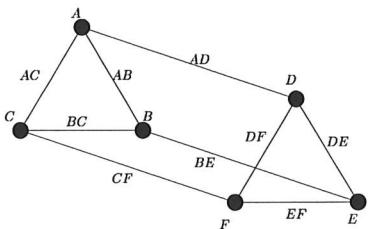


Figure 2: Graph $K_3 \times P_2$ showing node variable and edge variable naming

¹These results differ from those presented in [14] as GAP-SBDD has since been updated for efficiency and now provides more domain removals

We began analysing where GAP-SBDS and GAP-SBDD differ, by finding where the first difference in the search tree occurs. In finding all graceful labellings of $K_3 \times P_2$ this actually happens quite late in the search, after the first two solutions (from a possible 4) have been found. Of note is the fact that the backtrack count in table 1 refers to *deep-backtracks*. A deep-backtrack is when the search has moved passed a point it later has to revisit. A *shallow-backtrack* is where the $\text{var} = \text{val}$ branch has been tried, and due to propagation of that choice reversed in favour of the $\text{var} \neq \text{val}$ branch. The number of deep-backtracks is the standard backtrack count in most constraint programming environments, so eases the comparison of methods across environments, but perhaps in this case it does not show the most accurate picture of a search tree. This is important when studying GAP-SBDS, as every time the $\text{var} \neq \text{val}$ branch is followed, symmetry breaking constraints can be placed.

Looking more closely at the branch of the search tree where the first difference occurs (this can be found in figure 3) shows that GAP-SBDS enables earlier pruning than GAP-SBDD. This pruning happens after setting $C = 5$. GAP-SBDS immediately reverses from this decision to follow the $C \neq 5$ branch, whereas GAP-SBDD carries on from here to set $E = 1$, and ends up performing a deep-backtrack back to this point later on in search.

To understand why this difference occurs we have to look into the variable propagation. In the first instance both GAP-SBDS and GAP-SBDD perform propagation over the current partial assignment, in conjunction with the problem constraints. Past this point GAP-SBDS, propagates any symmetry breaking constraints previously employed on this branch. The two that are vital in this case are a combination of the graph symmetry and the complement symmetry for $B \neq 1$: namely $E \neq 8$ and $F \neq 8$. These extra constraints are placed on the node variables, but they provide extra information for propagation to occur on the edge variables as well. As we have already discussed, the values assigned to these variables form a permutation giving the *alldifferent* constraint more scope for pruning. This added propagation causes $C = 5$ to fail and the alternative path to be followed.

In contrast to this, GAP-SBDD just returns a boolean to indicate whether the current node is dominated or not, and possibly a list of values to prune from the domains of specific search variables. In the current implementation, the only variable/value pairs returned for domain pruning are, when the variable is the only one in the current partial assignment not to cause domination to be detected. So in this case $E/8$ and $F/8$ are not returned. This successfully breaks the symmetry and prunes the search tree, but it does not provide information that can propagate on any non-search variables, in this case the edge variables. Further details of these experiments and analysis can be found in [14].

3 Combining SBDS and SBDD

The fact that GAP-SBDD returns limited information to ECLⁱPS^e allows GAP-SBDD to solve much larger problems than GAP-SBDS as found by Gent *et. al.* in there BIBD experiments [9]. However our experiments and analysis have shown the disadvantage of this reduced communication. We

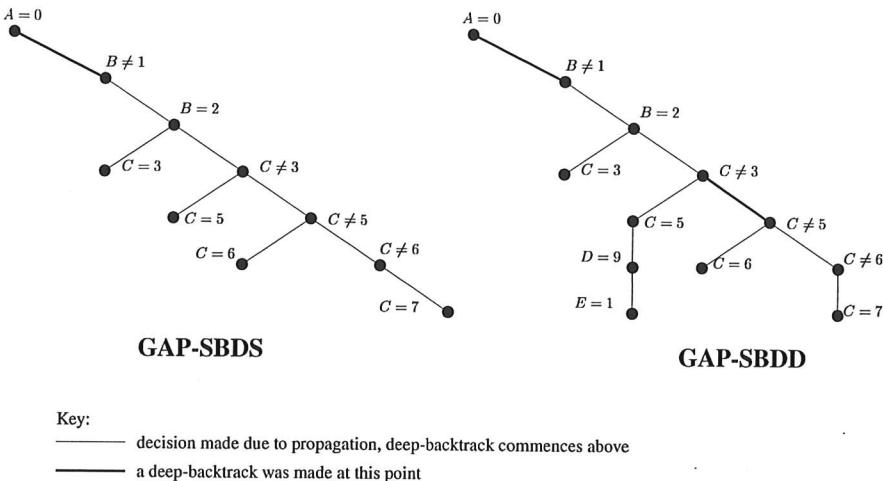


Figure 3: The search tree branch where GAP-SBDS and GAP-SBDD differ

have found that the constraints which GAP-SBDS places during search are adding information to the problem, and for some problems this can cause an increase in propagation. These observations led us to implement hybrid algorithms SBDS+D and SBDD+S, which combine the advantages of both methods.

When using either GAP-SBDS or GAP-SBDD, the methods are called at each node on the search tree, so the simplest way to combine the methods is to have different methods called at different nodes. This avoids duplicating any of the symmetry breaking effort. In SBDS+D we perform SBDS to a given depth in the search tree before switching to SBDD, in SBDS+D SBDD proceeds SBDS. This allows us to use SBDS at either the root or leaf nodes long enough to aid propagation before or after switching to/from the more efficient SBDD. In order to implement this change, two further parameters *method* and *level* are added to the symmetry breaking module. The *method* (either SBDD or SBDS) is started from the depth indicated by *level* in the search tree, with the other method operating until this level i.e. *method = SBDD* and *level = 1* would perform SBDD on the entire search tree, *method = SBDD* and *level = 2* would perform SBDS+D switching to SBDD from level 2. Figure 4 shows how this switch works at some other depths in the search tree.

Both the SBDS+D and the SBDD+S methods called at each node in search, operates the correct symmetry breaking method at the current depth. GAP-SBDS and GAP-SBDD operate on different properties of the symmetry group, we update the group information for both properties at each node. This allows the methods to be interchanged at a later point in search without the overhead of calculating symmetry data on any nodes visited previously on the search tree, but does provide an overhead over GAP-SBDS or GAP-SBDD alone. This interchanging can be done on backtracking, if we backtrack back past *level* as well as while forward searching. Throughout this paper we refer to the components of SBDS+D and SBDD+S as SBDS or SBDS as although they both utilise group theory, they no longer completely follow the GAP-SBDS and GAP-SBDD algorithms. The algorithm for this method is outlined in full.

A point to note (from the algorithm) is that in SBDS asserting *var = val* does not necessitate any symmetry breaking as it is all performed on the *var ≠ val* branch, whereas in SBDD both the *var = val* and *var ≠ val* cases are checked for domination. This means that SBDD works actively to break symmetry at all nodes, whereas SBDS just operates actively on a small subset.

4 SBDD before SBDS

The SBDD+S algorithm is an incomplete symmetry breaking method, by this we mean that it does not return only the non-isomorphic solutions. Performing SBDD at the top of the tree, performs a check to see if any of the nodes previously visited dominate the current node. At the required level in the search tree SBDS places constraints on backtracking to break the symmetry after the current depth. This combination does not give complete symmetry breaking as there is no mechanism to stop branches symmetric to those explored when performing SBDD, being chosen later in the search tree where SBDS is the chosen method. This method is a partial symmetry breaking method. McDonald and Smith [13] performed experiments using a subset of the total symmetries for each

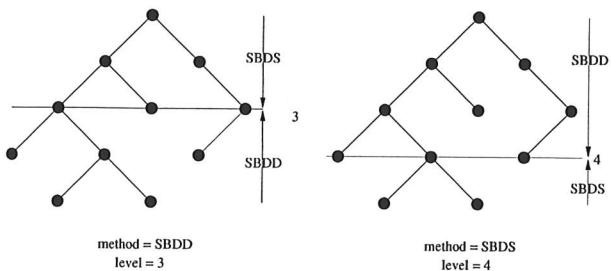


Figure 4: Various arrangements of *method* and *level* using algorithm SBDS+D and an SBDD+S respectively

```

SBDS+D;D+S( $A$ ,  $Var$ ,  $Val$ ,  $Method$ ,  $Level$ )
assert( $Var = Val$ )
 $Depth$  is  $Current\_Depth + 1$ 
update_sbds_info( $Var$ ,  $Val$ ) in GAP
update_sbdd_info( $Var$ ,  $Val$ ) in GAP
if (( $Method = SBDS$  and  $Level \geq Depth$ ) or
( $Method = SBDD$  and  $Level < Depth$ )) then
  if  $Var = Val$  is true then
    no symmetry breaking
  else
    retract( $Var = Var$ )
    get non_broken_symms from GAP
    for  $g$  in non_broken_symms do
      assert( $(g(A) \Rightarrow g(var \neq val))$ )
    end do
  end if
else
  get is_node_dominated(Bool, Removals) from GAP
  if ( $(Var = Val$  is true) and ( $Bool = false$ )) then
    reduce_domains(Removals)
    dominance_check_partial_assignment
  else if ( $(Var = Val$  is false) and ( $Bool = false$ )) then
    retract( $Var = Val$ )
    reduce_domains(Removals)
    dominance_check_partial_assignment
  else
    dominance detected so backtrack
  end if
end if

```

The SBDS+D and SBDD+S algorithm

problem with SBDS. They found that each symmetry has an overhead, so there is an optimum number to place for symmetry breaking in order to get the maximum gain in efficiency to solve the problem. In our method we are placing less symmetry breaking constraints than if we were performing pure SBDS, but still place a subset which will aid propagation.

We have undertaken experiments on the graceful graphs problem, to see how our method performs. Plots of experimental data showing number of solutions against level of change from SBDD to SBDS, indicate the amount of symmetry broken at each level. Plots of backtracks against level of change, indicate the search effort associated with each level. This data is found in figure 5 and figure 6 for graceful labellings of $K_3 \times P_2$ and $K_4 \times P_2$ respectively.

The data point at $level = 1$ on the plots shows the results for performing SBDS on the entire tree. Studying the first plot in both figures 5 and figure 6 we can see that $K_3 \times P_2$ has 4 and $K_4 \times P_2$ has 15 non-isomorphic solutions. Performing SBDD from $level = 2$ causes a dramatic rise in the number of solutions, which peaks at $level = 3$ before slowly decreasing. This dramatic rise is due to the fact that if using GAP-SBDS when backtracking past the root node in a search tree none of the symmetry is broken in the current partial assignment so a symmetry constraint is placed for every one of the problem symmetries, which can rule out a vast number of branches from being followed later in search. The slow decrease after the plots peak is because SBDD is conduct-

ing symmetry breaking on more of the tree as the level which SBDS enters at decreases, until a level which finds only the non-isomorphic solutions is reached. At this level, which is lower for $K_4 \times P_2$ than $K_3 \times P_2$ due to the increased number of search variables, the depth in the search where SBDS would be triggered is never reached so full symmetry breaking is conducted by SBDD.

Studying the second graph in both figure 5 and figure 6 we see that the number of backtracks, and hence the amount of search undertaken, rises in proportion to the number of solutions at each level. The total running time also rises in proportion to the number of backtracks. Using SBDD+S never performs better than GAP-SBDD or GAP-SBDS alone. These results have been verified on other graceful graphs and on other problem classes.

5 SBDS before SBDD

The SBDS+D algorithm is a complete symmetry breaking method. SBDS operates at the top of the tree at and near the root node, at this level few decisions have been undertaken which could break symmetry; so if we backtrack past these nodes constraints to break a large percentage of the total problem symmetries can be placed. The propagation of these constraints can reduce branches being explored in the search tree, hence they can cut down the number of failed variable to value assignments. It is worth noting here that SBDD only returns candidate values for domain removal for decisions that if were to be the next decision in the search tree would cause dominance to be detected. Hence early in the search tree there

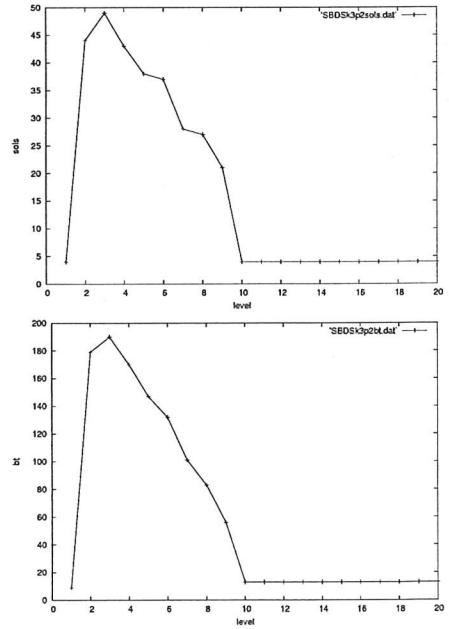


Figure 5: Plots of $K_3 \times P_2$ showing depth of change from SBDD to SBDS against number of solutions and number of backtracks respectively.

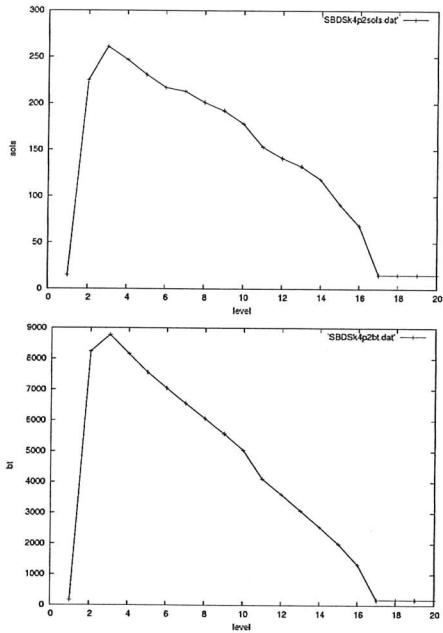


Figure 6: Plots of $K_4 \times P_2$ showing depth of change from SBDD to SBDS against number of solutions and number of backtracks respectively.

are unlikely to be any of these domain removal values to be returned, so the use of SBDD at this point would not aid constraint propagation. Later in the tree we change from SBDS to SBDD, which performs the faster dominance check at each node. In doing so SBDD checks all solutions for dominance by a previous solution, ensuring only the non-isomorphic solutions are returned.

We have performed experiments with SBDS+D on various problems, with various ‘types’ of symmetry groups. The results of these experiments have shown that SBDS+D, can do better than GAP-SBDD or GAP-SBDS alone.

5.1 Graceful Graphs

The graceful graphs problem, as we have shown, is a problem where GAP-SBDS operates more efficiently than GAP-SBDD. This is due to the fact that the search variables are not the most constrained variables, and the symmetry breaking constraints placed by SBDS propagate with these non-search variables. GAP-SBDD performs a more efficient symmetry check at each node than GAP-SBDS. So we experimented with SBDS+D to see whether the extra propagation of SBDS combined with SBDD will produce a better symmetry breaking method for this class of problems. We are also interested to see at which level this will happen. Figure 7 and figure 8 show plots of $K_3 \times P_2$ and $K_4 \times P_2$ respectively showing backtracks and time against level of change from SBDS to SBDD.

The data point at $level = 1$ on the plot represents performing GAP-SBDD on the entire search tree. Studying the first plot in both figure 7 and figure 8, we see the backtrack

count for GAP-SBDD alone is 13 and 173 for $K_3 \times P_2$ & $K_4 \times P_2$ respectively; at $level = 2$ this falls to 9 & 163 which it then remains at for all subsequent levels. Using GAP-SBDS $K_3 \times P_2$ & $K_4 \times P_2$ have 9 and 163 backtracks respectively. By placing SBDS constraints at the root node the search effort of SBDS+D is equal to that of GAP-SBDS. Turning to the time plots; the time for GAP-SBDD alone is higher than that for any level of SBDS+D for both graphs labellings. The time falls dramatically from $level = 1$ to $level = 2$ in $K_3 \times P_2$ and to $level = 3$ in $K_4 \times P_2$, we then see a slight rise as the overhead for placing SBDS constraints, while maintaining group information for SBDD begins to take effect. The time decreases again as SBDS is used across a bigger proportion of the search tree, until SBDD is never triggered. The symmetry breaking is performed purely by SBDS at $level = 10$ for $K_3 \times P_2$ and $level = 17$ for $K_4 \times P_2$. The total time taken by SBDS+D is comparative to that taken by SBDS from $level = 5$ for $K_3 \times P_2$ and $level = 11$ for $K_4 \times P_2$.

5.2 N-Queens

The N -queens problem is to place N queens on an $N \times N$ chessboard so that no queens can attack each other along a horizontal, vertical or diagonal line. The symmetry group of this problem comprises of the 7 board symmetries. It is an example of a problem with a small symmetry group which is the same order for any N . This class of problems is more efficiently solved by GAP-SBDS than GAP-SBDD as the communication overhead of GAP-SBDD is less than that of GAP-SBDS, but the group theory calculations are more complex. The results of applying SBDS+D to N-queens are in table 2.

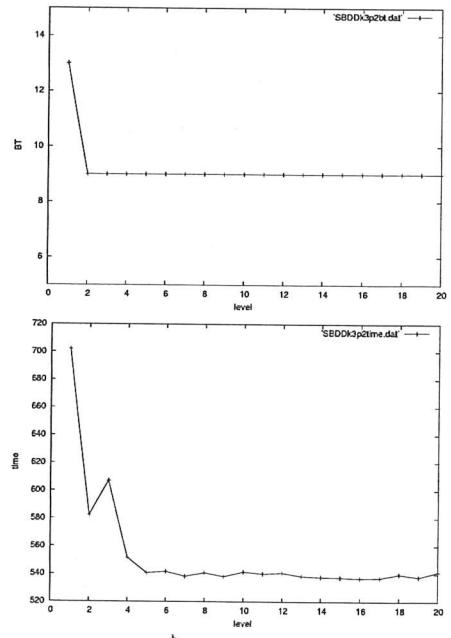


Figure 7: Plots of $K_3 \times P_2$ showing depth of change from SBDS to SBDD against number of backtracks and time (in Ms) respectively.

N	GAP-SBDS		SBDS+D level = 1		SBDS+D level = 2		SBDS+D level = 3		SBDS+D level = 4		SBDS+D level = 5		SBDS+D level = 6	
	BT	time	BT	time	BT	time	BT	time	BT	time	BT	time	BT	time
4	2	312	2	317	2	311	2	305	2	305	2	303	2	311
5	2	310	3	326	2	320	2	313	2	311	2	324	2	309
6	4	338	7	346	6	358	4	338	4	336	4	341	4	337
7	11	360	15	408	13	406	11	383	11	352	11	360	11	361
8	34	436	39	636	37	579	34	574	34	530	34	463	34	481
9	130	875	146	1379	137	1256	130	1130	130	1114	130	983	130	936
10	461	2496	505	4248	477	3674	462	3337	461	3387	461	3156	461	2873

Table 2: Comparison of GAP-SBDS and SBDS+D showing backtracks (bt) and the time (in Ms) for finding all N-Queens solutions.

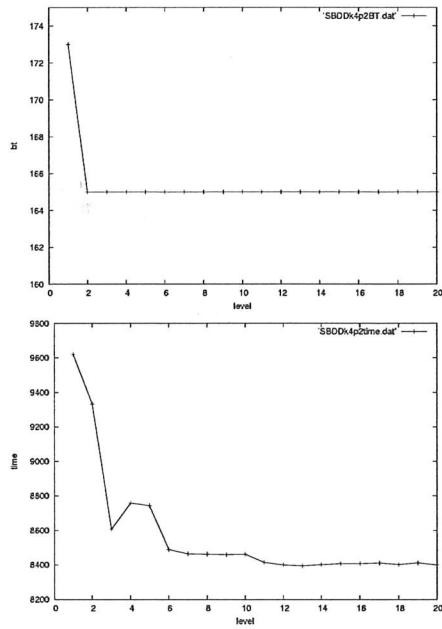


Figure 8: Plots of $K_4 \times P_2$ showing depth of change from SBDS to SBDD against number of backtracks and time (in Ms) respectively.

Analysing the results for N-Queens we see that SBDS is always faster and more efficient than GAP-SBDD alone (SBDS+D & $level = 1$). The number of backtracks using SBDS+D across all instances decreases to be the same as that of GAP-SBDS as the level of change increases. The total time for SBDS+D is always less than that of GAP-SBDD and in the smaller cases can also be less than that of GAP-SBDS. Continuing to increase the level further in SBDS+D for $N = 8, 9 \& 10$ never decreases the time to below that of SBDS. Overall, using SBDS+D decreases the search effort and time in comparison to GAP-SBDD.

5.3 BIBDs

The computational version of the (v, b, r, k, λ) balanced incomplete block design (BIBD) problem is to find a $v \times b$ binary

matrix such that each row has exactly r ones, each column has exactly k ones and the scalar product of each pair of distinct rows is λ . The columns and rows of the matrix can all be permuted, giving the problem a symmetry group of size $v! \times b!$. Gent *et. al.* found GAP-SBDD capable of solving more instances than GAP-SBDS; for the problems both could solve GAP-SBDD was the more efficient method. Our experiments (in table 3) compare GAP-SBDD across the whole problem (SBDS+D with $level = 1$) against using SBDS just at the root node (SBDS+D with $level = 2$). SBDS+D with $level > 2$ does not show any improvement over SBDS+D with $level = 2$ for all instances apart from (8,14,7,4,3), which has a larger search tree than other instances.

Studying table 3 we see that by using SBDS to place symmetry breaking constraints at the root node, we have reduced the number of backtracks in comparison to SBDD by 1, in every instance. Given the small number of backtracks needed to solve these instances, this is a reasonable reduction in search. The solving time is also slightly reduced in every instance apart from (8, 14, 7, 4, 3), this instance requires placing SBDS constraints to a deeper level to show an improvement, due to its larger search tree.

We can not solve some of the instances with SBDS+D that SBDD alone can solve, such as the (6, 20, 10, 3, 4) and the (7, 21, 6, 2, 1) instances, which have $O(10^{21})$ and $O(10^{23})$ problem symmetries respectively. With such large symmetry groups the communication overhead between GAP & ECLⁱPS^e required to place symmetries even at the root node is too high. In the future we hope to remedy this situation

v	b	r	k	λ	Parameters		$level = 1$		$level = 2$	
					BT	time	BT	time	BT	time
7	7	3	3	1	2	0.66	1	0.63		
6	10	5	3	2	3	1.90	2	1.80		
7	14	6	3	2	8	59.72	7	59.25		
9	12	4	3	1	2	3.05	1	2.79		
11	11	5	5	2	3	15.79	2	15.47		
8	14	7	4	3	10	450.38	9	465.37		
13	13	4	4	1	2	23.17	1	21.67		

Table 3: Comparison of SBDS+D with $level = 1$ and $level = 2$, showing backtracks (bt) and the time (in seconds) for finding all BIBD solutions.

by allowing the SBDS+D user to fix a maximum number of constraints to be placed as well as a level in the search tree to change symmetry breaking method.

The timings shown seem comparable with that of the double-lex symmetry breaking method [4] where constraints are placed to break symmetry before search commences. Double-lex is an incomplete symmetry breaking method, whereas our SBDS+D method guarantees only to return non-isomorphic solutions.

6 Conclusion

We have compared the GAP-SBDS and GAP-SBDD symmetry breaking methods; to conclude that although GAP-SBDD has less of an ECLⁱPS^e/GAP communication overhead than GAP-SBDS, the constraints GAP-SBDS places can cause it to be a more efficient symmetry breaking method. In order to combine the advantages of both methods we implemented hybrid methods SBDS+D and SBDD+S which allows a switch between SBDS and SBDD, or vice versa, at a given depth in the search tree.

Using SBDD+S is an incomplete symmetry breaking method; our experiments found this to be worse than GAP-SBDD or GAP-SBDS. In contrast, using SBDS+D is a complete symmetry breaking method. We performed experiments with SBDS+D on three problems, all with different symmetry ‘types’. In the graceful graphs problem the symmetry is more efficiently broken by GAP-SBDS than by GAP-SBDD due to the extra propagation of constraints. In this problem SBDS+D breaks the symmetry comparably to GAP-SBDS and more efficiently than GAP-SBDD. The N-queens problem has a symmetry group of order 8 for all instances, a small symmetry group is more efficiently broken by GAP-SBDS than GAP-SBDD; SBDS+D outperforms GAP-SBDD in all instances and GAP-SBDS, for small values of N . The BIBD has a large symmetry group so the symmetry is more efficiently dealt with by GAP-SBDD. The instances we could solve with SBDS+D had smaller search trees than GAP-SBDD and for all but one instance the total time was reduced. Overall, our hybrid SBDS+D method compares favourably with the use of either GAP-SBDS or GAP-SBDD, in nearly all instances undertaken. This is a preliminary study into this new algorithm, more experiments have to be conducted before we can claim under which conditions SBDS+D will outperform GAP-SBDD or GAP-SBDS, but these initial results show that it is a welcome addition to the suite of CGT dynamic symmetry breaking methods.

Acknowledgements

The author is a member of the CP-Pod group and would like to thank the other members; especially Ian Gent, Tom Kelsey, Barbara Smith and Paula Sturdy. I am also very grateful to Warwick Harvey & Steve Linton for their technical assistance, Neil Yorke-Smith, Chris Beck, Meinolf Sellmann & Ian Miguel for their discussions. This work was supported by EPSRC grant GR/R29673.

References

- [1] R. Backofen and S. Will. Excluding symmetries in constraint-based search. In J. Jaffar, editor, *Proc. of CP'99*, LNCS 1713, pages 73–87. Springer, 1999.
- [2] A. M. Cheadle, W. Harvey, A. J. Sadler, J. Schimpf, K. Shen, and M. G. Wallace. ECLiPSe: An introduction. Technical Report IC-Parc-03-1, IC-Parc, 2003. www.icparc.ic.ac.uk/eclipse/.
- [3] T. Fahle, S. Schamberger, and M. Sellmann. Symmetry breaking. In T. Walsh, editor, *Proc. of CP'01*, LNCS 2239, pages 93–107. Springer, 2001.
- [4] P. Flener, A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Breaking row and column symmetries in matrix models. In P. V. Hentenryck, editor, *Proc. of CP'02*, LNCS 2470, pages 462–476. Springer, 2002.
- [5] F. Focacci and M. Milano. Global cut framework for removing symmetries. In T. Walsh, editor, *Proc. of CP'01*, LNCS 2239, pages 77–92. Springer, 2001.
- [6] J. Gallian. A Dynamic Survey of Graceful Labeling. In *The Electronic Journal of Combinatorics*, 2002. (<http://www.combinatorics.org/Surveys>).
- [7] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.2*, 2000. (<http://www.gap-system.org>).
- [8] I. P. Gent, W. Harvey, and T. Kelsey. Groups and constraints: Symmetry breaking during search. In P. V. Hentenryck, editor, *Proc. of CP'02*, LNCS 2470, pages 415–430. Springer, 2002.
- [9] I. P. Gent, W. Harvey, T. Kelsey, and S. Linton. Generic SBDD Using Computational Group Theory. In F. Rossi, editor, *Proc. of CP'03*, LNCS 2833, pages 333–347. Springer, 2003.
- [10] I. P. Gent and B. M. Smith. Symmetry breaking in constraint programming. In *Proc. of ECAI-2002*, pages 599–603. IOS Press, 2000.
- [11] W. Harvey. Symmetry Breaking and the Social Golfer Problem. In *Proc. SymCon-01: Symmetry in Constraints*, pages 9–16, 2001.
- [12] I.J.Lustig and J.-F. Puget. Program Does Not Equal Program: Constraint Programming and Its Relationship to Mathematical Programming. In *INTERFACES*, volume 31(6), pages 29–53, 2001.
- [13] I. McDonald and B. M. Smith. Partial symmetry breaking. In *Proc. of CP'02*, LNCS 2470, pages 431–445. Springer, 2002.
- [14] K. Petrie. Why SBDD can be worse than SBDS. In *Proc. SymCon-03: Symmetry in Constraints*, pages 168–176, 2003.
- [15] K. E. Petrie and B. M. Smith. Symmetry breaking in graceful graphs. In *Proc. of CP'03*, LNCS 2833, pages 930–934. Springer, 2003.

Exploiting Dominance in Three Symmetric Problems

Steven Prestwich and J. Christopher Beck

Cork Constraint Computation Centre

Department of Computer Science, University College, Cork, Ireland

{s.prestwich,c.beck}@4c.ucc.ie

Abstract

Symmetry breaking has led to huge improvements in search performance, and has recently been the subject of considerable research. The related concept of dominance is even more powerful than symmetry, yet it has been relatively unused in Constraint Programming. This paper describes previously unexploited dominances for three well-studied symmetric problems. Experiments show the benefit of adding constraints to exclude both symmetric and dominated solutions.

1 Introduction

A great deal of work has been done by the Constraint Programming (CP) community in the area of detection and exploitation of symmetric search states. Informally, a pair of search states s_i, s_j is symmetric if there is a validity preserving mapping between them. For each extension of s_i (including s_i itself) there is a corresponding state that is an extension of s_j with the same validity. If the extension to s_i is a (non-)solution then so is the corresponding extension to s_j . Obviously this relationship is bi-directional: if s_i is symmetric to s_j then s_j is symmetric to s_i .

Dominance relations are a standard tool in the search for optimal solutions to combinatorial optimization problems. Informally, a dominance relation is a relation on a pair of search states s_i, s_j stating that the best solution that is an extension of s_j is no better than the best extension of s_i . Therefore only s_i needs to be extended. There are two primary differences between dominance relations and symmetry: the cost function of the former and the bi-directionality of the latter. While symmetry has traditionally been applied in satisfaction problems, we can easily model satisfaction problems as having a cost function that is zero for any solution and infinite for a non-solution. The bi-directionality can be dealt with by adding constraints to enforce the anti-symmetric condition of the dominance relation.

Even well-studied problems in the symmetry literature may also contain unexploited dominances that can be used to significantly improve search performance. Proll & Smith [16] used the term *pseudo-symmetry* to denote the same weakening of bi-directionality. They added pseudo-symmetry breaking constraints to improve search in a template design prob-

lem. Getoor et al. [12] added domain-specific redundant constraints to remove sub-optimal solutions from online scheduling problems. Gent et al. [10] recently added constraints to the Graceful Graphs problem to exclude dominated solutions. But despite these examples, dominance has been far less exploited than symmetry in CP.

With the aim of stimulating further research in this area, we present three case studies using symmetric problems from the CP literature. Section 2 provides some background. Section 3 studies the Maximum-Density Still Life problem, Section 4 Steel Mill Slab Design, and Section 5 Peaceable Armies of Queens. Section 6 concludes the paper. All our experiments are performed on a 733 MHz Pentium II.

2 Dominance Relations and Symmetry

Ibaraki [14] provides a formal definition of dominance relations to find a single optimal solution.¹ Assuming a standard constructive tree search as in common in CP, let S be the set of all search states, and let $f(s)$ be the minimum cost feasible solution that is an extension of the search state, $s \in S$. If s is infeasible then $f(s) = \infty$. A dominance relation is a binary relation \preceq on search states that satisfies the following conditions:

- $s_i \preceq s_j$ implies $f(s_i) \leq f(s_j)$
- \preceq is a partial ordering: transitive, reflexive, and anti-symmetric
- $s_i \preceq s_j \wedge s_i \neq s_j$ implies that there exists some extension $s_{i'}$ of s_i such that for all extensions $s_{j'}$ of s_j , $s_{i'} \preceq s_{j'} \wedge s_{i'} \neq s_{j'}$

The anti-symmetric requirement means that if $f(s_i) = f(s_j)$ and $s_i \neq s_j$ then only one of $s_i \preceq s_j$ or $s_j \preceq s_i$ can hold. In that case, a simple way to tie-break is to allow $s_i \preceq s_j$ if s_i was found before s_j in the tree search.

As an example of dominance relations, Ibaraki uses the (now) familiar 8-queens where the dominance relation $s_i \preceq s_j$ holds if and only if the patterns represented by the search states are isomorphic and s_i was found before s_j .

¹Ibaraki presents alternative conditions for finding all optimal solutions, while another condition that we do not discuss here arises from technical aspects of branch-and-bound search.

Because an explicit relation on all pairs of search states is impractical, dominance relations are often based on properties of the search states. That is, a property X is identified and it is proved that search states with X dominate search states without X . For example, in job shop scheduling, solutions which are “semi-active” have been shown to dominate non-semi-active solutions and a common heuristic searches only for semi-active solutions [7]. We will follow this property-based approach.

Suppose that we are able to identify a property P of a search state with the following attribute: if there exists a solution satisfying P then there also exists at least one solution satisfying $\neg P$. This is a one-way relationship because the existence of a solution satisfying $\neg P$ need not imply the existence of a solution satisfying P . However, by only considering solutions satisfying $\neg P$ we can reduce the search space without affecting validity. One way to do this is to add constraints to enforce $\neg P$. We shall call this technique *dominance enforcement*. It is directly analogous to the addition of symmetry breaking constraints to a model [17] and is justified by the following simple theorem:

Theorem. *Dominance enforcement preserves validity.*

Proof. Consider two cases. (i) There is no solution satisfying P . Then adding the constraint $\neg P$ excludes no solutions. (ii) There is at least one solution satisfying P . Then there also exists at least one solution satisfying $\neg P$, which the constraint $\neg P$ does not exclude. QED.

However, this does not prove that combining two dominance constraints is guaranteed to preserve at least one solution. (The same applies to symmetry breaking, for example when breaking both row and column symmetries in a matrix model [8] one must be careful to combine the two sets of constraints in the correct way.) In this paper we shall ignore this important point, as this is a work in progress, but a more formal treatment of the subject would be a useful direction for future work. This might proceed along similar lines to the many recent applications of group theory to symmetry breaking. Other approaches such as Symmetry Breaking During Search (SBDS) [2; 11] might also be generalized to dominance enforcement.

Note that, unlike symmetry breaking, by enforcing dominance we may lose access to the full set of solutions. But this is unimportant for problems in which we are interested in finding any [optimal] solution, or in proving insolubility [optimality].

3 Maximum-Density Still Life

Our first case study is the game of Life, invented by Conway in the 1960s. In an infinite 2-dimensional array, each cell is either alive or dead and has 8 neighbours. The game is initialized by setting each cell alive or dead. Subsequently the array is transformed into a new pattern for as many iterations as desired using a few simple rules: (1) a cell with 2 living neighbours is unchanged in the new pattern; (2) a cell with 3 neighbours is alive in the new pattern; and (3) any other cell is dead in the new pattern. A *still-life* is a pattern that does not change between iterations. A *maximum density* still-life

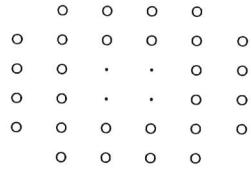


Figure 1: A still-life pattern for dominance enforcement.

is one with the greatest number of live cells, in a finite square region R of $N \times N$ cells (all cells outside R are dead).

3.1 A Basic Model

Two integer program (IP) formulations of this problem were given in [3]. We use the second, better model which has a 0/1 variable x_e for each cell $e \in R$, 1 denoting a live cell and 0 a dead one. The constraints are as follows:

- Death by isolation: $2x_e - \sum_{f \in N(e)} x_f \leq 0$, where $N(e)$ denotes the variables corresponding to the neighbours of e 's cell.
- Death by overcrowding: $3x_e + \sum_{f \in N(e)} x_f \leq 6$.
- Birth: $\forall S \subset N(e), |S| = 3 : -x_e + \sum_{f \in S} x_f - \sum_{f \in N(e)-S} x_f \leq 2$.
- Cells outside R cannot become alive: $x_f + x_g + x_h \leq 2$, where f, g and h are 3 cells lying in a line along the boundary of R .
- The density must be at least d : $\sum_{e \in R} x_e \geq d$.

To solve the problem we solve a series of CSPs with increasing d .

3.2 Dominance

Consider the pattern of cells in Figure 1 with \circ denoting dead cells. In any still-life the cells marked \cdot can be all live or all dead. Moreover, if there is a solution of density at least d in which they are all dead then there is also a solution of density greater than d in which they are not all dead. This is the property P we need to apply dominance enforcement constraints: we add a constraint $\neg P$ forcing one of them to be live. However, if one of the four is live they must all be live, in order to form a still-life, so we can add a stronger constraint: $\sum_{i \in O} 4x_i + \sum_{j \in X} x_j \geq 4$. X denotes the cells marked \cdot while O denotes those marked \circ . The pattern may be partly off the edge of the finite region R , as long as the four central cells are inside R . We call these constraints D .

A second dominance is based on the following observation: if there is a solution whose top row contains only dead cells then there is also a solution in which this is not true. We can simply translate the pattern one or more rows upward until a live cell appears in the top row. Therefore we can add a translational dominance enforcement constraint to exclude patterns in which the top row is all dead. Similarly, we can exclude patterns in which the left column is all dead: $\sum_{i=1}^N x_{i1} \geq 1$ and $\sum_{j=1}^N x_{1j} \geq 1$. We call these constraints T . Note that T are not symmetry breaking constraints: not every pattern with live top row cells can be translated to one whose top cells are all dead.

N	M	$M+T$	$M+D$	$M+T+D$
5 opt proof	999 709	877 639	999 709	877 639
6 opt proof	7501 25703	3009 22292	7484 25700	2993 22289
7 opt proof	161626 159718	145563 149054	161542 159686	145487 149023
8 opt proof	4893631 3289248	4682075 3143639	4882021 3282801	4672235 3137996

N	M	$M+T$	$M+D$	$M+T+D$
5 opt proof	<0.1s <0.1s	<0.1s <0.1s	<0.1s <0.1s	<0.1s <0.1s
6 opt proof	0.5s 1.4s	0.2s 1.2s	0.5s 1.5s	0.2s 1.3s
7 opt proof	11s 11s	10s 11s	12s 13s	10s 11s
8 opt proof	6m2s 4m23s	5m48s 4m18s	6m22s 4m40s	6m34s 4m16s

Figure 2: Still-life results (backtracks and CPU time)

3.3 Results and Discussion

We transform the basic model M , with and without the T and D constraints, to linear pseudo-Boolean form and then apply a simple backtracker to solve the problem. Pseudo-Boolean form is a special form of 0/1 integer program that can be solved by SAT-based algorithms (see for example [1]). The backtracker uses a lexicographical variable ordering (except that a variable whose domain size becomes 1 is immediately assigned) and a value ordering that tries 0 before 1. The results are shown in Figure 2: “opt” is the CPU time or backtracks needed to find an optimal solution and “proof” is the time or backtracks to prove that no denser solution exists (restarting the algorithm with a lower bound equal to the best known value plus 1).

In all cases the dominance enforcement constraints reduce or leave unchanged the required number of backtracks. The D constraints result in little reduction in backtracks while incurring a CPU time overhead. The T constraints, on the other hand, reduce both the backtracks and the CPU time. The results are comparable with the basic CP and IP results of [6] but not as good as their hybrid approach nor other state-of-the-art approaches [15; 19]. Though the improvement due to dominance is small we feel that it is worth reporting, because the geometrically-inspired dominances may inspire more effective versions for this or other problems. (In fact we reuse the idea of translational dominance in Section 5.)

4 Steel Mill Slab Design

Our second case study is a simplified industrial problem. In a steel mill, slabs are produced from molten iron in a finite number of sizes which are later cut to fulfil individual orders. Given a set of orders of certain sizes, we must pack the orders onto the slabs while minimizing the total size of the slabs. In addition, each order is assigned a colour, corresponding to a route through the mill. There is a limit, p , on how many

different colours may be assigned to a slab. We denote the S slab sizes by σ_i , the K colours by κ_i , and the O order weights by ω_i .

4.1 A Basic Model

We start from a basic model similar to the IP model of [13], but explicitly model wastage as suggested in [9]. The number of slabs is not known in advance so we assume that there may be as many slabs as orders. A *size variable* s_{ij} is 1 if and only if slab i takes size σ_j . Each slab has up to one size, $\sum_j s_{ij} \leq 1$, and a slab with no size implicitly has size 0, denoting that it is unused. The optimization problem of minimizing the total slab size can be reduced to a series of CSPs with decreasing upper bound U on the size. Each CSP has a total capacity constraint: $\sum_i \sum_j s_{ij} \sigma_j \leq U$. An *order variable* o_{ij} is 1 if and only if order j is assigned to slab i . Each slab capacity must not be exceeded and each order is assigned to exactly one slab:

$$\sum_j o_{ij} \omega_j \leq \sum_k s_{ik} \sigma_k \quad \sum_i o_{ij} = 1$$

A *colour variable* c_{ij} is 1 if and only if colour j is assigned to slab i . (In some instances not all colours are used, so we relabel them to consecutive numbers.) No more than p colours may be assigned to a slab, and if an order is assigned to a slab then so is its colour: $\sum_j c_{ij} \leq p$ and $o_{ij} \leq c_{ik\kappa_j}$. A *wastage variable* e_{il} for $l = 1 \dots B$ where $B = \lceil \log_2(U - \sum_j \omega_j + 1) \rceil$ is such that the wastage for slab i is bounded by $\sum_{l=1}^B 2^{l-1} e_{il}$:

$$\sum_j s_{ij} \sigma_j - \sum_k o_{ik} \omega_k \leq \sum_l 2^{l-1} e_{il}$$

The total wastage is bounded by:

$$\sum_i \sum_l 2^{l-1} e_{il} \leq U - \sum_k \omega_k$$

4.2 Symmetry

When variable sets form matrices, as they do here, a powerful way of breaking symmetry is to impose lexicographical ordering on rows and/or columns [8]. In principle this can be expressed in linear constraints by comparing weighted sums of the form $\sum_i 2^{i-1} x_i$ (assuming binary variables), but the finite word length of a computer makes this impractical for large vectors of variables. Instead we approximately break symmetry by using sums of the form $\sum_i i^2 x_i$ but retain the symmetry breaking ideas of Frisch et al. This technique (which we have not seen used before but do not claim to be original) potentially breaks some symmetry but leaves at least one solution, because at least one weighted sum will be greatest whatever coefficients we choose. But it may not break all symmetries because more than one weighted sum may be equal. In future work we hope to repeat the experiments below using lexicographic ordering.

Orders of the same weight and colour are interchangeable so we approximately order those columns of the order variable matrix: $\sum_i i^2 (o_{ik} - o_{ij}) \geq 0$, where $j < k$, $\kappa_j = \kappa_k$

and $\omega_j = \omega_k$. The constraints

$$\left(\sum_l l^2 \right) \sum_k k^2 (s_{ik} - s_{jk}) + \sum_l l^2 (o_{il} - o_{jl}) \geq 0$$

order slabs by decreasing size, and where two slabs take the same size force their order vectors to be approximately ordered. The o variables are prevented from interfering with the slab size ordering by the coefficient $\sum_l l^2$.

4.3 Implied Constraints

Consider an implied constraint of Frisch et al. If symmetry breaking constraints are used then the first slab must be large enough to be assigned the largest order μ (though it is not necessarily assigned to the first slab). Furthermore, because of the slab size ordering, order μ must be assigned to a slab $1 \dots M$ where $M = \min(O, \lfloor U/\omega_\mu \rfloor)$: $o_{i\mu} = 0$ and $i > M$. The upper bound U on the total capacity can be used in implied constraints:

$$\sum_{k \geq i} \sum_j s_{kj} \sigma_j + (i-1) \sum_j s_{ij} \sigma_j \leq U$$

where $2 \leq i \leq O$. Similar constraints impose a lower bound on the total capacity:

$$\sum_{k \leq i} \sum_j s_{kj} \sigma_j + (O-i) \sum_j s_{ij} \sigma_j \geq \sum_k \omega_k$$

4.4 Dominance

We have found four dominances in the steel mill slab design problem. To the best of our knowledge the fourth is new, but the other three could have been incorporated immediately into the model without considerations of dominance. However, they were not used in previous work on this problem, and we aim to show that thinking in dominance terms can lead to them in a natural way.²

We use a small example of Frisch et al. as an illustration: the available slab sizes are $\{1, 3, 4\}$, the available colours are {red, green, blue, orange, brown} and the input orders are shown in Figure 3. It also shows three optimal solutions, the first taken from Frisch et al.

Colour Dominance. Consider a hypothetical solution (not in Figure 3) in which only two orders are assigned to slab 1: 5,6 which are both orange. Assume that $p = 2$ so that each slab can be assigned up to 2 colours. Then we are free to assign another colour such as blue to slab 1 without violating a colour constraint, even though no blue orders are assigned to it. This is a form of dominance: if a solution exists with orders 5,6 assigned to slab 1 which is assigned the colour orange, then there exists a solution with slab 1 *also* assigned the colour blue. To enforce the dominance we add constraints to exclude states in which a colour is assigned to a slab but

²These dominances may have been made unnecessary by not branching on the related variables, for example by not branching on colour variables we do not encounter colour-dominated solutions. It is an open question which strategy works best, but by leaving the choice of branching variables free we are able to apply a generic solver that does not provide branching control.

order	1	2	3	4	5	6	7	8	9
weight	2	3	1	1	1	1	1	2	1
colour	R	G	G	B	O	O	O	B	B

Solution 1			Solution 2			Solution 3		
slab	size	orders	slab	size	orders	slab	size	orders
1	4	7,8,9	1	3	8,9	1	3	8,9
2	3	1,3	2	3	1,3	2	3	1,3
3	3	2	3	3	2	3	3	2
4	3	4,5,6	4	3	4,5,6	4	1	4
			5	1	7	5	1	7
						6	1	5
						7	1	6

Figure 3: A small example and three optimal solutions

no order of that colour is: $c_{ik} \leq \sum_{j \in S_k} o_{ij}$, where $S_k = \{j \mid \kappa_j = k\}$. The definition of the colour constraint therefore has been changed from an “if” to an “if-and-only-if”.

Wastage Dominance. The \leq -definition of wastage can be transformed to an $=$ -definition by adding inequalities:

$$\sum_j s_{ij} \sigma_j - \sum_k o_{ik} \omega_k \geq \sum_l 2^{l-1} e_{il}$$

Now the e_{il} give the exact wastage for slab i instead of an upper bound. This is a dominance: if a solution exists in which the wastage is greater than necessary, then there also exists one in which it is exactly the total slab size minus the total order weight. This is not a symmetry as we may not be able to increase a slab wastage without violating the upper bound on total wastage.

Capacity Dominance. Consider any solution in which the size of a slab is larger than necessary: that is, it could be reduced to the next smaller size (or even smaller) while still exceeding or equalling the sum of the weights of its assigned orders. The existence of the wasteful solution implies the existence of the better solution (but not vice-versa) so we can add constraints to exclude the former:

$$\sum_{k=2}^O s_{ik} (1 + \sigma_{k-1}) \leq \sum_j o_{ij} \omega_j \quad s_{i1} \leq \sum_j o_{ij}$$

A slab’s size is now a function of its orders: it is the smallest size that is large enough to contain the slab’s orders. If no orders are assigned then a slab has size 0.

Cutting Dominance. The second solution of Figure 3 is derived from the first by cutting slab 1 of size 4 into slab 1 of size 3 and slab 5 of size 1. The third solution is derived from the second by cutting slab 4 of size 3 into slabs 4,6,7 each of size 1. These transformations are reversible and could be treated as a symmetry, giving an opportunity for further symmetry breaking by adding constraints such as: $s_{13} + o_{17} + o_{18} + o_{19} \leq 3$. But we may need to enumerate a large number of assignment-size combinations. This situation can be improved by relaxing the symmetry requirement to a dominance as follows. Consider any solution in which slab 1 has size 4 and is assigned orders $\{7\} \cup S$ for some orders S . Then it can be decomposed into two slabs, one of size

O	U_{opt}	Solver	PBS			
			(a)	(b)	(c)	(d)
12	77	0.3	1.89	0.44	0.4	1.59
16	99	18.9	13.2	4.8	3.81	5.43
18	110	226	948	57.9	48.5	45.4
19	115		1493	36.9	141	7.00
20	122		2683	394	10.5	54.1
21	135		246	801	0.74	1085
25	166		882	18.5	964	31.3
30	195		—	—	37.6	198

Figure 4: Search times in seconds for optimum solutions.

1 assigned order 7, the other of size 3 assigned orders S . This is always possible: no colour constraint can become violated by cutting a slab into two, nor can the total capacity constraint be violated as the capacity is unchanged. The reverse transformation (merging two slabs into one) might violate a colour constraint so this is not a symmetry. This dominance can be enforced by adding a constraint to exclude any solution in which slab 1 has size 4 (size number 3) and is assigned order 7: $s_{13} + o_{17} \leq 1$. This binary constraint subsumes several symmetry breaking constraints of higher arity.

These binary constraints do not enforce all cutting dominances. In the second solution of Figure 3, orders 4,5,6 are assigned to slab 4, which therefore has size $1 + 1 + 1 = 3$. We cannot cut this into two slabs of sizes 1 and 2 because 2 is not a valid slab size. We can cut it into 3 slabs of size 1 as in the third solution. All the cutting dominance constraints can be described by $s_{ij} + \sum_{k \in \Omega} o_{ik} \leq |\Omega|$ where:

- Ω contains orders of no more than p different colours;
- either (i) $\sum_{k \in \Omega} \omega_k = \sigma_j$ and there is a proper partitioning of Ω into subsets each of whose size is a slab weight; or (ii) $\sigma_j - \sum_{k \in \Omega} \omega_k = \sigma_{j'}$ for some j' and there is a partitioning of Ω (possibly the trivial partitioning $\{\Omega\}$) into subsets each of whose size is a slab weight;
- Ω has no proper subset that can be so partitioned (to avoid generating subsumed constraints).

4.5 Results and Discussion

We transform the linear constraint models to pseudo-Boolean form and apply PBS [1] (with VSIDS variable ordering and G=50 as recommended). Using the instances from [9; 13], Figure 4 shows the time in seconds taken to find an optimum solution for ILOG Solver 5.0 on a 750 MHz Pentium III [?]; this has similar performance to our machine so the times are roughly comparable. Case (a) and the Solver results use symmetry breaking and implied constraints, case (b) adds capacity dominance enforcement constraints (implied when U is set to the optimum value), case (c) adds to (b) colour and wastage dominance enforcement constraints, and case (d) adds to (c) cutting dominance enforcement constraints up to arity 3. Times longer than one hour are denoted by “—”. The upper bound U was set to the known optimum value for the problem.

PBS with symmetry breaking and implied constraints is (perhaps surprisingly) not much worse than Solver. Adding

capacity dominance enforcement constraints significantly improves performance. Adding colour and wastage dominance enforcement constraints has a slightly erratic effect but is positive overall, and enables PBS to solve the largest problem. Adding cutting dominance enforcement constraints also has an erratic effect, with improvements on some instances.

Figure 5(i) compares PBS on model (c) with the results of [13] who used a faster 1.17 GHz Pentium III. (The CPLEX results are the best of several models; CPLEX was unable to solve all instances within the time limit with any single model.) They experimented with: IP models implemented in OPL and solved with CPLEX; CP models with two different branching strategies, solved with ILOG Solver 5.2; and hybrid CP/IP models solved by Solver and CPLEX. Both Solver and CPLEX were called via OPL. *Solver model 1* denotes an IP model similar to our model (a), and *model 2* denotes a hybrid model with channelling constraints.

We start PBS with $U = 1000$ instead of ∞ because our model needs a finite upper bound in order to express the wastage constraints (a better method would be to compute an upper bound for U). Each time we find a solution we restart the search with U set to the total slab size of that solution minus 1. There is no other communication between iterations so some search effort is wasted. When starting from a high value of U the best PBS variable ordering heuristic turns out to be a fixed ordering (o, s, c, e) instead of the recommended VSIDS ordering. We now fail to solve the largest instance within the time limit but obtain better results overall. All times include proof of optimality, which is trivial on these instances because each has a “perfect” solution with zero wastage. We use a threshold of 5746 seconds instead of one hour, to allow for our slower machine. Numbers in brackets indicate the best result found within the time limit, while times longer than the threshold are denoted by “—”.

PBS with model (c) now has better overall performance than Solver with either constraint model, though our results are not as good as their CPLEX or hybrid results. Note that dominance enforcement constraints could be added to the hybrid model. (In retrospect it may have been more informative to add dominance in the same models as used by previous researchers.) As noted, for these problem instances the proof of optimality is trivial as they all have perfect solutions. We expect the dominance enforcement constraints to have an impact on the time required to prove optimality as they prune alternative (but not better) solutions. We therefore designed a set of benchmarks without perfect optimum solutions. For a problem of size N , we define a set of N orders with size 2 and colour 1, N orders with size 3 and colour 2, and 1 order of size 4 and colour 3. The available slab sizes are 3 and 6.

Figure 5(ii) shows the optimum total slab size U_{opt} and total order weight $\sum \omega$ for various values of N , with execution times for proofs of optimality ($U = U_{opt} - 1$) under various models. Values of N having perfect solutions are omitted. Cases (a), (c) and (d) are as in Figure 4. The colour, wastage and capacity dominance enforcement constraints make a very large difference to the proofs of optimality.

O	Solver model 1	Solver model 2	CPLEX model 1	Solver+ CPLEX (c)	PBS (c)
12	(79)	0.28	9.90	0.52	1.05
16	(112)	6.51	10.7	0.53	11.9
18	(121)	145	1.98	0.67	118
19	(121)	303	2.6	1.91	78.7
20	(152)	2014	5.91	2.97	509
21	—	21.8	12.0	4.84	46.4
25	—	1216	65.2	7.6	5.1
30	—	(197)	1180	15.9	(196)

N	$\Sigma \omega$	U_{opt}	(a)	(c)	(d)
3	19	21	0.01	0.02	0.02
5	29	30	0.05	0.03	0.03
6	34	36	0.07	0.07	0.05
8	44	45	2.16	0.54	0.2
9	49	51	0.58	0.68	0.14
11	59	60	45.5	6.32	0.69
12	64	66	5.33	5.79	0.39
14	74	75	1777	131	6.7
15	79	81	51.2	55.5	0.99
17	89	90	—	1320	27.1

Figure 5: Slab design results.

5 Peaceable Armies of Queens

For our final case study we consider the problem of placing equally sized armies of black and white queens on an $N \times N$ chess board so that no white queen can attack a black queen (or vice-versa), and to maximize the size of the armies [4]. An IP model was defined by Plastria [5]. Smith et al. [18] defined and tested three constraint models for the problem, breaking symmetry by using SBDS [2; 11].

5.1 A Basic Model

We define a new IP for the problem, drawing on ideas from the constraint models of [18]. Associate a pair of 0/1 variables b_{ij} and w_{ij} with each square. The b_{ij} (resp. w_{ij}) take value 1 if there is a black (resp. white) queen on square (i, j) and 0 otherwise. In addition to these *square variables*, define 0/1 *line variables* b_ℓ and w_ℓ for each line ℓ (row, column or diagonal; one diagonal at each corner contains a single square). The optimization problem can be expressed as a series of CSPs with increasing lower bound Q on the number of black and white queens:

$$\sum_i \sum_j b_{ij} \geq Q \quad \sum_i \sum_j w_{ij} \geq Q$$

Any surplus queens may be removed to obtain a pattern with exactly Q black and Q white queens. The other constraints are as follows. No square or line may be both black and white: $b_{ij} + w_{ij} \leq 1$ and $b_\ell + w_\ell \leq 1$. If a square is black [white] then its four associated line variables are black [white]:

$$4b_{ij} \leq \sum_{\ell \in L_{ij}} b_\ell \quad 4w_{ij} \leq \sum_{\ell \in L_{ij}} w_\ell$$

where L_{ij} denotes the lines passing through square (i, j) . If a line is black [white] then at least one of its associated squares

is black [white]:

$$b_\ell \leq \sum_{(i,j) \in S_\ell} b_{ij} \quad w_\ell \leq \sum_{(i,j) \in S_\ell} w_{ij}$$

where S_ℓ denotes the squares on line ℓ .

5.2 Symmetry

Smith et al. use SBDS to break four rotational symmetries, two reflectional symmetries and one colour symmetry (flip all colours). We add constraints to the model before search, which is weaker than SBDS but has the advantage that it can be used with any search algorithm. First the colour symmetry is broken by insisting that no white queens are placed on the top row: $w_1 = 0$. For the rotational and reflectional symmetries, we consider the eight half-rows and half-columns at the edges of the board as binary representations of integers. We then post constraints to make the black left half of the top row represent the greatest number:

$$\begin{aligned} \sum_{i=1}^{\lfloor N/2 \rfloor} 2^{i-1}(b_{i1} - b_{Ni} - w_{Ni}) &\geq 0 \\ \sum_{i=1}^{\lfloor N/2 \rfloor} 2^{i-1}(b_{i1} - b_{iN} - w_{iN}) &\geq 0 \\ \sum_{i=1}^{\lfloor N/2 \rfloor} 2^{i-1}(b_{i1} - b_{1i} - w_{1i}) &\geq 0 \\ \sum_{i=1}^{\lfloor N/2 \rfloor} 2^{i-1}(b_{i1} - b_{N-i-1N} - w_{N-i-1N}) &\geq 0 \\ \sum_{i=1}^{\lfloor N/2 \rfloor} 2^{i-1}(b_{i1} - b_{1N-i-1} - w_{1N-i-1}) &\geq 0 \\ \sum_{i=1}^{\lfloor N/2 \rfloor} 2^{i-1}(b_{i1} - b_{N-i-11} - w_{N-i-11}) &\geq 0 \\ \sum_{i=1}^{\lfloor N/2 \rfloor} 2^{i-1}(b_{i1} - b_{NN-i-1} - w_{NN-i-1}) &\geq 0 \end{aligned}$$

5.3 Dominance

Suppose that in a solution, queens are *only* placed in a smaller rectangle than $N \times N$, for example rows and columns $2 \dots N$. The solution can still be rotated and reflected within the smaller square, and also translated upward until a queen appears in the top row, and leftwards until a queen appears in the left column. The reverse is not true: a solution with queens in the top row and left column cannot necessarily be translated, because other queens might fall off the bottom row and right column. Thus we have two translational dominances, which can be enforced by insisting that at least one queen be placed in the top row and at least one queen be placed in the left column: $b_1 + w_1 \geq 1$ and $b_{N+1} + w_{N+1} \geq 1$, where line $N+1$ denotes the left column. Combining these constraints with the colour symmetry breaking constraint above, results in the constraint $b_1 = 1$, requiring a black queen in the top row.

Another dominance occurs when placing a queen on a square cannot violate any constraint. There are three cases. Firstly, if a solution exists with an empty square whose lines are uncoloured then a solution also exists with a queen of either colour in that square (recall that in our formulation only lower bounds are placed on the sizes of the armies). We call this the *uncoloured dominance* and it can be enforced by placing a queen in that square; more specifically, a black queen. Secondly, if a solution exists in which an empty square has no white lines and at least one black line passing through it, then a black queen may be placed there. We call this the *black dominance*. Thirdly, if a solution exists in which an empty square has no black lines and at least one white line,

N	Q	pseudo-Boolean			ILOG Solver				
		basic	symm	dom	both	+VO	basic	symm	U+VO
4	2	<0.1	<0.01	<0.1	<0.1	0.01	0.03	0.02	0.02
5	4	0.12	0.02	0.08	0.01	0.02	0.11	0.04	0.03
6	5	1.6	0.15	0.6	0.07	0.08	2.9	0.41	0.13
7	7	29	2.2	6	0.49	0.66	56	7.8	1.12
8	9	236	41	93	5.4	5.25	2100	240	11.7
9	12	26091	1704	1262	93	31			116
10	14				879	298			2460
11	17				12379	2874			37100
12	21					17868			
13	24					247331			

Figure 6: Results for the Peaceable Armies of Queens

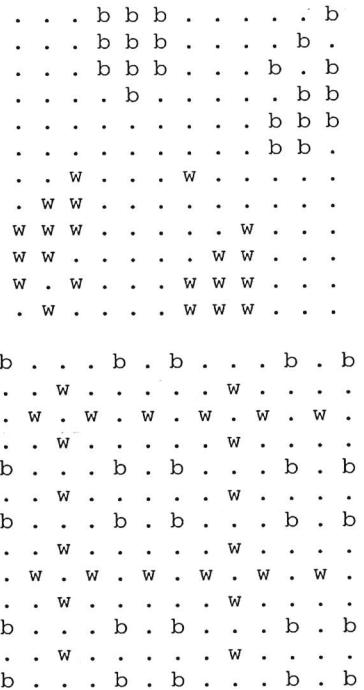
then a white queen may be placed there. This is the *white dominance*. Two families of constraints can be used to enforce these dominances. One enforces both the black and uncoloured and the other enforces the white:

$$b_{ij} + \sum_{\ell \in L_{ij}} w_{\ell} \geq 1 \quad b_{ij} + w_{ij} + \sum_{\ell \in L_{ij}} b_{\ell} \geq 1$$

5.4 Results and Discussion

As above, these models are transformed to pseudo-Boolean form. We apply a naive backtracker using a lexicographical variable ordering: first the square variables row by row, then the line variables (experiments with the much more sophisticated PBS solver [1] surprisingly gave inferior results). Figure 6 shows the CPU times for the algorithms to find and prove optimal solutions for various values of N . It shows results for the basic model (*basic*), the basic model with symmetry breaking (*symm*), with dominance enforcement (*dom*), with both (*both*), and both with a modified (but still static) variable ordering heuristic (*+VO*). The modified variable ordering chooses variables with the highest score $pn + p + n$ where p and n are the number of positive and negative occurrences of the variable, breaking ties lexicographically. The figure also shows results for ILOG Solver with and without SBDS on a basic squares-only model (*basic* and *symm*), and ILOG Solver results with SBDS on the best (*unattacked squares*) model and a *most-unattacked-square* variable ordering heuristic (*U+VO*). The results for Solver are taken from Smith et al. [18] who use a 600 MHz Celeron PC. Q denotes the number of queens.

Both symmetry breaking and dominance enforcement make a significant difference to search time. Though our symmetry breaking approach is simple it achieves speedups comparable to those of SBDS on this problem. Dominance enforcement achieves similar speedups, that increase with N until it gives better results than symmetry breaking (we do not know whether this can be extrapolated to larger instances). Applying both symmetry breaking and dominance enforcement gives even better results than ILOG Solver, and adding the modified variable ordering further reduces execution times so that we are able to increase N by 2. Figure 7 shows the first known optimal solutions, as far as we are aware, for $N = 12$ and $N = 13$. The solution for $N = 13$

Figure 7: Optimal 12×12 and 13×13 solutions.

actually contains 25 black queens and 24 white ones; to obtain a solution with equal-sized armies we can simply remove any black queen.

6 Conclusion

Problem symmetry has recently received a great deal of attention in the literature, and exploiting it has led to huge improvements in search performance. Symmetry can be seen as a special (bi-directional) form of dominance, which is a weaker, more general property. However, dominance has been far less exploited than symmetry in the CP literature. We believe that it should rank alongside symmetry breaking as a generic CP technique, and that it can be profitable to treat both in a uniform way.

To support this thesis we added new dominance enforcement constraints to three well-studied symmetric problems. In the Still-Life and Peaceable Armies of Queens problems we found geometrically-inspired dominances that are analogous to some common symmetries: whereas *rotation* and *reflection* are common symmetries in geometric problems, we found *translational* and pattern-based dominances, leading to improved results on the latter problem. In the Steel Mill Slab Design problem we found a new form of dominance (*cutting*) that improves proofs of optimality. We hope that these examples will serve to stimulate further research on dominance in CP.

It may be that dominance has been relatively unexploited because it is more natural to think in terms of symmetry. However, we found that after a little practice it became quite natural to think in dominance terms, and that it helped to guide the modeling process. In the Steel Mill Slab De-

sign problem we found several dominance enforcement constraints that transform *if* and *less-than-or-equal-to* definitions to tighter *if-and-only-if* and *equals* definitions. Though these could have been exploited without thinking in dominance terms, they were not used by previous researchers and we were led to them by considerations of dominance.

Dominance also led us to an insight on another problem. The best constraint model for the Peaceable Queens problem found by Smith et al. [18] was the *unattacked queens* model. That model represents only the white queens, ensuring that there are at least as many unattacked squares as white queens; black queens are implicitly placed in these squares. The stated advantages of this model are that the search variables have smaller domains and that there are fewer constraints. But another perspective on this model is that by merging the cases of a square containing no queen and a black queen, it implicitly enforces the black and uncoloured (but not the white or translational) dominances. Perhaps part of the advantage of this model can be traced to the absence of these dominances. In retrospect, the unattacked queens model *could have been* inspired by the identification and elimination of dominated solutions. Given that constraint modeling is a poorly understood and difficult process, heuristics that lead to good models are an important research direction.

Acknowledgments

This work was supported in part by the Boole Centre for Research in Informatics, University College, Cork, Ireland, and also by Science Foundation Ireland under Grant 00/PI.1/C075. Thanks to Ken Brown, Ian Gent, Brahim Hnich, Ian Miguel and Barbara Smith for helpful discussions.

References

- [1] F. Aloul, A. Ramani, I. Markov, and K. Sakallah. PBS: A Backtrack Search Pseudo-Boolean Solver. *Symposium on the Theory and Applications of Satisfiability Testing*, 2002.
- [2] R. Backofen, S. Will. Excluding Symmetries in Constraint-Based Search. *Fifth International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* vol. 1713, Springer-Verlag 1999, pp. 73–87.
- [3] R. Bosch. Integer Programming and Conway's Game of Life. *SIAM Review* 41(3), 1999, pp. 594–604.
- [4] R. Bosch. Peaceably Coexisting Armies of Queens. *Optima (Newsletter of the Mathematical Programming Society)* vol. 62 pp. 6–9, 1999.
- [5] R. Bosch. Armies of Queens, Revisited. *Optima (Newsletter of the Mathematical Programming Society)* vol. 64, 2000, p. 15.
- [6] R. Bosch and M. Trick. Constraint Programming and Hybrid Formulations for Life. *Workshop on Modelling and Problem Formulation*, Cyprus, 2001.
- [7] H.-L. Fang. *Genetic Algorithms in Timetabling and Scheduling*. PhD thesis, Department of Artificial Intelligence, University of Edinburgh, 1994.
- [8] P. Flener, A. Frisch, B. Hnich, Z. Kızıltan, I. Miguel, J. Pearson, T. Walsh. Symmetry in Matrix Models. *Workshop on Symmetry in Constraints*, Cyprus, 2001.
- [9] A. M. Frisch, I. Miguel, T. Walsh. Symmetry and Implied Constraints in the Steel Mill Slab Design Problem. *Workshop on Modelling and Problem Formulation*, Cyprus, 2001.
- [10] I. P. Gent, I. McDonald, I. Miguel, B. M. Smith. Approaches to Conditional Symmetry Breaking. *4th International Workshop on Symmetry and Constraint Satisfaction Problems*, Toronto, Canada, 2004.
- [11] I. P. Gent, B. M. Smith. Symmetry Breaking During Search in Constraint Programming. *Fourteenth European Conference on Artificial Intelligence*, Berlin, Germany, 2000, pp. 599–603.
- [12] L. Getoor, G. Ottosson, M. P. J. Fromherz, B. Carlson. Effective Redundant Constraints for Online Scheduling. *Fourteenth National Conference on Artificial Intelligence*, Providence, Rhode Island, 1997, pp. 302–307.
- [13] B. Hnich, Z. Kızıltan, I. Miguel, T. Walsh. Hybrid Modelling for Robust Solving. *Annals of Operations Research* (to appear).
- [14] T. Ibaraki. The Power of Dominance Relations in Branch-and-Bound Algorithms. *Journal of the Association for Computing Machinery*, vol. 24, 1977, ACM Press, pp. 264–279.
- [15] J. Larrosa, E. Morancho. Solving 'Still Life' with Soft Constraints and Bucket Elimination. *Ninth International Conference on Principles and Practice of Constraint Programming*, Kinsale, County Cork, Ireland, 2003, pp. 466–479.
- [16] L. Proll, B. M. Smith. Integer Linear Programming and Constraint Programming Approaches to a Template Design Problem. *INFORMS Journal of Computing* vol. 10, 1998, pp. 265–275.
- [17] J.-F. Puget. On the Satisfiability of Symmetrical Constrained Satisfaction Problems. J. Komorowski, Z. W. Ras (eds.), *Methodologies for Intelligent Systems, International Symposium on Methodologies for Intelligent Systems, Lecture Notes in Computer Science* vol. 689, Springer-Verlag, 1993, pp. 350–361.
- [18] B. M. Smith, K. E. Petrie, I. P. Gent. Models and Symmetry Breaking for Peaceable Armies of Queens. *ECAI Workshop on Modelling and Solving Problems with Constraints*, 2002.
- [19] B. M. Smith. A Dual Graph Translation of a Problem in 'Life'. *Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* vol. 2470, Springer-Verlag, 2002, pp. 402–414.

Breaking symmetries in all different problems

Jean-François Puget

ILOG, 9 avenue de Verdun, 94253 Gentilly, France

Abstract

Adding symmetry breaking constraints is one of the oldest ways of breaking variable symmetries for CSPs. We have established two major results. First of all, all variable symmetries can be broken with at most $n - 1$ binary constraints if all the n variables of a CSP are subject to an all different constraint. Second, symmetry breaking constraints can be safely used together with the GE-tree method of [11]. Both results have been applied to the sub graph isomorphism problem, a prevalent problem in computer science applications. In particular, the SBDD method of [9] solves many SGIP in order to prune search. The symmetry breaking techniques discussed in this paper can be applied to those sub problems. Experiments on BIBD problems show that this is highly effective.

1 Introduction

Adding symmetry breaking constraints is one of the oldest ways of breaking variable symmetries for constraint satisfaction problems (CSPs). For instance, it is shown in [1] that all variable symmetries could be broken by adding one lexicographical ordering constraint per symmetry. Unfortunately, this method is not tractable in general, as there may be an exponential number of symmetries. It has been shown that in general there is no way to break all symmetries of a problem with a polynomial number of constraints[12]. In [2], a linear number of constraints are used to break symmetries for matrix problems. As expected, since there are a polynomial number of constraints, not all symmetries are broken. However, a polynomial number of constraints may be sufficient in some special cases. For instance, in [8], we have shown that when the symmetry group is the full group, then a linear number of constraints can break all symmetries: one simply needs to order the variables. In this paper we consider *all different problems*. These are CSPs such that the variables are subject to an all different constraint among other constraints. We show in section 3 that for such CSPs, all variable symmetries can

be broken with at most $n - 1$ binary constraints, where n is the number of variables.

In [11] a general purpose method for breaking all value symmetries is given: the GE-tree method. We show in section 4 that this method can be safely combined with symmetry breaking constraints, under some conditions on the order in which the search tree is traversed.

In section 5, we apply the previous results to a prevalent problem in computer science applications, namely the sub graph isomorphism (SGI) problem. In section 6, we report various experiments using a variant of SBDD where the dominance test amounts to solve a SGI problem. These experiments show that the overhead of computing graph automorphism is more than offset by the speedups resulting from symmetry breaking. In section 7, we summarize our findings and discuss some possible generalizations.

2 Symmetries, Graphs and CSPs

The symmetries we consider are permutations, i.e. one to one mappings (bijections) from a finite set onto itself. Without loss of generality, we can consider permutations of I^n , where I^n is the set of integers ranging from 1 to n . For instance, we can label the vertices of a graph with integers, such that any graph automorphism is completely described by a permutation of the labels of its vertices. Similarly, any variable symmetry in a CSP can be described by a permutation of the indices of the variables. This is formalized as follows.

2.1 Automorphism groups

Let S^n be the set of all permutations of the set I^n . The image of i by the permutation σ is denoted i^σ . A permutation $\sigma \in S^n$ is fully described by the vector $[1^\sigma, 2^\sigma, \dots, n^\sigma]$. The product of two permutations σ and θ is defined by $i^{(\sigma\theta)} = (i^\sigma)^\theta$.

Given $i \in I^n$ and a permutation group $G \subseteq S^n$, the orbit of i in G , denoted i^G , is the set of elements to which i can be mapped to by an element of G :

$$i^G = \{i^\sigma | \sigma \in G\}$$

Given $i \in I^n$ and a permutation group $G \subseteq S^n$, the stabilizer of i in G , denoted i_G , is the set of permutations of G that leave i unchanged:

$$i_G = \{\sigma \in G \mid i^\sigma = i\}$$

2.2 Graphs

A graph is a pair (V, E) where V is a finite set of vertices, and E a set of edges between these vertices. An edge is a set of two vertices.

Given a graph (V, E) , we can without loss of generality assume that the set V is equal to I^n where n is the number of vertices of the graph. Any permutation σ of the vertices induces a permutations of the edges : the image of the edge $\{i, j\}$ by σ is $\{i^\sigma, j^\sigma\}$.

An automorphism of the graph (V, E) is a permutation σ of V such that

$$\forall e \in E, e^\sigma \in E$$

The set of automorphism of a graph G is noted $aut(G)$. This set is a subgroup of S^n . Graph automorphism can be computed using packages such as Nauty[6].

The SGI problem can be defined as follows. We are given two graphs, $\mathcal{G}_1 = (V_1, E_1)$ and $\mathcal{G}_2 = (V_2, E_2)$, and we want to know if \mathcal{G}_1 is isomorphic to a sub graph of \mathcal{G}_2 , i.e. if there exists a function f from V_1 to V_2 such that

$$\forall i, j \in V_1, i \neq j \rightarrow f(i) \neq f(j)$$

$$\forall e \in E_1, f(e) \in E_2$$

$$\text{where } f(\{i, j\}) = \{f(i), f(j)\}$$

2.3 CSP and symmetries

A constraint satisfaction problem \mathcal{P} (CSP) with n variables is a triple $\mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$ where \mathcal{V} is a finite set of variables $(v_i)_{i \in I^n}$, \mathcal{D} a finite set of finite sets $(\mathcal{D}_i)_{i \in I^n}$, and \mathcal{C} is a subset of the cross product $\bigotimes_{i \in I^n} \mathcal{D}_i$. Without loss of generality, we can assume that $\mathcal{D}_i \subseteq I^k$ for some k .

An *assignment* is a member of \mathcal{S} , i.e. a vector of values $(a_i)_{i \in I^n}$ such that $a_i \in \mathcal{D}_i$ for all $i \in I^n$, and is denoted $(v_i = a_i)_{i \in I^n}$. A *partial assignment* is sub vector of an assignment.

A *solution* to $(\mathcal{V}, \mathcal{D}, \mathcal{C})$ is an assignment that is a member of \mathcal{C} .

Given a permutation σ of I^n , we define a variable permutation on (partial) assignments as follows:

$$((v_i = a_i)_{i \in I^n})^\sigma = (v_{i^\sigma} = a_{i^\sigma})_{i \in I^n}$$

Such permutation is called a *variable symmetry* if it maps solutions to solutions.

Given a permutation θ of I^k , we define a value permutation on (partial) assignments as follow:

$$((v_i = a_i)_{i \in I^n})^\theta = (v_i = a_i^\theta)_{i \in I^n}$$

Such permutation is called a *value symmetry* if it maps solutions to solutions.

3 Breaking variable symmetries

Without loss of generality, we can assume that domains are subsets of I^k for some k , with the usual ordering on integers.

3.1 Lex leader constraints

Adding constraints is one of the oldest methods for reducing the number of variable symmetries of a CSP[8]. In [1], it is shown that all the variable symmetries of any CSP can be broken by the following constraints.

$$\forall \sigma \in G, \mathcal{V} \preceq \mathcal{V}^\sigma \quad (1)$$

For a given σ , the constraint $(\mathcal{V} \preceq \mathcal{V}^\sigma)$ is semantically equivalent to the disjunction of the constraints:

$$\begin{aligned} & v_1 < v_{1^\sigma} \\ & v_1 = v_{1^\sigma} \wedge v_2 < v_{2^\sigma} \\ & \vdots \\ & v_1 = v_{1^\sigma} \wedge \dots \wedge v_{i-1} = v_{(i-1)^\sigma} \wedge v_i < v_{i^\sigma} \\ & \vdots \\ & v_1 = v_{1^\sigma} \wedge \dots \wedge v_{n-1} = v_{(n-1)^\sigma} \wedge v_n < v_{n^\sigma} \\ & v_1 = v_{1^\sigma} \wedge \dots \wedge v_{n-1} = v_{(n-1)^\sigma} \wedge v_n = v_{n^\sigma} \end{aligned}$$

If the last constraint is omitted, the set of constraints is denoted $\mathcal{V} \prec \mathcal{V}^\sigma$.

3.2 A polynomial number of constraints

The number of constraints (1) can grow exponentially with the size of G . Using the fact that the variables \mathcal{V} are subject to an all different constraint, we can significantly reduce the number of symmetry breaking constraints.

Given a permutation σ , let $s(\sigma)$ be the smallest i such that $i^\sigma \neq i$, and let $r(\sigma)$ be equal to $(s(\sigma))^\sigma$. By definition $k^\sigma = k$ for all $k < s(\sigma)$, and $s(\sigma)^\sigma \neq s(\sigma)$. Let us now look at the constraint $\mathcal{V} \preceq \mathcal{V}^\sigma$. There is an all different constraint on the variables \mathcal{V} , which means that $v_i = v_{i^\sigma}$ if and only if $i^\sigma = i$. In particular, $v_k = v_{k^\sigma}$ for all $k < s(\sigma)$, and $v_{s(\sigma)} \neq v_{(s(\sigma))^\sigma}$. Therefore, only one disjunct for the constraint can be true, namely:

$$v_1 = v_{1^\sigma} \wedge \dots \wedge v_{s(\sigma)-1} = v_{(s(\sigma)-1)^\sigma} \wedge v_{s(\sigma)} < v_{(s(\sigma))^\sigma}$$

Since $k^\sigma = k$ for $k < s(\sigma)$ and $s(\sigma)^\sigma = r(\sigma)$, this can be simplified into

$$v_{s(\sigma)} < v_{r(\sigma)}$$

We have just proved the following result.

Lemma 1. Given a CSP where the variables \mathcal{V} are subject to an all different constraint, and a variable symmetry group G for this CSP, then all variable symmetries can be broken by adding the following constraints:

$$\forall \sigma \in G, v_{s(\sigma)} < v_{r(\sigma)} \quad (2)$$

Note that if two permutations σ and θ are such that $s(\sigma) = s(\theta)$ and $r(\sigma) = r(\theta)$, then the corresponding symmetry breaking constraints are identical. Therefore, it is sufficient to state only one symmetry breaking constraints for each pair i, j such that there exists a permutation σ with $i = s(\sigma)$ and $j = r(\sigma)$.

The set of these pairs can be computed using what is known as the Schreier Sims algorithm[13]. This algorithm constructs a stabilizers chain G_0, G_1, \dots, G_n as follows:

$$G_0 = G$$

$$\forall i \in I^n, G_i = i_{G_{i-1}}$$

By definition,

$$G_n \subseteq G_{n-1} \subseteq \dots G_1 \subseteq G_0$$

The Schreier Sims algorithm also computes set of coset representatives U_i . Those are orbits of i in G_{i-1} :

$$U_i = i^{G_{i-1}}$$

From now on, we will assume that all the groups we use are described by a stabilizers chain and coset representatives.

By definition, for each element $j \in U_i$, there exists at least one permutation $\sigma \in G_{i-1}$ such that $i^\sigma = j$ and $j = r(\sigma)$. The converse is also true. If there exists a permutation σ such that $i = s(\sigma)$ and that $j = r(\sigma)$, then $j \in U_i$. Therefore, the constraints (2) can be rewritten into:

$$\forall i \in I^n, \forall j \in U_i, i \neq j \Rightarrow v_i < v_j$$

There are $\sum_{i=1}^n (|U_i| - 1)$ such constraints. All the permutations of G_{i-1} leave the numbers $1, \dots, i-1$ unchanged. Therefore U_i is a subset of $\{i, \dots, n\}$. Then $|U_i| - 1 \leq n - i$. Therefore, the number of constraints is bounded from above by $\sum_{i=1}^n (n - i) = n(n - 1)/2$. We have just proved the following result¹.

Theorem 2. *Given a CSP with n variables \mathcal{V} such that there exists an all different constraint on these variables, and given coset representatives sets U_i for the variable symmetry group of the CSP, then all the variable symmetries can be broken by at most $n(n - 1)/2$ binary constraints. These constraints are given by :*

$$\forall i \in I^n, \forall j \in U_i, i \neq j \rightarrow v_i < v_j \quad (3)$$

3.3 A linear number of constraints

The previous result can be improved by taking into account the transitivity of the $<$ constraints. Given $j \in I^n$, it may be the case that j belongs to several of the sets U_i . In such case, let us define $r(j)$ as the largest i different from j such that j belongs to U_i . If j belongs to no U_i other than U_j , then let $r(j) = j$.

Before stating our main result, let us prove the following.

Lemma 3. *With the above notations, if $j \in U_i$ and $i \neq j$ then $r(j) \in U_i$ and $r(j) < j$*

¹A reviewer tells me that Chris Jefferson independently discovered this result without publishing it.

Proof. Let us assume that $j \in U_i$ and $i \neq j$. By definition of U_i there exists a permutation $\sigma \in G_{i-1}$ such that $i^\sigma = j$. Let $k = r(j)$. By definition of $r(j)$, $i \leq k$ and $j \in U_k$. Therefore, there exists a permutation $\theta \in G_{k-1}$ such that $k^\theta = j$. Let $\nu = \sigma\theta^{-1}$. Then, $i^\nu = i^{\sigma\theta^{-1}} = j^{\theta^{-1}} = k$. Moreover, $\nu \in G_{i-1}$ because $\sigma \in G_{i-1}$ and $\theta \in G_{k-1} \subseteq G_{i-1}$. Therefore, $k \in U_i$. The fact that $r(j) < j$ is an immediate consequence of the definition of $r(j)$.

We can now state our main result.

Theorem 4. *With the above notations, given a CSP with n variables \mathcal{V} , such that there exists an all different constraint on these variables, then all variable symmetries can be broken by at most $n - 1$ binary constraints. These constraints are given by :*

$$\forall j \in I^n, r(j) \neq j \rightarrow v_{r(j)} < v_j \quad (4)$$

Proof. The number of constraints (4) is at most n by definition. Note that $r(1) = 1$ by definition of r , therefore, the number of constraints is at most $n - 1$. Let us consider one of the constraints of (3). We are given i and j such that $j \in U_i$ and $i \neq j$. We want to prove that the constraint $c = (v_i < v_j)$ is implied by the constraints (4). Let us consider the sequence $(j, r(j), r(r(j)), r(r(r(j))), \dots)$. Let us assume that the sequence never meets i . We have that $j \in U_i$ and $i \neq j$. By application of lemma 3, we get $r(j) \in U_i$ and $r(j) < j$. Since $r(j) \neq i$ by hypothesis, lemma 3 can be applied again. By repeated applications of lemma 3 we construct an infinite decreasing sequence of integers all included in U_i . This is not possible as U_i is finite. Therefore, there exists k such that $i = r^k(j)$. Moreover, we have established $r^k(j) \neq r^{k-1}(j), \dots, r(r(j)) \neq r(j), r(j) \neq j$. Therefore, the constraints $v_{r^k(j)} < v_{r^{k-1}(j)}, \dots, v_{r(r(j))} < v_{r(j)}, v_{r(j)} < v_j$ are constraints of (4). Together they imply $v_{r^k(j)} < v_j$ which is the constraint c . We have proved that the constraints (3) are implied by the constraints (4). Since the set of constraints (4) is a subset of the constraints (3), both sets of constraints are equivalent. Then, by theorem 2, the constraints (4) break all variable symmetries.

3.4 Graceful graphs

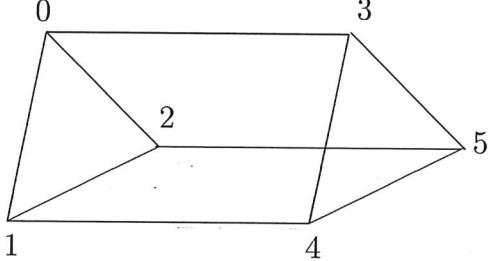
A graph with m edges is *graceful* if there exists a labeling f of its vertices such that:

- $0 \leq f(i) \leq m$ for each vertex i ,
- the set of values $f(i)$ are all different,
- the set values $|f(i), f(j)|$ for each edge (i, j) are all different.

A straightforward translation into a CSP exists where there is a variable v_i for each vertex v_i , see [5]. The variable symmetries of the problem are induced by the

automorphism of the graph. There is one value symmetry, which maps v to $m - v$. More information on symmetries in graceful graphs is available in [7].

Theorem 4 then states that all the variables symmetries can be broken by $n - 1$ binary constraints at most, where n is the number of vertices. Let us consider the following graph $K_3 \times P_2$ ²:



The group of variable symmetries of the corresponding CSP is equivalent to the automorphism of the graph. This group G is:

$$\begin{aligned} & \{[0, 1, 2, 3, 4, 5], [0, 2, 1, 3, 5, 4], [1, 0, 2, 4, 3, 5], \\ & [1, 2, 0, 4, 5, 3], [2, 0, 1, 5, 3, 4], [2, 1, 0, 5, 4, 3], \\ & [3, 4, 5, 0, 1, 2], [3, 5, 4, 0, 2, 1], [4, 3, 5, 1, 0, 2], \\ & [4, 5, 3, 1, 2, 0], [5, 3, 4, 2, 0, 1], [5, 4, 3, 2, 1, 0]\} \end{aligned}$$

Therefore the constraints given by [1] are

$$\begin{aligned} & (v_0, v_1, v_2, v_3, v_4, v_5) \preceq (v_0, v_1, v_2, v_3, v_4, v_5) \\ & (v_0, v_1, v_2, v_3, v_4, v_5) \preceq (v_0, v_2, v_1, v_3, v_5, v_4) \\ & (v_0, v_1, v_2, v_3, v_4, v_5) \preceq (v_1, v_0, v_2, v_4, v_3, v_5) \\ & (v_0, v_1, v_2, v_3, v_4, v_5) \preceq (v_1, v_2, v_0, v_4, v_5, v_3) \\ & (v_0, v_1, v_2, v_3, v_4, v_5) \preceq (v_2, v_0, v_1, v_5, v_3, v_4) \\ & (v_0, v_1, v_2, v_3, v_4, v_5) \preceq (v_2, v_1, v_0, v_5, v_4, v_3) \\ & (v_0, v_1, v_2, v_3, v_4, v_5) \preceq (v_3, v_4, v_5, v_0, v_1, v_2) \\ & (v_0, v_1, v_2, v_3, v_4, v_5) \preceq (v_3, v_5, v_4, v_0, v_2, v_1) \\ & (v_0, v_1, v_2, v_3, v_4, v_5) \preceq (v_4, v_3, v_5, v_1, v_0, v_2) \\ & (v_0, v_1, v_2, v_3, v_4, v_5) \preceq (v_4, v_5, v_3, v_1, v_2, v_0) \\ & (v_0, v_1, v_2, v_3, v_4, v_5) \preceq (v_5, v_3, v_4, v_2, v_0, v_1) \\ & (v_0, v_1, v_2, v_3, v_4, v_5) \preceq (v_5, v_4, v_3, v_2, v_1, v_0) \end{aligned}$$

The stabilizer chain is

$$\begin{aligned} G_0 &= G \\ G_1 &= 0_{G_0} = \{[0, 1, 2, 3, 4, 5], [0, 2, 1, 3, 5, 4]\} \\ G_2 &= 1_{G_1} = \{[0, 1, 2, 3, 4, 5]\} \end{aligned}$$

All remaining stabilizers G_3, G_4, G_5 are equal to G_2 . Coset representatives are:

$$\begin{aligned} U_1 &= 0^{G_0} = \{0, 1, 2, 3, 4, 5\} \\ U_2 &= 1^{G_1} = \{1, 2\} \\ U_3 &= 2^{G_2} = \{2\} \end{aligned}$$

²Vertices numbers start from 0 instead of 1 in this example.

All remaining coset representatives U_4, U_5 are equal to U_3 .

Therefore, the constraints (3) given by theorem 2 are:

$$\begin{aligned} v_0 &< v_1 \\ v_0 &< v_2 \\ v_0 &< v_3 \\ v_0 &< v_4 \\ v_0 &< v_5 \\ v_1 &< v_2 \end{aligned}$$

Those constraints are much simpler and less numerous than the previous ones.

From coset representatives we get:

$$\begin{aligned} r(0) &= 0 \\ r(1) &= 0 \\ r(2) &= 1 \\ r(3) &= 0 \\ r(4) &= 0 \\ r(5) &= 0 \end{aligned}$$

Therefore, the constraints (4) given by theorem 4 are:

$$\begin{aligned} v_0 &< v_1 \\ v_0 &< v_3 \\ v_0 &< v_4 \\ v_0 &< v_5 \\ v_1 &< v_2 \end{aligned}$$

We see that the constraint $v_0 < v_2$ is implied by the above constraints.

These constraints can be automatically added to the problem. Indeed, we have implemented an algorithm similar to Nauty[6] for computing graph automorphisms, as well as a Schreier Sims algorithm[13].

3.5 The pigeon hole problem

Let us look at another example, namely the pigeon hole problem. We are given n variables $(v_i)_{i \in I^n}$ with domains equal to I^{n-1} . There is an all different constraint on these variables. The group of variable symmetries for this problem is the set of all permutations S^n . The stabilizers chain is:

$$G_0 = S^n \quad \forall i \in I^n, G_i = \{\sigma \in S^n \mid \forall k \leq i, k^\sigma = i\}$$

The coset representatives are given by:

$$\forall i \in I^n, U_i = \{i, i+1, \dots, n\}$$

From this we get:

$$r(1) = 1, r(2) = 1, \dots, r(i) = i-1, \dots, r(n) = n-1$$

The symmetry breaking constraints (4) are therefore:

$$v_1 < v_2, v_2 < v_3, \dots, v_{i-1} < v_i, \dots, v_{n-1} < v_n$$

Bound propagation on these constraints is sufficient to show that the CSP has no solution. Note that we only used $n-1$ constraints, whereas (1) contains an exponential number of constraints in this case.

4 Breaking both variable symmetries and value symmetries

In [11], a general method for breaking all value symmetries is described. This method uses the group of value symmetries of the CSP. We will show that this method can be combined with symmetry breaking constraints when there are both variable symmetries and value symmetries.

4.1 GE-tree and symmetry breaking constraints

We are given a CSP \mathcal{P} with n variables v_i subject to an all different constraint among other constraints. Without loss of generality, we can assume that the domains of the variables are subsets of I^k for some k . Let us add $n \times k$ additional binary variables x_{ij} (variables with domains equal to $\{0, 1\}$). We also add the following channelling constraints:

$$\forall i \in I^n, j \in I^k, (x_{ij} = 1) \equiv (v_i = j)$$

It should be clear that the new CSP \mathcal{P}' is equivalent to \mathcal{P} , i.e. that there is a one to one mapping between the solutions of the two CSPs. Let us assume that the variables x_{ij} are ranked in increasing values of i then increasing values of j in the vector X .

Any variable symmetry σ of P is now a permutation of the rows of the matrix (x_{ij}) . From [1], this variable symmetry is broken by the constraints:

$$X \preceq X^\sigma$$

that is,

$$(x_{11}, x_{12}, \dots, x_{nk}) \preceq (x_{1^\sigma 1}, x_{1^\sigma 2}, \dots, x_{n^\sigma k}) \quad (5)$$

Let us compare lexicographically the first k variables in both sides of the constraint. We have the two vectors:

$$\begin{aligned} & (x_{11}, x_{12}, \dots, x_{1k}) \\ & (x_{1^\sigma 1}, x_{1^\sigma 2}, \dots, x_{1^\sigma k}) \end{aligned}$$

Let a be the value assigned to v_1 , and b be the value assigned to v_{1^σ} in a given solution. Then $x_{1a} = 1$ and $x_{1j} = 0$ for $j \neq a$. Similarly, $x_{1^\sigma b} = 1$ and $x_{1^\sigma j} = 0$ for $j \neq b$. Then the first vector is lexicographically smaller than the second one if and only if $a \leq b$. This is equivalent to the condition $v_1 \leq v_{1^\sigma}$. By repeated applications of a similar argument, once for every row of the x_{ij} matrix, we prove the following result.

Lemma 5. *With the above notations, the constraint (5) is equivalent to:*

$$(v_1, v_2, \dots, v_n) \preceq (v_{1^\sigma}, v_{2^\sigma}, \dots, v_{n^\sigma})$$

Let us consider a value symmetry θ for \mathcal{P} . Then θ is a permutation of the columns of the matrix. This symmetry is broken by the constraint:

$$X \preceq X^\theta$$

that is

$$(x_{11}, x_{12}, \dots, x_{nk}) \preceq (x_{11^\theta}, x_{12^\theta}, \dots, x_{nk^\theta}) \quad (6)$$

Let us compare lexicographically the first k variables in both sides of the constraint. We have the two vectors:

$$\begin{aligned} & (x_{11}, x_{12}, \dots, x_{1k}) \\ & (x_{11^\theta}, x_{12^\theta}, \dots, x_{1k^\theta}) \end{aligned}$$

Let a be the value assigned to v_1 . Then $x_{1a} = 1$ and $x_{1j} = 0$ for $j \neq a$. Similarly, $x_{1j^\theta} = 1$ if and only if $j = a^{\theta^{-1}}$. Therefore, the following holds if and only if $a \leq a^{\theta^{-1}}$

$$(x_{11}, x_{12}, \dots, x_{1k}) \preceq (x_{11^\theta}, x_{12^\theta}, \dots, x_{1k^\theta})$$

This must be true for all possible θ . This is true if and only if a is the minimum of its orbit in G , where G is the group of value symmetries for the CSP:

$$a = \min(a^G) \quad (7)$$

Let us now consider the second group of k variables on both sides:

$$\begin{aligned} & (x_{21}, x_{22}, \dots, x_{2k}) \\ & (x_{21^\theta}, x_{22^\theta}, \dots, x_{2k^\theta}) \end{aligned}$$

Let b be the value assigned to v_2 . Then, the following holds if and only if $b \leq b^{\theta^{-1}}$:

$$(x_{21}, x_{22}, \dots, x_{2k}) \preceq (x_{21^\theta}, x_{22^\theta}, \dots, x_{2k^\theta})$$

Then, let us consider the first $2k$ variables on each side altogether. We have that

$$(x_{11}, x_{12}, \dots, x_{2k}) \preceq (x_{11^\theta}, x_{12^\theta}, \dots, x_{2k^\theta})$$

in exactly one of the following two cases. The first case is:

$$(x_{11}, x_{12}, \dots, x_{1k}) \prec (x_{11^\theta}, x_{12^\theta}, \dots, x_{1k^\theta})$$

The second case is :

$$(x_{11}, x_{12}, \dots, x_{1k}) = (x_{11^\theta}, x_{12^\theta}, \dots, x_{1k^\theta})$$

and

$$(x_{21}, x_{22}, \dots, x_{2k}) \preceq (x_{21^\theta}, x_{22^\theta}, \dots, x_{2k^\theta})$$

The first case if equivalent to $a < a^{\theta^{-1}}$. The second case is equivalent to $a = a^{\theta^{-1}}$ and $b \leq b^{\theta^{-1}}$. The condition $a = a^{\theta^{-1}}$ is equivalent to $\theta \in a_G$. When considering all possible θ , we get the following conditions:

$$\forall \theta \in a_G, b \leq b^{\theta^{-1}}$$

This means that b must be the minimum of its orbit in a_G when $b \neq a$. Note that if $b = a$, b is also the minimum of its orbit in a_G , because its orbit is then $\{a\}$. We have proved that

$$(x_{11}, x_{12}, \dots, x_{2k}) \preceq (x_{11^\theta}, x_{12^\theta}, \dots, x_{2k^\theta})$$

holds if and only if the following holds: a is the minimum of its orbit in G , and b is the minimum of its orbit in a_G .

More generally, let a_i be the value assigned to the variable v_i in a solution to the CSP. By repeating the above argument with the first i rows of the x_{ij} matrix, we get the following result.

Lemma 6. *With the above notations, a_i is the minimum of its orbit in the group of symmetries that leave a_1, a_2, \dots, a_{i-1} unchanged.*

This is equivalent to the GE-tree method for breaking all value symmetries [11], when the variables and the values are tried in an increasing order during search.

From [1], it is safe to add all possible symmetry breaking constraints (1). In particular, it is safe to state all the constraints (5) and all the constraints (6) together. By lemma 5, the set of constraints (5) is equivalent to all the symmetry breaking constraints for \mathcal{P} . By lemma 6, the set of constraints (6) is equivalent to the GE-tree method for breaking value symmetries. We have just proved the following result.

Theorem 7. *Given a CSP, its group of variable symmetries G_1 , and its group of value symmetries G_2 , then the combination of the GE-tree method for breaking value symmetries with the symmetry breaking constraints (1) computes a set of solutions \mathcal{S} such that:*

$$\forall S \in \text{sol}(\mathcal{P}), \exists \sigma \in G_1, \exists \theta \in G_2, \exists S' \in \mathcal{S}, S^{\sigma\theta} = S'$$

Theorem 4 in section 3 says that the set of all those constraints (1) is equivalent to the constraints (4) when there is an all different constraint on all the variables \mathcal{V} . This yields the following result.

Corollary 8. *Given a CSP where the variable are subject to an all different constraint, its group of variable symmetries G_1 , and its group of value symmetries G_2 , then the combination of the GE-tree method for breaking value symmetries with the symmetry breaking constraints (4) computes a set of solutions \mathcal{S} such that:*

$$\forall S \in \text{sol}(\mathcal{P}), \exists \sigma \in G_1, \exists \theta \in G_2, \exists S' \in \mathcal{S}, S^{\sigma\theta} = S'$$

In our implementation, we did not fully implement the GE-tree method, because it requires more computational group algorithms than what we have implemented so far. We simply use equation (7). This requires the computation of the orbits for the group G of value symmetries. Then, only the minimum element of each orbit is left in the domain of the variable v_1 .

4.2 Graceful graphs

Let us go back to our example. There are 9 edges. There is one non trivial symmetry, which maps a to $9 - a$. Therefore, the orbits for this group are the sets $\{a, 9-a\}$, for $0 \leq a \leq 4$.

Therefore, one can restrict the domain of v_0 to $\{0, 1, 2, 3, 4\}$. Together with the constraints given by theorem 4, breaking all variables and all value symmetries can be done with the addition of the following constraints

$$v_0 < v_1$$

$$v_0 < v_3$$

$$v_0 < v_4$$

$$v_0 < v_5$$

$$v_1 < v_2$$

$$v_0 \leq 4$$

We have tested this approach on the graceful graphs of [7]. For each graph we report the number of solutions of the CSP (sol), the size of the search tree (node) and the time (time) needed to compute all these solutions the running time. We also report these figures when the above symmetry breaking constraints are added (sym). In this case the running time includes the time needed to perform all the group computations. It is worth noting that the running times are much better than the ones reported in [7]. Running times are measured on a 1.4 GHz Dell Latitude D800 laptop running Windows XP. The implementation is done with ILOG Solver 6.0[4].

graph	no sym			sym		
	sol	node	time	sol	node	time
$K_3 \times P_2$	96	1518	0.12	8	83	0.01
$K_4 \times P_2$	1440	216781	13.6	30	1863	0.27
$K_5 \times P_2$	480	34931511	4454	2	53266	6.5
$K_6 \times P_2$				0	1326585	305

Table 1. Computing all solutions for graceful graphs.

Let us look at the graph $K_5 \times P_2$. This graph has 10 vertices and 25 edges. We list the values for the variables v_0, v_1, \dots, v_9 for the two solutions:

$$(0, 4, 18, 19, 25, 23, 14, 6, 3, 1)$$

$$(0, 6, 7, 21, 25, 24, 22, 19, 11, 2)$$

Let us apply the non trivial value symmetry to the second one. We get:

$$(25, 19, 18, 4, 0, 1, 3, 6, 14, 23)$$

Let us apply the following variable symmetry to it:

$$[4, 3, 2, 1, 0, 9, 8, 7, 6, 5]$$

This yields the first solution!

This example shows that we did not break all symmetries that are a product of one variable symmetry with one value symmetry. This is so despite the fact that all variable symmetries and all value symmetries are broken.

5 Sub graph isomorphism

The previous results can be applied to a prevalent problem in computer science applications, namely sub graph isomorphism (SGI). See [10] for applications in chemistry for instance. The SGI problem is easily cast into a CSP. We are given two graphs, \mathcal{G}_1 and \mathcal{G}_2 and we want to know if \mathcal{G}_1 is isomorphic to a sub graph of \mathcal{G}_2 . A CSP is constructed where there is one variable v_i per vertex i of \mathcal{G}_1 , and where the possible values are the vertices of \mathcal{G}_2 . The constraints of the CSP express the isomorphism relationship. First of all, the variable must be all different. Second, for each edge $\{i, j\} \in E_1$ a binary constraint states that the edge $\{v_i, v_j\}$ is in E_2 . Several global constraints have been devised for enforcing this relationship, see [10][14][9] for instance. The purpose of this paper is not to discuss how to efficiently enforce the isomorphism relationship. It is rather to look at the symmetries in this CSP. They are induced by the automorphisms of the graphs \mathcal{G}_1 and \mathcal{G}_2 as follows.

Let $\sigma \in \text{aut}(\mathcal{G}_1)$ be an automorphism of \mathcal{G}_1 . Then σ induces a variable symmetry. Indeed, let us assume that we have an isomorphism f between \mathcal{G}_1 and a sub graph of \mathcal{G}_2 . Then, $(v_i = f(i))_{i \in I^n}$ is a solution of the CSP. Let us apply σ to this solution. We get the following assignment:

$$(v_{i^\sigma} = f(i))_{i \in I^n} \quad (8)$$

Let us consider an edge e of \mathcal{G}_1 . By automorphism of σ , there exists an edge $e' = \{i, j\}$ of \mathcal{G}_1 such that $e = e'^\sigma$, i.e. $e = \{i^\sigma, j^\sigma\}$. By isomorphism of f , $f(e') = \{f(i), f(j)\}$ is an edge of \mathcal{G}_2 . We have proved:

$$\forall i, j \in V_1, \{i^\sigma, j^\sigma\} \in E_1 \Rightarrow \{f(i), f(j)\} \in E_2$$

This means that the assignment (8) is a solution of the CSP. This proves that σ is a variable symmetry.

Similarly, let $\theta \in \text{aut}(\mathcal{G}_2)$ be an automorphism of \mathcal{G}_2 . Then θ induces a value symmetry. Indeed, let us apply θ to the solution $(v_i = f(i))_{i \in I^n}$ of the SGI CSP. We get the following assignment:

$$(v_i = f(i)^\theta)_{i \in I^n} \quad (9)$$

Let us consider an edge $e = \{i, j\}$ of \mathcal{G}_1 . By isomorphism of f , $f(e) = \{f(i), f(j)\}$ is an edge of \mathcal{G}_2 . By automorphism of θ , $f(e)^\theta = \{f(i)^\theta, f(j)^\theta\}$ is also an edge of \mathcal{G}_2 . We have proved:

$$\forall i, j \in V_1, \{i, j\} \in E_1 \Rightarrow \{f(i)^\theta, f(j)^\theta\} \in E_2$$

This means that the assignment (9) is a solution for the CSP. This proves that θ is a value symmetry.

Corollary 8 states that all the variable symmetries, and all the values symmetries can be broken by $|V_1| - 1$ binary constraints together with the use of the GE-tree method. An implementation of these symmetry breaking constraint requires an algorithm for computing the automorphism groups of \mathcal{G}_1 and \mathcal{G}_2 .

6 SBDD with sub graph isomorphism

The SBDD method of [9] relies on SGI sub problems in order to prune search. The symmetry breaking techniques discussed in this paper can be applied to those sub problems. We chose to evaluate our results on the balanced incomplete designs (BIBD) (problem 28 in the CSPLib [3]). A BIBD can be represented as a CSP with a v by b matrix model. Each variable in the matrix is a binary variable m_{ij} with domain $\{0, 1\}$. There are three sets of constraints:

1. $\sum_{j \in I^b} m_{ij} = r$, for all $i \in I^v$
2. $\sum_{i \in I^v} m_{ij} = k$, for all $j \in I^b$
3. $\sum_{j \in I^b} m_{ij}m_{i'j} = \lambda$, for all $i \in I^v, i' \in I^v, i < i'$

This problem is called the *master problem* in what follows. Any permutation of the rows or of the columns is symmetry of the master problem.

For every node ν explored during the search for this master problem, a graph \mathcal{G}_ν is constructed as follows. There is a node per line i , and a node per column j . There is an edge between a line i and a column j if, and only if, $m_{ij} = 1$ in that state. For every node s in the search tree for the master problem, and for each stored no-good ν for the master problem, the SBDD method checks if ν dominates s . If ν dominates s , then the node s is pruned and the search backtracks. We say that ν dominates s if and only if there is an SGI between the graph \mathcal{G}_ν and the graph \mathcal{G}_s . The symmetry breaking techniques discussed in this paper can be applied to these SGI sub problems.

We ran experiments with various instances of the BIBD problem. For every instance we report the time needed for three variants of SBDD:

- **A** : the running time for the SBDD method of [9]
- **B** : the running time for the method of **A** where constraints (5) are added to the SGI sub problems
- **C** : the running time of the method of **B** where the GE-tree method for breaking value symmetries is used for the SGI sub problems.

All 3 variants explore the same search tree. They also compute the same number of solutions. Those are also reported.

We also report the ratio between the time for **A**, and **B** and **C**. The geometrical means of the ratios are given. This shows that **B** is about 1.8 times faster than **A**, and that **C** is about 2.7 times faster than **A** on average. **C** can be up to 50 times faster than the basic SBDD method. Those results are quite impressive, given that we need to compute graph automorphism groups before solving each SGI sub problem. This can probably be made even better by using an implementation of the full GE-tree method instead of simply using (7).

BIBD	SOLS	A	B	C	A/B	A/C
6 3 6	6	0.3	0.15	0.1	2.0	3.0
7 3 3	10	0.41	0.31	0.21	1.3	2.0
6 3 8	13	0.8	0.45	0.36	1.8	2.2
15 5 2	0	1.1	0.78	0.77	1.4	1.4
16 6 2	3	1.2	0.82	0.39	1.5	3.1
13 3 1	2	0.55	0.6	0.49	0.9	1.1
9 4 3	11	1.7	1.4	1	1.2	1.7
22 7 2	0	2.3	1.7	1.7	1.4	1.4
6 3 10	19	2.1	1	0.8	2.1	2.6
16 4 1	1	2.1	1.2	0.29	1.8	7.2
7 3 4	35	2.8	1.3	0.97	2.2	2.9
15 7 3	5	3.3	2	0.92	1.7	3.6
9 3 2	36	6.6	3.6	2.7	1.8	2.4
10 5 4	21	7.7	6.5	4.8	1.2	1.6
7 3 5	109	17	7.5	6.4	2.3	2.7
25 5 1	1	48	5.3	0.95	9.1	50.5
7 3 6	418	114	46	43	2.5	2.7
19 9 4	6	139	112	80	1.2	1.7
Mean					1.8	2.7

Table 2. Time for computing all solutions for BIBD.

7 Discussion

We have established two major results. First of all, all variable symmetries can be broken by a linear number of binary constraints if there is an all different constraints on all the variables of the CSP. Second, symmetry breaking constraints of [1] can be safely used in conjunction with the GE-tree method of [11]. Both results have been applied to the SGI problem, which is a prevalent problem in computer science applications. In particular, the SBDD method of [9] extensively uses solutions to SGI sub problems in order to prune search. Experiments on BIBD problems show that the symmetry breaking techniques discussed in this paper are highly effective.

The results described in this paper can be generalized. First of all, theorem 4 is valid for all CSPs where the variables are subject to an all different constraint. It would be interesting to see if similar results can be obtained for other forms of CSPs. Second, we assumed that all graphs were undirected. The same approach for breaking symmetries could be applied to the SGI problem for directed graphs. Third, the same symmetry breaking method can be used for the graph isomorphism problem. Indeed, this amounts to assume that the graphs \mathcal{G}_1 and \mathcal{G}_2 have the same number of vertices and the same number of edges.

It is worth mentioning that we presented a method for breaking all variable symmetries, and all value symmetries. However, our method does not break products of both kinds of symmetries. It remains to be seen if a simple combination of variable and value symmetry breaking techniques can break all such symmetries. It is also worth mentioning that we consider symmetries that are either variable symmetries or value symmetries. Our work does not address cases where symmetries can permute values and variables.

Acknowledgements. Pascal Massimino pointed out that the SGI sub problems in the SBDD method could contain many symmetries. Marie Puget greatly im-

proved the readability of this paper. I warmly thank them both.

References

- [1] Crawford, J., Ginsberg, M., Luks E.M., Roy, A. "Symmetry Breaking Predicates for Search Problems." In proceedings of KR'96, 148-159.
- [2] P. Flener, A. M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, T. Walsh.: "Breaking Row and Column Symmetries in Matrix Models. " Proceedings of CP'02, pages 462-476, 2002
- [3] Gent, I.P., Walsh, T., Sellmann, B.: CSplib <http://www-users.cs.york.ac.uk/~tw/csplib/>
- [4] ILOG: ILOG Solver 6.0. User Manual. ILOG, S.A., Gentilly, France, Septembre 2003
- [5] Lustig, I.J., and Puget, J.F. (2001). "Program Does Not Equal Program: Constraint Programming and its Relationship to Mathematical Programming," Interfaces 31(6), 29-53.
- [6] Mc Kay, B.: "Practical Graph Isomorphism" Congr. Numer. 30, 45-87, 1981
- [7] Petrie, K., Smith, B.M. : "Symmetry breaking in graceful graphs." In proceedings of CP'03, LNCS 2833, 930-934, Springer Verlag, 2003.
- [8] Puget, J.-F.: "On the Satisfiability of Symmetrical Constraint Satisfaction Problems." Proceedings of ISMIS'93 (1993), 350-361.
- [9] Puget, J.-F.: "Symmetry Breaking Revisited." Proceedings of CP'02, 446-461.
- [10] Regin, J.-C.: "Developpement d'Outils Algorithmiques pour l'Intelligence Artificielle. Application la Chimie Organique." PhD thesis, Universite de Montpellier II, 1995 (in French)
- [11] Roney-Dougal C.M., Gent, I.P., Kelsey T., Linton S.: "Tractable symmetry breaking using restricted search trees" To appear in proceedings of ECAI'04.
- [12] Roy, A., Luks, E.: "The complexity of symmetry-breaking formulas", *Annals of Mathematics and Artificial Intelligence*, 41 (2004), 19-45 (with A. Roy).
- [13] Seress, A.: *Permutation Group Algorithms* Cambridge University Press, 2003.
- [14] Sorlin, S., Solnon, C.: "A global constraint for graph isomorphism problems" In proceedings of CPAIOR'04, LNCS, Springer.

Automatically Exploiting Symmetries in Constraint Programming

Arathi Ramani and Igor L. Markov

Department of EECS, University of Michigan
1301 Beal Avenue, Ann Arbor, MI 48109, USA
{ramania, imarkov}@eecs.umich.edu

Abstract

We introduce a framework for studying and solving a class of CSP formulations. The framework allows constraints to be expressed as linear and non-linear equations, then compiles them into SAT instances via Boolean logic circuits. While in general reduction to SAT may lead to the loss of structure, we specifically detect several types of structure in high-level input and use them in compilation. Linearity is preserved by the use of pseudo-Boolean (PB) constraints in conjunction with a 0-1 ILP solver that extends common SAT-solving techniques. Symmetries are detected in high-level constraints by solving the graph automorphism problem on parse trees. Symmetry-breaking predicates are added during compilation. Our system generalizes earlier work [10; 2; 29] on symmetries in SAT and 0-1 ILP problems. Empirical evaluation is performed on instances of the social golfers and Hamming code generation problems. We show substantial speedups with symmetry-breaking, especially on unsatisfiable instances. In general, our runtimes with the specialized 0-1 ILP solver Pueblo [26] are competitive with results reported for ILOG Solver [28] in [15].

1 Introduction

Traditional constraint programming (CP) techniques such as generalized arc consistency (GAC) are frequently the methods of choice for hard problems arising in real-world applications. Well-known packages such as ECLⁱPS^e [23] and ILOG Solver [28] offer powerful environments for constraint specification and solver deployment. These systems provide for the development of problem-specific solvers using the best available techniques for a given problem. Another option is *reduction* - a problem for which no solver is available can be reduced to another problem for which a solver does exist.

Boolean satisfiability (SAT) is commonly used in problem reductions, since it is widely-known and many SAT solvers are available in the public domain. Unfortunately, in most cases reduction-based methods are not competitive with CP approaches developed for a problem. While CP-based techniques can take advantage of problem-specific bounds to re-

tain tighter control of the search, SAT solvers cannot. This disadvantage is mitigated to some extent by recent breakthroughs in SAT-solving. With new exact SAT solvers such as ZChaff [20], the size and scope of application-derived instances that can be solved has widened [21]. Unfortunately, many applications do not benefit from breakthroughs in SAT solving due to inefficiencies introduced during encoding. The CNF format used for SAT instances is very restrictive, and even encoding constraints with simple linear operations can result in a blowup in size. Another cause of inefficiency is the *loss of structure during problem reductions*. Examples of structure in constraints include *linearity* and *symmetry*.

The presence of symmetries slows down search due to the existence of redundant search paths. The work in [10] describes how symmetries can be detected in a SAT instance by reduction to graph automorphism and broken by adding lexicographic ordering constraints, called MinLex symmetry-breaking predicates (SBPs). The addition of MinLex SBPs accelerates SAT solvers. Linear “counting” constraints popular in applications are studied in [2]. These constraints can be efficiently expressed using ILP, where linear equations are allowed, but expressing them in CNF may be expensive. On the other hand, generic ILP solvers such as CPLEX are sometimes not competitive with leading-edge SAT solvers on Boolean constraints. Linearity can be preserved using 0-1 ILP, a problem closely related to SAT but with an ILP-like input format. Specialized techniques developed for SAT can be adapted to 0-1 ILP without paying any penalty for generality. Recently, several specialized 0-1 ILP solvers such as PBS [2], Galena [8] and Pueblo [26] have been introduced. Symmetry-breaking techniques from [10; 1] were extended to 0-1 ILP in [4]. In [15], the author proposes symmetry-breaking ordering constraints for CSPs with *matrix models*, but it is not clear how these constraints extend to SAT/0-1 ILP reductions.

This work contributes a framework for structure-aware compilation of a class of constraint programming problems by reduction to SAT and 0-1 ILP. We generalize techniques proposed in [10; 4] to detect symmetries in constraints via reduction to *graph automorphism*. Unlike earlier work, we detect symmetries in *high-level input* and add symmetry-breaking predicates to the original specification. Our system facilitates comparison of different encoding strategies and SAT reductions. This is useful since recent work [29;

5; 6] has shown that the encoding used can dramatically affect search speed. Our goals here are (1) to generalize earlier work on the detection of symmetries and linearity in SAT instances so that it is applicable to a more general class of high-level CSPs (2) to automate the tasks of reduction to SAT/0-1 ILP and structure detection (3) to use this framework to study whether using structure can improve the performance of reduction-based methods. Earlier work [10] detects and breaks symmetries after problems have been reduced to CNF. Our work generalizes these techniques by detecting structure *before reduction* and using it to produce more effective encodings. Our empirical results for the social golfer and Hamming code generation problems show that breaking symmetries during reduction vastly improves the performance of both SAT and 0-1 ILP solvers. On many instances, our runtimes are competitive with results reported using ILOG Solver [28] in [15]. Symmetries detected by our method can be used by *any* constraints solver, not just one that assumes reduction to SAT, since we detect symmetries in high-level input. While we add SBPs during preprocessing, there are several methods that focus on breaking declared symmetries during search [25; 13] that can make use of the symmetries we detect.

The rest of the paper is organized as follows. Section 2 discusses background and previous work. Section 3 explains how symmetries are detected and broken in high-level constraints. Section 4 discusses more comprehensive symmetry-breaking, with empirical results in Section 5. Section 6 concludes the paper. The details of compilation to SAT and 0-1 ILP and the encodings we use are discussed in the Appendix.

2 Background and Previous Work

Boolean Satisfiability (SAT). A SAT instance consists of a set of 0-1 variables V , and a set of clauses C , where each clause is a *disjunction of literals*. A literal is a variable or its complement. The SAT problem asks to find an assignment to the variables in V that satisfies all clauses in C , or prove that no such assignment exists.

0-1 ILP. 0-1 ILP allows a CNF formula to be augmented with Pseudo-Boolean (PB) constraints, or linear inequalities with integer coefficients of the form: $(a_1x_1 + a_2x_2 + \dots + a_nx_n \leq b)$ where $a_i, b \in \mathbb{Z}$, $a_i, b \neq 0$, and x_i are literals of Boolean variables.

CNF vs. 0-1 ILP. Recent work has shown that formulating problem instances as 0-1 ILP instead of SAT can result in accelerated search. Specialized 0-1 ILP solvers have been developed in [2; 8; 26], and have been shown to perform better than both leading-edge SAT solvers [20] and generic ILP solvers such as CPLEX on some 0-1 ILP formulas. However, this is not always the case. For an application, there can be several reductions to SAT, and some encodings are more difficult to solve than others. CNF encodings for circuit layout applications in [2] contain large numbers of symmetries, increasing their difficulty. In [29], Warners proposes an efficient encoding where a PB constraint is replaced by a linear number of CNF clauses. In [5], a tree-based linear conversion is proposed to translate 0-1 ILP constraints to CNF. More recently, [6] discusses a GAC-preserving encod-

ing, with a solver modification that results in SAT instances that are solved faster than their 0-1 ILP counterparts. Our approach constructs a parse tree and instantiates Boolean circuits for addition, multiplication and subtraction. Most previous work performs reduction to SAT on a per-problem basis, but we provide a high-level specification language in which constraints can be easily expressed and conversion to SAT/0-1 ILP is automated for all problems. Given the impact that efficient encodings have on search speed, our framework is designed so that different encodings can be easily plugged in and used with our symmetry-breaking infrastructure. Our work is relevant to the recent \mathcal{NP} -SPEC project [7], which aims to provide a formal specification language for all problems in \mathcal{NP} , and translate them into SAT. However, [7] does not address symmetry and linearity during compilation.

Symmetry detection and breaking. A *symmetry* of a discrete object is a reversible transformation of its components that leaves the object unchanged, for example, permutations of graph vertices that map edges into edges. Symmetries occurring in a SAT instance indicate the presence of redundant search paths, and breaking symmetries can prune the search tree and reduce search time. Detection of symmetries in CNF formulas by reduction to graph automorphism is proposed in [10]. A graph is built from a CNF formula such that there is a one-one correspondence between symmetries of the formula and the graph. The graph automorphism software Nauty [17] is used to detect symmetries in the graph. The symmetry group induces an equivalence relation on the set of variable assignments for a CNF formula. *Lex-leader* symmetry-breaking predicates (MinLex SBPs) that allow only the *lexicographically smallest* assignment in an equivalence class are defined in [10]. A more efficient SBP construction is proposed in [3]. Symmetry detection via graph automorphism is extended to 0-1 ILP in [4]. Our work generalizes these methods to a broader class of problems that use integer coefficients, non-binary variables and non-linear operations. Instance symmetries are detected at a higher level, eliminating the risk that some symmetries may be obscured during reduction. In [15], the author defines high-level lexicographic (MinLex), anti-lexicographic (anti-Lex) and multiset ordering constraints for CSPs with matrix models that exhibit symmetry. However, row and column symmetries must first be identified in matrix models for individual problems and constraints designed accordingly. Our system allows symmetries to be automatically detected in any problem instance, not just a matrix model, and used by any solver. This functionality may be useful to methods that focus on declared symmetries during search. A modified search procedure that performs partial symmetry-breaking for matrix models is proposed in [25], where SBPs are specified for a stabilizer set that is a subgroup of the symmetry group. We find generators of the symmetry group using the graph automorphism program Saucy [11], and these generators can be used by the algorithms in [25] to compute SBPs. Another related work is [13], which takes as input some generators of the symmetry group and uses them to check for dominating elements in the search tree. Since our system automatically detects generators it may be applicable to such algorithms. At present, we use only MinLex SBPs from [10]. We have not yet studied other types of

SBPs such as those in [15]. Symmetries in linear programming problems have also been discussed in [18].

3 Symmetry Detection

Unlike earlier work [10; 4], which detects symmetries in SAT/0-1 ILP instances *after* reduction, we detect symmetries in high-level input using the parse tree created from the constraint specification. Symmetries in high-level input correspond directly to symmetries of the instance and can be used to prune the search tree by multiple solvers. Symmetries detected in a SAT instance can only be used by SAT solvers, or must be traced back to the original instance to understand their significance. This reconstruction may be difficult. Also, some symmetries may be obscured during reduction. For example, counting constraints are symmetric, but the most compact encodings for these constraints [29] use comparator circuits which are not symmetric. The methods we describe here efficiently detect symmetries in high-level constraints and add SBPs to eliminate redundant search paths.

Detecting symmetries in CNF and 0-1 ILP via graph automorphism was first proposed in [10]. We follow a similar approach for high-level symmetry detection. A parse graph is built from the constraints such that there is a one-to-one correspondence between the symmetries of the constraints and the graph symmetries. We describe the graph construction only for the arithmetic operators '+', '−', and '*', but it can be extended to include more arithmetic or logical operators by adding more colors. An example formula in our specification language and the corresponding graph construction are shown in Figure 1. The formula declares two 3-bit integers x_1 and x_2 , and the constraint $x_1^2 + x_2^2 == 25$. The specification language we use is described in the Appendix. Vertex shapes in the figure indicate different colors. The figure shows the symmetry between vertices for x_1 and x_2 .

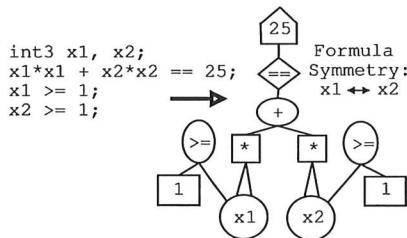


Figure 1: **Constraints declaration in our specification language and the corresponding parse graph construction. Vertices are shaped differently to indicate different colors.**

The graph construction is outlined as follows.

Step 1. Each binary variable x_i in a formula is represented by two positive and negative literal vertices, v_i and v_i' , which are given the same color. The vertices v_i and v_i' are connected by an edge to ensure Boolean consistency. Each multi-bit variable x_j is represented by a single variable vertex v_j . A unique color is associated with each bit size.

Step 2. For each constraint C_i , two vertices T_i and R_i represent the constraint type ($\leq, \geq, ==, !=$) and RHS value re-

spectively. A unique color is associated with each constraint type and RHS value. The vertices T_i and R_i for a constraint C_i are connected by an edge. Additionally, for each C_i :

Step 2a. Variables/literals are grouped by the priority of operations in which they occur. Multiplication between variables or by coefficients has the highest priority. '+', '−' and '*' operators have distinct colors. Each distinct coefficient value in the formula is also given a unique color. Variables connected by a '*' operator are grouped under a single *coefficient vertex* that represents the product of their coefficients (if the product is unity, this vertex is omitted). This coefficient vertex is in turn attached to a *multiplication vertex*. Variables/literals not involved in multiplication operations are grouped by coefficient, with all variables having the same coefficient value connected to a common coefficient vertex.

Step 2b. After grouping multiplicative terms, we have single variables/groups or multiplicative groups connected by '+' or '−' operations. Variables/groups associated with a '+' sign are connected directly to the constraint type vertex T_i ('+' is the default operation, so there are no special vertices for it). Variables/groups associated with a '−' operation are connected to a *negation vertex* to indicate subtraction. The negation vertex is connected to the type vertex T_i .

Theorem 3.1. *Assume that a colored parse graph is constructed from a given formula of constraints as outlined above. Then, the symmetries of the constraints correspond one-to-one to the symmetries of the graph.*

Proof. We first prove that *a symmetry in the constraints is a symmetry in the parse graph*. Consider a formula with a set V of formula variables and a set C of constraints. Consider two variables, $v_1, v_2 \in V$, and let $C_1, C_2 \subset C$ be the sets of constraints that v_1 and v_2 occur in respectively. Let v_1 and v_2 be symmetric. Then, for every constraint c in C_1 there is a corresponding constraint in C_2 that is its symmetric image.

We construct a colored parse graph $G(X, E)$ for the formula where X is the set of vertices in the graph and E the set of edges. Let x_1 and x_2 be the vertices created for v_1 and v_2 respectively, and E_1 and E_2 be the edges incident on x_1 and x_2 . Assume that x_1 and x_2 are *not* symmetric in the graph construction. For this to be true, it must be true that the edge sets E_1 and E_2 are not symmetric. Without loss of generality, assume there exists some edge $e \in E_1$ that does not have an image in E_2 . From the graph construction rules, an edge can connect a variable vertex to one of the following: (i) a complementary literal (ii) a constraint type vertex (for addition with unit coefficient) (iii) a negation vertex (for subtraction with unit coefficient) (iv) a multiplication vertex (for multiplication with unit coefficients) and (v) a coefficient vertex that is connected to a multiplication/negation/constraint type vertex. In the first case, assume that e connects x_1 to a complementary literal vertex, and x_2 does not possess such an edge. Then, v_2 is not a binary variable, and it cannot be symmetric to v_1 . In the second case, e indicates the presence of a constraint $c \in C_1$ where v_1 is added with a coefficient of 1. Since v_1 and v_2 are symmetric in the formula, there *must* be a constraint in C_2 that matches c . However, if such a constraint existed, there would be an edge representing it in E_2 , symmetric to e . The same argument applies to cases (iii) and (iv). The only special case occurs in (v), when variables

are multiplied together with different coefficients. We use the product of all coefficient values as the resulting coefficient. This reflects the fact that multiplication is commutative, i.e. $(av_1)(bv_2) = (ab)(v_1)(v_2)$ and $(cv_3)(dv_2) = (cd)(v_1)(v_2)$, so if $ab = cd$ then the expressions are symmetric.

For the other direction, we note that symmetries in the parse graph can only exist between vertices of the *same color*. Additional vertices are created to represent operations, but they can never be mapped to variable vertices. Thus, the only spurious symmetries we need to consider are between *variable vertices of the same bit size*. It is clear that the proof for the forward direction can be reversed for this case, i.e. edge sets incident on both vertices must be symmetric and represent symmetric constraints in the formula.

□

Avoiding abstraction overhead. Our graph construction generalizes earlier work in [10; 4] for CNF and 0-1 ILP formulas. Often, generalization involves paying a performance penalty - in this case, dealing with a more expressive input format that includes non-linear constraints can introduce additional vertices. This penalty can be avoided by modifying the graph when special cases are detected. Consider the case where an instance contains *only* 0-1 ILP constraints with no non-linear operations and only 1-bit variables. IN this case, our construction is designed to mimic the construction in [4], and produce *exactly* the same graphs. For pure CNF formulas, some modification is required to produce graphs as compact as the specialized constructions from [10; 1]. Since there are no coefficients or RHS values, constructions in [10] and [1] use only two types (colors) of vertices: literal and clausal. A clause with > 2 literals is represented by a clausal vertex, connected to its literal vertices. Binary clauses are represented by an edge between both literals. Graphs created by our system require constraint type and RHS value vertices for each constraint. However, CNF formulas are easy to detect. A CNF formula involves *only* binary variables. All coefficients are unity. Clauses can be expressed in two ways: as the logical-or ("||") of literals, or as the additive constraint that the sum of literals must be ≥ 1 . These characteristics can be tested for, and graph construction altered accordingly.

Symmetry-Breaking Predicates (SBPs). The parse graph is analyzed for symmetries using the efficient automorphism program Saucy [11], which returns generators of the symmetry group. We generate high-level lex-leader SBPs from the generators, and add them as constraints to the original instance. These SBPs are also compiled into SAT. For multi-bit variables, SBPs may be large and complex if a generator has several cycles (for a detailed description of cycles in a generator, and the resulting predicates, see [10]). We break only the first few (1 or 2) cycles in multiple-cycle generators for simplicity. For binary variables, we implement the efficient linear-sized SBP construction in [3] and add these SBPs to the CNF formula. The problems we test here all use matrix models with binary variables. The design of efficient SBPs for multi-bit variables is a direction for future research.

4 Comprehensive Symmetry Breaking

In this section, we discuss simple extensions to increase the system's coverage of symmetries.

Symmetries in Associative Expressions. Many of the operators that we support, such as '+' and '*' are associative, i.e. $x_1 + x_2 + x_3 = x_2 + x_3 + x_1$ and $(x_1 + x_2) + x_3 = x_1 + (x_2 + x_3)$. However, parse trees built from constraints often do not reflect this symmetry. In parsing, language rules are recursively matched. This imposes a non-symmetric structure on the parse tree. We avoid this non-symmetric structure by grouping all variables connected by an associative operation together. For example, given the expression $x_1 + x_2 + x_3$ the parser first matches $x_1 + x_2$ as a single group, and then matches x_1 and x_2 individually, which is not symmetric. Our construction treats all '+'s as a single '+' operation connecting a number of expressions, which may be either identifiers or multiplicative terms. Symmetry in associative operations can also be missed when nested parentheses are used. Our system currently does not support the nesting of expressions through the '(' and ')' operators, but can be easily extended to do so. Here we discuss symmetry-detection for this case. Detecting symmetries in associative operations has been addressed in the CGRASS system [12]. However, CGRASS detects symmetries in an ad-hoc way, by keeping track of the number and type of constraints a variable occurs in and matching these for different variables. Detection via graph automorphism is more comprehensive, and given efficient software such as Saucy, incurs hardly any overhead. Our method, like CGRASS, is not complete - it uses only the generators of the symmetry group found by Saucy. For complete symmetry-breaking, the full group would have to be reconstructed from the generators. This has been found to be very time-consuming [10], whereas using only generators is more efficient and often just as effective. CGRASS also undertakes simplification of constraints in other ways, which our system does not cover.

Consider the expressions $x_1 + (x_2 + x_3) + x_4$ and $x_1 + (x_2 + (x_3 + x_4))$, which are the same, but are evaluated differently due to parenthesization. The order of evaluation imposed by parentheses hides the symmetry between variables, since expressions enclosed within '(' symbols are treated as separate sub-expressions. Evaluating parenthesized expressions separately does not account for symmetries due to associativity of operations. However, it is possible to simplify high-level input so that such symmetry is preserved. We list simplification rules for the operators '+', '-' and '*'.

Rule 1. Nested () symbols must be simplified before the outermost () operation can be simplified.

Rule 2. If an expression within () symbols is flanked by '+' and '-' operations on the left and right sides, parentheses are unnecessary, e.g., in $\dots + (x_1 + x_2) + \dots$ the () operators can be ignored.

Rule 3. If an expression within () symbols is multiplied by a single term, the resulting expression can be evaluated, e.g., $x_2 * (x_1 + x_4)$ is written as $x_2 * x_1 + x_2 * x_4$. It is possible to simplify the parenthesized products, e.g. $(x_1 + x_2) * (x_3 + x_4)$ by implementing multiplication rules, but this may cause a size blowup in graphs for large expressions.

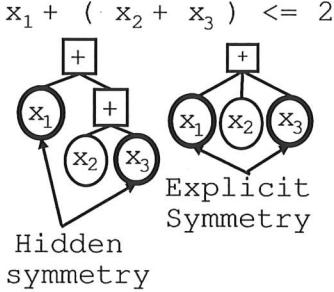


Figure 2: **Associative symmetry with parenthesized sub-expressions:** x_1 and x_3 are symmetric but the original parse tree is asymmetric, since the sub-expression is represented with a separate node.

The above list of rules can be extended further, but it already facilitates the detection of symmetries in simple associative expressions. This is illustrated in Figure 2, where x_1 and x_3 are symmetric, but the symmetry is not visible in the parse graph. With the proposed modifications the associative symmetry is preserved. Our system already implements this feature for ‘+’ and ‘-’ operations without parentheses, where we ignore the order in which the operations occur.

Value Symmetry. Our work so far detects *formula symmetries*, that are determined by the occurrence of variables in constraints. However, *value symmetries* that occur between the actual domains of variables can also be significant. Ordering constraints for declared value symmetries are discussed in [15], and [16] describes an algorithm to detect and break value symmetries during search. We discuss how our system may be extended to detect value symmetry.

Value symmetry can arise from *operators* that control the value of a variable, e.g. the *complement* operation on binary variables, i.e. $a' = 1 - a$. The mapping $a \leftrightarrow a'$ is known as a *phase shift symmetry*. The construction from [10] does not always account for phase-shift symmetries, but [1] proposes an improvement that detects phase-shift symmetries in almost all cases. For the non-binary case, such symmetries may arise in problems with a *cyclic* nature, for example, scheduling problems. Any scheduling solution for {Monday, Tuesday, Wednesday} can often be shifted to {Tuesday, Wednesday, Thursday}. Such shifts can also be described by an operator - if a variable’s domain is a cyclic group modulo 4, we can say $a'' = (a + 1)\%4$. Intuitively, the graph construction to represent a cyclic group of values is a cycle of vertices. However, if the domain size is > 2 , this will result in spurious symmetries if all vertices are given the same color, since a'' can map to $(b'')''$, and so on. Each vertex in the cycle must be given a different color for this construction to work. This allows cyclic symmetries to be detected. The SBPs we use may need to be modified for this case, and our ongoing work is focused on proving correctness in SBP construction.

Giving each value in a variable’s domain a different color prevents the detection of value symmetries between values in the domain of the same variable. A set of constraints satisfied when $a = 0$ may also be satisfied when $a = 2$. This type of symmetry-detection is addressed in [16]. Adapting our tech-

niques to detect such symmetries is more difficult, since it may require the enumeration of variable and constraint values in the graph, resulting in very large and complex graphs. Another focus of our current work is developing efficient graph constructions for this case.

5 Empirical Results

We test our system on constraint programming problems with *matrix models* with row and/or column symmetries from [15]. Each problem is modeled using the constraints described in [15] and specified in our system’s input language, followed by symmetry detection and compilation to SAT and 0-1 ILP. SBPs are added to the CNF or ILP instances. We use Saucy [11] to detect symmetries, ZChaff to solve SAT instances, and the new 0-1 ILP solver Pueblo [26] to solve 0-1 ILP instances. We show results for the balanced incomplete block design problem (BIBD), social golfer problem (SG) and Hamming code generation (HC) problems. Results here are obtained using a Intel Pentium processor processor at 1GHz for the SG and HC problems, and an Intel Xeon dual processor at 2 GHz. Both systems have 1GB of RAM and run RedHat Linux 9.0. ZChaff and Pueblo runtimes are the average of 3 starts. Timeout is set at 600 seconds. For BIBD instances, we use the Xeon processor at 2GHz to compare our encodings with those in [24]. For SG and HC instances, we use the 1GHz Pentium processor to allow runtime comparisons with [15]. Symmetry-breaking ordering constraints in [15] are implemented using ILOG Solver and tested on a 1 GHz Pentium processor running Windows XP. We note that [15] also reports a “number of failures” metric, which is the number of incorrect decisions made by Solver at nodes in the search tree. We do not have access to Solver and the SAT/0-1 ILP solvers we use do not report such a statistic. The SBPs we use are added as part of the instance and a SAT/0-1 ILP solver cannot distinguish between SBPs and regular constraints. Therefore, we cannot report a similar metric for our techniques, and runtime is the only comparable statistic¹. However, we use exactly the same hardware as [15] so that runtime comparisons are fair. Since it is not possible for us to use Solver, we use results directly from [15].

Balanced Incomplete Block Design Problem (BIBD). This problem asks to find $b > 0$ subsets of a set V of $v \geq 2$ elements such that each subset contains exactly k elements ($v > k > 0$), each element appears in exactly $r > 0$ subsets, and each pair of elements appears together in exactly $\lambda > 0$ subsets. An instance is expressed as the 5-tuple (v, b, r, k, λ) , and named `bibd(v, b, r, k, λ)` in the results table. We use the matrix model described in [15] (originally from [19]). We initially tested encodings with and without SBPs using ZChaff and Pueblo on the large instances used in [15] (originally from [9]). However, our observation on these instances was that adding MinLex SBPs actually affects performance negatively for the Pueblo solver (ZChaff is unable to solve most instances within the time limit, with or without SBPs). For satisfiable instances, this is not unusual and has been noted earlier in [10]. When there are several solutions, adding

¹Due to lack of space, we cannot report both number of failures and runtime from [15]

SBPs may prevent some solutions from being found earlier in the search. This is borne out by our results on other problems. However, this does not explain the poor performance on unsatisfiable instances of this problem, which may be because MinLex SBPs are not useful in this case. In [15], several types of SBPs are tested, and the most effective SBPs for the BIBD problem anti-Lex ordering constraints. Since anti-Lex orderings are the reverse of MinLex orderings, they permit different assignments than MinLex, and may be more helpful in finding solutions for BIBD. However, we use this problem to illustrate the importance of efficient encodings. SAT encodings for the BIBD problem have been developed in [24], where the instances used are difficult for many SAT solvers, but are solved by CP solvers in a few minutes. These encodings are available at [14], with and without symmetry-breaking clauses from [24]. Table 1 shows a comparison of both encodings. The first column gives the instance parameters, followed by Saucy statistics for high-level symmetry-detection. This is followed by ZChaff and Pueblo runtimes for our encoding, and ZChaff runtimes for encodings from [24] with and without SBPs. Pueblo does not accept instances without 0-1 ILP constraints. Both Pueblo and ZChaff solve all instances with our encoding in a few seconds, but ZChaff times out on several instances from [24]. All instances possess symmetries, but Saucy runtimes are negligible.

Social Golfers (SG). This problem seeks to divide $g \times s$ golfers into g groups of size s for each of w weeks. Each golfer must play once a week. Any two golfers play in the same group at most once. A problem instance is described by its parameters (g, s, w) and is named $sg(g, s, w)$ in the results table. We use the modified 3-D matrix model from [15], and the same instances used in [15]. Instances are tested on ZChaff and Pueblo with and without SBPs.

Results are shown in Table 2. The first column gives instance parameters (sg for SG instances), followed by the number of symmetry generators and runtime for Saucy. Next, we show approximate instance sizes and runtimes with and without SBPs. For SAT conversions, we show the number of variables and clauses. For 0-1 ILP instances we also show the number of PB constraints, which is the same as the number of high-level constraints in the instance specification. The best runtimes for a given instance are boldfaced. For this problem, adding SBPs speeds up Pueblo considerably on *unsatisfiable* benchmarks. For *all* cases where Pueblo is slower with SBPs, the instance is satisfiable. ZChaff is faster with SBPs for both SAT and UNSAT cases, but is not competitive with Pueblo. All instances possess large numbers of symmetries. The last column shows results reported in [15]². Pueblo is usually competitive with Solver results from [15] on SAT instances without the addition of SBPs. However, on UNSAT instances, SBPs are needed to make it competitive, and are effective in doing so. For the larger instances, Saucy runtimes are significant. This increases the overall time for our flow. However, [15] requires SBPs to be designed and implemented separately for individual problems. Our system is automated and generalized. Moreover, [15] reports results for four mod-

els of SBPs. Two of these are basic SBPs that assign values to a subset of the variables in an instance, thus forcing assignments that satisfy constraints on the remaining variables. The other two models use MinLex and anti-Lex constraints. Here, we report the best results among all models. Given an instance it may not be clear which model to use for best results until several have been tried. There is no model in [15] which consistently performs well for this problem. Our system uses only MinLex SBPs.

Hamming Code Generation (HC). This problem seeks to find b -bit code words to code n symbols, where the Hamming distance between two symbols is at least d . An instance is specified by the parameters (n, b, d) . We use the matrix model from [15], and report results with and without symmetry-breaking in the last four rows of Table 2. The instances $hc(10, 15, 9)$ and $hc(12, 20, 12)$ are unsatisfiable, and the other two are satisfiable. [15] Results for the first two instances are available in [15], the last two are listed as N/A. We observe that symmetry-breaking is useful for both SAT and UNSAT instances, with greater benefit for UNSAT instances. Adding SBPs speeds up ZChaff in all cases, but it is not competitive with Pueblo and Solver. Results reported from [15] are the best out of several combinations of lexicographic and multiset-ordering SBPs. However, several of these combinations are not competitive with our results using Pueblo with SBPs.

Overall, the detection of structure - both linearity through 0-1 ILP and symmetries by the addition of SBPs - improves performance considerably for both Pueblo and ZChaff. For most unsatisfiable instances, the best results are obtained using Pueblo with SBPs added. For satisfiable instances, Pueblo is not improved by SBPs, and in some cases is actually slower. However, ZChaff benefits from SBPs for both SAT and UNSAT instances. This may be because SBPs have greater impact on variable orderings for Pueblo. In most cases Pueblo's results are competitive with results reported for Solver in [15] over a variety of symmetry-breaking ordering constraints. For the cases where Pueblo is faster with SBPs, the average speedup over its performance without SBPs is 83.2, not including timeouts for the no-SBP version. On satisfiable instances, the average slowdown with SBPs is 5.6, but it is much less than that in most cases and there are no timeouts with SBPs. Our system uses academic solvers whose source code and/or binaries are publicly available, but runtimes are comparable with those of Solver, a highly optimized commercial tool.

All results here use problems with matrix models, which frequently possess large numbers of symmetries by construction. While row and column symmetries can be detected manually in a matrix model, our system provides a way to detect and break these symmetries automatically without having to give it any knowledge of the problem semantics. Moreover, it is not restricted to matrix models, and may be used for problems that are likely to have symmetry, but for which matrix models do not exist. It is also applicable in cases where added constraints may disrupt the symmetry in matrix models, e.g. for instances with "customized" requirements. For example, in the social golfer problem, we can add the constraint that certain pairs of golfers must *never* be in the same

²Results in [15] are on a logarithmic scale, so our numbers are not exact, but all runtimes are rounded *down* for fairness.

Instance Name	Symmetry Stats			Our Encoding				Encoding in [24]	
	Symm.	Gen.	Saucy Time	W. SBPs		W/o. SBPs		W. SBPs	W/o. SBPs
				ZChaff	Pueblo	ZChaff	Pueblo	ZChaff	ZChaff
bibd(7,7,3,3,1)	2.54e7	12	0	0.08	0	0.01	0	0.29	T/O
bibd(6,10,5,3,2)	2.61e9	14	0	0.54	0	0.03	0	54.24	T/O
bibd(7,14,6,3,2)	4.39e14	19	0.01	0.38	0.01	1.25	0.01	T/O	T/O
bibd(9,12,4,3,1)	1.73e14	19	0.02	0.64	0.01	1.89	0.013	T/O	T/O
bibd(8,14,7,4,3)	3.51e15	20	0.02	0.72	0.01	1.57	0	T/O	T/O

Table 1: ZChaff results and Saucy statistics for BIBD instances using our encodings and those in [24], with and without SBPs. T/O indicates timeout at 600s. Pueblo is not tested on encodings in [24], since they are not available as 0-1 ILP.

Instance Params	Saucy Stats		Size with SBPs						Size w/o SBPs						Sol ver	
	G.	Tm.	CNF - ZChaff			0-1 ILP - Pueblo			CNF - ZChaff			0-1 ILP - Pueblo				
			Var.	Cl.	Tm.	Var.	Cl.	PB	Tm.	Var.	Cl.	Tm.	Var.	Cl.	PB	Tm.
sg(2,5,4)	16	0.02	6311	33K	0.06	1694	1361	141	.003	6139	32K	0.12	1522	721	141	0.01 .01
sg(2,6,4)	18	0.02	9076	48K	0.14	2418	1835	178	.006	8868	46K	0.15	2210	1057	178	0.01 0.1
sg(2,7,4)	20	0.03	12K	65K	0.31	3270	2373	219	0.01	12041	63894	0.14	3026	1457	219	0.02 5
sg(2,8,5)	24	0.07	22K	125K	1.25	5320	3761	300	0.02	22K	123K	0.89	4962	2401	300	0.02 30
sg(3,5,4)	25	0.09	26K	155K	2.27	5645	4138	249	0.05	26K	152K	T/O	5222	2521	249	7.54 0.5
sg(3,6,4)	28	0.14	37K	221K	1.63	8072	5629	321	0.09	37K	219K	T/O	7562	3673	321	25.7 0.4
sg(3,7,4)	31	0.21	51K	299K	7.7	10K	7336	402	0.17	50K	296K	120	10K	5041	402	24.8 0.5
sg(4,5,4)	34	0.30	70K	430K	11.5	13K	9115	382	0.25	69K	426K	T/O	12K	6081	382	T/O 0.2
sg(4,6,5)	42	0.75	134K	837K	T/O	23K	15K	556	0.5	132K	831K	T/O	22K	11K	556	T/O 2
sg(4,7,4)	42	0.79	135K	829K	T/O	25K	16K	634	0.62	134K	824K	T/O	24K	12K	634	T/O 5
sg(4,9,4)	50	1.75	221K	1.35M	T/O	42K	25K	950	1.41	220K	1.34M	T/O	40K	20K	950	T/O 2.5
sg(5,4,3)	33	0.26	64K	394K	17.1	12K	8502	340	0.37	64K	391K	315	11K	5701	340	0.07 0.1
sg(5,5,4)	43	0.89	145K	911K	300	25K	16K	540	1.3	144K	906K	T/O	24K	12K	540	1.17 0.9
sg(5,7,4)	53	2.79	281K	1.76M	T/O	50K	30K	915	1.8	279K	1.75M	T/O	48K	23K	915	T/O 7
sg(5,8,3)	53	2.3	250K	1.51M	107	48K	29K	1050	1.76	248K	1.51M	T/O	47K	23K	1050	T/O 0.6
sg(6,4,3)	40	0.61	118K	733K	496	21K	14K	456	0.86	117K	729K	T/O	20K	9937	456	0.47 0.5
sg(6,5,3)	46	1.25	182K	1.13M	T/O	33K	20K	651	1.9	181K	1.12M	T/O	31K	15K	651	1.02 0.6
sg(6,6,3)	52	2.51	260K	1.61M	T/O	47K	28K	882	2.57	259K	1.60M	T/O	46K	22K	882	0.1 50
sg(7,5,3)	54	3.06	301K	1.89M	T/O	52K	32K	847	3.85	299K	1.88M	T/O	50K	24K	847	1.9 1K
sg(7,5,5)	68	11.4	551K	3.55M	T/O	87K	54K	1015	59.2	547K	3.53M	T/O	84K	41K	1015	37 20
hc(10,15,9)	38	0.07	32K	206K	93.4	5842	3762	45	0.59	32K	205K	T/O	5552	2701	45	T/O 7.2
hc(10,10,5)	28	0.04	19K	122K	T/O	3892	2487	45	22.2	19K	121K	T/O	3702	1801	45	T/O 0.4
hc(10,15,8)	38	0.07	32K	206K	T/O	5842	3762	45	275	32K	205K	T/O	5552	2701	45	286 N/A
hc(12,20,12)	50	0.19	66K	426K	T/O	11K	7023	66	2.77	65K	10K	424K	5281	10K	66	T/O N/A

Table 2: Results for social golfers and Hamming code generation problems. Best results for a given instance are boldfaced. T/O indicates timeout at 600s. The last column shows results from [15]. ‘K’ and ‘M’ in instance sizes indicate multiples of one thousand and one million. For UNSAT instances, using Pueblo with SBPs generally performs best. For SAT instances Pueblo is slowed down by SBPs, however ZChaff benefits from SBPs even on SAT instances. All runtimes are in seconds. N/A in the last two rows indicates that results for these instances are not shown in [15].

group. The present matrix model has symmetry along all three dimensions - groups, weeks and golfers. Adding pairwise constraints for specific golfers would leave only partial symmetry between golfers, which poses more effort for manual identification of symmetries. However, with our method added constraints can be analyzed and surviving symmetries detected without any modification. Even if row/column symmetry between certain rows and columns is destroyed, we can still detect symmetries that exist between specific variables in these rows and/or columns automatically. We also hope to identify problems that can be analyzed using our system, but for which matrix models are not applicable.

6 Conclusion

We present an integrated framework for studying and solving a class of CSPs by reduction to SAT and 0-1 ILP. The framework provides for the specification of constraints in a high-level language and automatic compilation into SAT. Specialized methods for SAT have improved considerably over the last 10 years, but these improvements do not necessarily apply to more sophisticated domains because SAT encodings are not always possible and may introduce inefficiencies due to the loss of *structure* in problem reductions. Our system automatically detects certain types of structure, such as linearity and symmetries during compilation and uses them to produce more efficient encodings. Linearity is preserved through the use of 0-1 ILP, a comparatively more sophisticated problem with specialized solvers that can use leading-edge techniques for SAT solving.

We extend earlier work on symmetry-detection in SAT and

0-1 ILP [10; 4] to a more general class of CSPs that may use non-binary variables and non-linear operations. Symmetries are detected in high-level input by solving the graph automorphism problem on parse trees. MinLex symmetry-breaking predicates (SBPs) from [10] are added to the resulting SAT/0-1 ILP encodings. Other work [15] has focused on symmetry-breaking ordering constraints for known or declared symmetries in generalized CSPs, but we detect and break symmetries automatically. Empirically, we evaluate our system on the balanced incomplete block design (BIBD), social golfers (SG) and Hamming code generation (HC) problems. We detect large numbers of symmetries in all instances, and show that breaking symmetries produces substantial speedups for the 0-1 ILP solver Pueblo [26] on unsatisfiable instances of the SG and HC problems. When symmetry-breaking is useful on unsatisfiable instances, the average speedup is 83.2 over the no-SBPs case (not including several timeouts). For satisfiable instances, there is a small slowdown with SBPs, but no timeouts. For CNF reductions, the SAT solver ZChaff [20] exhibits speedups for both satisfiable and unsatisfiable instances when symmetries are broken. Overall, CNF reductions are not competitive with 0-1 ILP reductions. A somewhat surprising observation is that on many satisfiable instances, Pueblo is slowed down by the addition of symmetry-breaking predicates (SBPs). This may be because adding SBPs to satisfiable instances prevents some solutions from being found by Pueblo. More effective SBPs need to be developed for this case. Overall, our runtimes for Pueblo with SBPs added are competitive with Solver runtimes reported in [15] on unsatisfiable instances of the SG and HC problems. We also show that our circuit-based CNF encodings for the BIBD problem are more efficient than those proposed in [24]. In general, our system facilitates the comparison of different SAT encodings, since any encoding can be plugged into our framework and automatically tested on several instances. This is useful since encodings often have a huge impact on search speed [29; 2; 5; 6]. Symmetries detected in high-level input can be used by *any* constraints solver and by other methods that add SBPs for declared symmetries during search [25; 13]. Moreover, SBPs can be added to a SAT/0-1 ILP reduction even if the actual encoding used obscures symmetries, since they are detected before reduction. We provide an extensible framework that can be easily modified to include other types of constraints and operations, and discuss two such extensions for symmetries due to *associative operations* and *value symmetries*. We plan to release code in the public domain to facilitate experimentation with different problems and encodings. At present, more information on this project, and contact addresses for source code, binaries and sample input files are available at [27].

Our current and future work is focused on extending our system to allow more comprehensive coverage of symmetries, e.g. symmetries in associative expressions and value symmetries briefly discussed in Section 4. We plan to extend our compiler to allow more operations and different types of constraints, and to support more OPL-like [22] syntax. Another direction is the development of efficient SBPs for non-binary variables and of symmetry-breaking constraints that are more effective on satisfiable instances.

References

- [1] F. A. Aloul, A. Ramani, I. L. Markov, K. A. Sakallah, "Solving Difficult SAT Instances In The Presence of Symmetry", *IEEE Transactions on CAD*, vol. 22(9), pp. 1117-1137, 2003.
- [2] F. A. Aloul, A. Ramani, I. L. Markov, K. A. Sakallah, "Generic ILP versus Specialized 0-1 ILP: An Update", in *Proceedings of the International Conference on Computer-Aided Design*, pp. 450-457, 2002.
- [3] F. A. Aloul, I. L. Markov, K. A. Sakallah, "Shatter: Efficient Symmetry-Breaking for Boolean Satisfiability", in *Proc. Intl. Joint Conf. on AI*, pp. 271-282, 2003.
- [4] F. A. Aloul, A. Ramani, I. L. Markov, K. A. Sakallah, "Symmetry-Breaking for Pseudo-Boolean Formulas", in *Proceedings of the Asia-South Pacific Design Automation Conference*, pp. 884-887, 2004.
- [5] O. Bailleux, Y. Boufkhad, "Efficient CNF Encoding of Boolean Cardinality Constraints", *Proceedings of Principles and Practice of Constraint Programming*, pp. 109-122, 2003.
- [6] O. Bailleux, Y. Boufkhad, "Full CNF Encoding: The Counting Constraints Case", in *7th Intl. Conf. on Theory and Applications of SAT Testing*, 2004.
- [7] M. Cadoli et. al, "NP-SPEC: An Executable Specification Language for Solving All Problems in NP", *Proceedings of Practical Aspects of Declarative Languages*, pp.16-30, 1999.
- [8] D. Chai, A. Kuehlmann, "A fast pseudo-boolean constraint solver", in *Proceedings of the Design Automation Conference*, pp.830-835, 2003.
- [9] C. H Colbourn, J. H. Dinitz, "The CRC Handbook of Combinatorial Designs", CRC Press, 1996.
- [10] J. Crawford, M. Ginsburg, E. M. Luks, A. Roy, "Symmetry-breaking predicates for search problems", in *Proc. of the Intl. Conf. on Principles of Knowledge Representation and Reasoning*, pp. 148-159, 1996.
- [11] P. Darga, "SAUCY Man Page", <http://vlsicad.eecs.umich.edu/BK/SAUCY/>
- [12] A.M. Frisch, I. Miguel, T. Walsh, "Cgrass: A System for Transforming Constraint Satisfaction Problems", *Jt. Workshop of ERCIM/CologNet area on Constr: Solving and Constr. Logic Prog.*, pp. 23-26, 2002.
- [13] I. P. Gent, W. Harvey, T. Kelsey, S. Linton, "Generic SBDD using Computational Group Theory", in *Principles and Practice of Constr. Prog.*, pp. 333-347, 2003.
- [14] I. P. Gent, T. Walsh, B. Selman, CSPLib Problem Library for Constraints; <http://www.csplib.org>
- [15] Z. Kiziltan, "Symmetry Breaking Ordering Constraints", *Doctoral Thesis*, Uppsala University, 2004.
- [16] A.Lal, B. Choueiry, "Dynamic Detection and Exploitation of Value Symmetries for Non-Binary Finite CSPs", *Workshop on Symmetry in CSPs*, 2003.
- [17] B. McKay, "Practical Graph Isomorphism", *Congressus Numerantium*, vol. 30, pp. 45-87, 1981.

- [18] F. Margot, "Exploiting Orbits in Symmetric ILP", Mathematical Programming Ser. B 98, pp. 3-21, 2003.
- [19] P. Meseguer and C. Torras, "Solving strategies for highly symmetric CSPs", in *Proceedings IJCAI*, pp. 400-405, 1999.
- [20] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, S. Malik, "Chaff: Engineering an Efficient SAT Solver", in *Proc. Design Automation Conf.*, pp. 530-535, 2001.
- [21] G. Nam, F. Aloul, K. Sakallah, R. Rutenbar, "A Comparative Study of Two Boolean Formulations of FPGA Detailed Routing Constraints", in *Proc. of the Intl. Symposium on Physical Design*, pp. 222-227, 2001.
- [22] P. van Hentenryck, "The OPL Optimization Programming Language", the MIT Press, 1999.
- [23] The ECLⁱPSE Team, "The ECLⁱPSE Constraint Logic Programming System":
<http://www.icparc.ic.ac.uk/eclipse/>
- [24] S. D. Prestwich, "Balanced Incomplete Block Design as Satisfiability", in *12th Irish Conference on Artificial Intelligence and Cognitive Science*, 2001.
- [25] J. F. Puget, "Symmetry Breaking Using Stabilizers", *Principles and Practice of Constraints Prog.*, pp. 585-599, 2003.
- [26] H. Sheini, "Pueblo": PB constraint solver;
<http://www.eecs.umich.edu/~hsheini/pueblo>
- [27] A. Ramani and I.L. Markov, "GSymEx": Generic Symmetry Extraction for Constraint Programming Problems;
<http://vlsicad.eecs.umich.edu/BK/GSymEx/>
- [28] ILOG Solver, <http://www.ilog.com/products/solver/>
- [29] J. P. Warners, "A Linear-Time Transformation of Linear Inequalities into Conjunctive Normal Form", in *Information Proc. Letters*, vol. 68(2), pp. 63-69, 1998.

Appendix: Compilation into SAT/0-1 ILP

Below, we describe how constraints are translated into CNF and 0-1 ILP. We use a C-like language for high-level constraint specification, and a customized parser that builds a parse tree for the system of constraints. Compilers for SAT and 0-1 ILP then walk the parse tree and translate the constraints into CNF/0-1 ILP formulas. The formulas are handed to SAT/0-1 ILP solvers and solutions are translated back into a form that is meaningful to the original problem. The input language uses C-like syntax to declare variables and specify constraints. Variables are specified as unsigned integers of varying bit sizes, e.g. int1 represents a 1-bit (binary) variable, etc. The mathematical operators allowed are addition (+), subtraction (-) and multiplication (*). Relational operators may be <=, >=, ==, and != (not-equal constraint). Complement notation is allowed to express the negative literal for a binary variable (x_1' for x_1). Numeric constants are allowed as coefficients or as the right-hand-side (RHS) value of equations. Division is not presently supported. The compiler also does not support the use of nested parentheses or unary negation but can be easily extended to do so. Support for

more sophisticated language constructs, e.g., those used by OPL [22], may be added in the future. An example of constraint declaration in the input language is shown in Figure 1 in Section 3.

To compile into SAT, Boolean "circuits" are instantiated to carry out mathematical operations. An n -bit variable is represented by n binary variables in the CNF instance plus a sign bit, which is necessary to perform subtraction with 2's complement notation. The size of the CNF circuits depends on the operation to be performed. Ripple-carry adders are instantiated for addition operations, and subtraction is performed using 2's complement representation. Both adder and subtractor circuits are linear in the input size. Multiplication is implemented using circuits for Booth's algorithm which are quadratic in the input size. Comparison against RHS values uses a linear comparator circuit. There are some built-in optimizations, e.g. smaller circuits for 1-bit addition and subtraction. 1-bit multiplication uses an AND gate. Circuits with a constant as input are partially evaluated. For compilation into 0-1 ILP, linearity is preserved by stating '+' and '-' operations directly as 0-1 ILP constraints. Inequalities (\leq , \geq , \neq) are also directly expressed in 0-1 ILP, with no need for comparator circuits. Coefficients can be directly written and not multiplied. Multiplication between variables uses CNF clauses, but multiplier outputs can be added/subtracted as part of a linear constraint.

Symmetry-Breaking and Local Search: A Case Study

Andrea Roli

Dipartimento di Scienze
Università degli Studi “G.D’Annunzio”
Pescara – Italy
a.roli@unich.it

Abstract

Symmetry-breaking has been proved to be very effective when combined with complete solvers. Conversely, it has been conjectured that the use of symmetry-breaking constraints has negative effect on local search-based solvers. This work presents an attempt to model the effect of symmetry-breaking on the search landscape explored by local search. The results, on the one hand, exclude that symmetry-breaking constraints negatively affect the topology of the search space. On the other hand, they strongly suggest that symmetry-breaking perturbs the configuration of local and global optima basins of attraction, making global optima more difficult to be reached.

1 Introduction

Symmetry-breaking has been proved to be very effective when combined with complete solvers [Crawford *et al.*, 1996; Puget, 2002]. This can be explained by observing that symmetry-breaking constraints considerably reduce the search space. Nevertheless, the use of symmetry-breaking constraints seem to have opposite effect on local search-based solvers, despite the search space reduction. In [Prestwich, 2001; 2002] some examples of this phenomenon are reported. When the problem is modeled with symmetry-breaking constraints, the search cost¹ is higher than the one corresponding to the model with symmetries. In [Prestwich, 2002], and also in [Ebner *et al.*, 2001] in a related context, it is suggested to use models that maximize the number of symmetries, contrarily to the case of complete solvers.

An important point to investigate is *why*, despite the search space reduction – that is often dramatic – local search-based techniques seem to be penalized by the introduction of symmetry-breaking constraints. Simplifying, the reasons why this happens can be either that (*i*) symmetry-breaking constraints modify the topology of the search landscape (e.g., by making the space not connected), or that (*ii*) they reduce the basins of attraction of global optima, or both.

In this paper, we introduce a model to study the effect of symmetry-breaking constraints on the search landscape ex-

plored by local search. This enables us to provide an abstraction of the system we want to investigate and to formulate general hypotheses, not bound to a particular problem. In the following sections, we will formulate more formally the search process performed by local search-based techniques and the conjectures we want to verify/falsify². The model we present introduces some simplifications and should be considered as a case study on the topic.

The structure of the paper is the following. In section 2, we define the model we use, along with the conjectures to be tested. Section 3 reports the experimental results concerning the previously defined hypotheses. Finally, we conclude in section 4 with a brief discussion and an outlook to the future.

2 A model for symmetry-breaking and local search

The aim of this section is to provide a model of the phenomenon we want to study. First of all, we will define the execution of a local search algorithm as a ‘walk’ along a graph. This enables us to outline important features of the search space and to relate them with the symmetry-breaking constraints. Then, we will introduce the kind of symmetries and the class of instances we consider.

Local search can be used to attack both constraint satisfaction problems and combinatorial optimization problems. The main operative difference is that, in the former case, the algorithm succeeds only if it finds a feasible solution to the original problem, which means that the local search algorithm must find a global optimum in the search landscape. In the latter case, the goal is usually to find, at least, a near-optimal solution. In the following, global optima can be considered either solutions better than any other solution or solutions belonging to the set of ‘acceptable solutions’, i.e., solutions considered good to the application at hand.

2.1 Local search and search graph

The local search process can be viewed as the exploration of a landscape aimed at finding a global optimum.

The *search landscape* is defined by a triple:

$$\mathcal{L} = (S, \mathcal{N}, F)$$

²According to [Popper, 2002], scientific empirical theories can only be falsified.

¹Runtime and number of variable flips

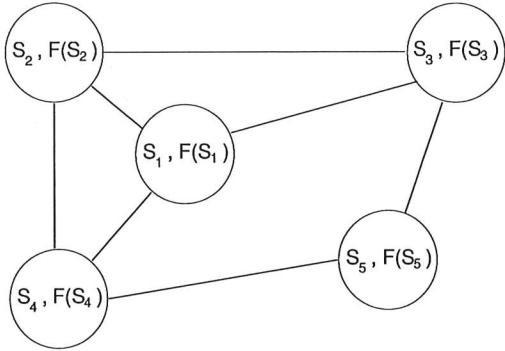


Figure 1: Example of undirected graph representing a search landscape. Each node is associated with a solution s_i and its corresponding fitness value $F(s_i)$. Arcs represent transitions between states by means of φ . Undirected arcs correspond to symmetric neighborhood structures.

where:

- S is the set of solutions (or states);
- \mathcal{N} is the neighborhood function $\mathcal{N} : S \rightarrow 2^S$ that defines the neighborhood structure, by assigning to every $s \in S$ a set of states $\mathcal{N}(s) \subseteq S$;
- F is the objective function $F: S \rightarrow \mathbb{R}^+$.

The search landscape can be interpreted as a graph (see Fig. 1) in which nodes are solutions (labeled with their fitness value) and arcs represent the neighborhood relation between states.

The neighborhood function \mathcal{N} implicitly defines an operator φ which takes a state s_1 and transforms it into another state $s_2 \in \mathcal{N}(s_1)$. Conversely, given an operator φ , it is possible to define a neighborhood of a variable $s_1 \in S$: $\mathcal{N}_\varphi(s_1) = \{s_2 \in S \setminus \{s_1\} \mid s_2 \text{ can be obtained by one application of } \varphi \text{ on } s_1\}$

Usually, the operator is *symmetric*: if s_1 is a neighbor of s_2 then s_2 is a neighbor of s_1 . In a graph representation (such as the one depicted in Fig. 1) undirected arcs represent symmetric neighborhood structures. A desirable property of the neighborhood structure is to allow a path from every pair of nodes (i.e., the neighborhood is strongly optimally connected) or at least from any node to an optimum (i.e., the neighborhood is weakly optimally connected). Nevertheless, there are some exceptions of effective neighborhood structures which do not enjoy this property [Nowicki and Smutnicki, 1996].

The search process of local search methods can be seen as the evolution in (discrete) time of a discrete dynamical system [Bar-Yam, 1997; Devaney, 1989]. The algorithm starts from an initial state (the initial solution) and describes a trajectory in the state space, that is defined by the search graph. The system dynamics depends on the strategy used; simple algorithms generate a trajectory composed of two parts: a *transient* phase followed by an *attractor* (a fixed point, a cycle or a complex attractor). Algorithms with advanced strategies generate more complex trajectories which can not be subdi-

vided in those two phases. The characteristics of the trajectory outline the behavior of the algorithm and its effectiveness with respect to the instance it is tackling. It is worth underlining that the dynamics is the result of the combination of problem representation, algorithm and instance. In fact, the problem representation defines the search landscape (and, in particular, the neighborhood structure defines the *topology* of the search landscape); the algorithm describes the strategy used to explore the landscape and, finally, the actual search space characteristics are defined by the instance to be solved.

For instance, let us consider a deterministic version of the Iterative Improvement local search (DII). The trajectory starts from a point s_0 , exhaustively explores its neighborhood, picks the neighboring state s' with minimal objective function value³ and, if s' is better than s_0 , it moves from s_0 to s' . Then this process is repeated, until a minimum \hat{s} (either local or global) is found. The trajectory does not move further and we say that the system has reached a fixed point (\hat{s}).

The relations between local search-based algorithms and dynamical systems is beyond the scope of this paper. Nevertheless, in the following, we will make extensive use of the notion of *basin of attraction* (BA), that stems from dynamical system theory. We will initially consider the case of deterministic systems, then we will relax this hypothesis and extend the definition to stochastic systems.

Definition Given a deterministic algorithm \mathcal{A} , the basin of attraction $\mathcal{B}(\mathcal{A}|s)$ of a point s , is defined as the set of states that, taken as initial states, give origin to trajectories converging to point s . The cardinality of a basin of attraction represents its size (in this context, we always deal with finite spaces).

Given the set S^* of the global optima, the union of the BA of global optima $I^* = \bigcup_{i \in S^*} \mathcal{B}(\mathcal{A}|i)$ represents the set of desirable initial states of the search. Indeed, a search starting from $s \in I^*$ will eventually find an optimal solution. Since it is usually not possible to construct an initial solution that is guaranteed to be in I^* , the ratio $|I^*|/|S|$ can be taken as an indicator of the probability to find an optimal solution. On the extreme case, if we start from a random solution, the probability to find a global optimum is exactly $|I^*|/|S|$. Therefore, the higher this ratio, the higher the probability of success of the algorithm.

In the case of stochastic local search, we may define a probabilistic basin of attraction, as a generalization of the previous case.

Definition Given a (stochastic) algorithm \mathcal{A} , the basin of attraction $\mathcal{B}(\mathcal{A}|s; p^*)$ of a point s , is defined as the set of states that, taken as initial states, give origin to trajectories converging to point s with probability $p \geq p^*$. Also in this case, we define the union of the BA of global optima: $I^*(p) = \bigcup_{i \in S^*} \mathcal{B}(\mathcal{A}|i; p)$. For simplicity, in the following we will write $\mathcal{B}(s; p^*)$ instead of $\mathcal{B}(\mathcal{A}|s; p^*)$ when the algorithm involved is clear from the context.

This definition includes the previous one as a special case.

³Ties are broken by enforcing a lexicographic order of states.

Indeed, if $p^* = 1$ we are interested in finding the states that will eventually converge to s . It is also important to note that if $p_1 > p_2$, then $\mathcal{B}(s; p_1) \subseteq \mathcal{B}(s; p_2)$.

Given a local search algorithm \mathcal{A} , the topology and structure of the search landscape determine the effectiveness of \mathcal{A} . In particular, the reachability of a global optimum is the key issue. Therefore, the characteristics of the BA of optimal solutions are of dramatic importance. The graph properties that affect the structure of BA are (*i*) the graph topology and (*ii*) the graph shape, namely, the number and distribution of local optima, the auto-correlation of the landscape, the presence/absence of plateaus, etc⁴. Hence, the goal of the algorithm designer is to have an algorithm \mathcal{A} such that the resulting total BA of global optima $I^*(p)$ is as large as possible, given a reasonably high value of p . It is clear that this is an *a posteriori* property, since it is the result of the application of a particular algorithm to a particular problem instance. However, this property can be used to explain why an algorithm is or is not effective on a problem instance, or a class of instances.

Finally, we would like to remark that, while the search graph topology is only dependent on the neighborhood structure, the BA and other related landscape characteristics (hereinafter referred to as search graph *shape*) depend also on the particular algorithm used.

In the following, we give the definition of a class of instances characterized by controlled properties of the search graph. This we will lead us to the study of the effect of symmetry-breaking constraints on the structure of the search graph and, consequently, on the behavior of local search algorithms.

2.2 A case study

In the following, we define the case study we analyze. We will first describe the search landscape that has some similarities with random landscapes and NK-landscapes [Kauffman, 1995; 1993]. Then, we will define the specific class of symmetries we consider in the model. Such a symmetry class is the one of permutations over problem variables.

The search graph

The class of problems we consider are defined over binary variables $x_i \in \{0, 1\}$, $i = 1, \dots, n$. Since our goal is to directly analyze the effect of symmetry-breaking constraints on the properties of the search graph, we directly define a model to construct a search landscape, abstracting from the specific problem we may deal with. Studies concerning random landscapes and NK-landscapes [Kauffman, 1995; 1993] can be found in the literature. In this work, we consider a landscape, in which the objective function is defined as follows. All the complete assignments are considered feasible (this is typically the case when local search is applied to problems defined on binary variables). Every assignment is given a value randomly chosen in a range $[r_1, r_2]$. Since usually the landscape has some degree of correlation, i.e., neighboring

⁴We forward the interested reader to specific literature on the topic [Hordijk, 1996; Merz and Freisleben, 1999; Reeves, 1999; Stadler, 1996].

solutions have objective values that are close, we introduce a distribution on the objective values such that some correlation is guaranteed. In practice, the distribution used is such that the higher the distance between two solutions, the larger the range of the difference of their objective values.

The topology of the graph is defined by the neighborhood structure. In our case, we use the neighborhood defined on unitary Hamming distance, that is the most used neighborhood for binary variables. Therefore, since n are the possible flips of an assignment, each node of the graph is connected with n other nodes. It is important to observe that this neighborhood structure generates a graph with a uniform degree (i.e., number of node edges) and this value is n . The impact of graph properties on system behavior has been recently received attention, as witnessed by the wide spectrum of publications on the subject [Watts and Strogatz, 1998; Adamic and Huberman, 2000].

Permutation symmetries

An important category of symmetries that can occur in combinatorial problems is the one of *permutation symmetries* [Gallian, 2001; Aloul *et al.*, 2002; Crawford *et al.*, 1996]. A permutation of a (finite) set Z is a function from Z to Z that is both injective and surjective. It is customary to use as set Z a (finite) set of naturals $\{1, 2, \dots, n\}$. A permutation can be expressed in the so called cycle notation, which is a composition of disjoint cycles of the form $(z_1 \dots z_m)$, $z_i \in \{1, 2, \dots, n\}$. For example, given $Z = \{1, 2, \dots, 6\}$, a permutation can be the composition of a 2-cycle and a 3-cycle such as $(1\ 2)(3\ 4\ 6)$, which means that 1 is mapped into 2 (and viceversa) and there is a cycle such that 3 is mapped into 4, 4 mapped into 6 and 6 into 3. Element 5 is mapped into itself. An important theorem [Gallian, 2001] states that every permutation in $\{1, 2, \dots, n\}$, $n > 1$, is either a 2-cycle or a product of 2-cycles⁵. Therefore, every possible permutation can be expressed as a composition of pairs $(z_i \ z_j)$, $i, j \in \{1, 2, \dots, n\}$.

We restrict our case study to permutation symmetries over a subset of problem variables. These symmetries can be expresses as a combination of 2-cycles. Moreover, we also add *phase shifts* in analogy with satisfiability problems. Phase shifts are such that exchanging a literal x_i with \bar{x}_i keeps the satisfying property of the assignments. We generated instances with n binary variables and m symmetries, each being either a 2-cycle or a phase shift. The m symmetries are randomly generated (without repetition), with the aim of covering a wide spectrum of cycle and phase shift combinations. The choice of random generation of cycles is motivated by the observation that, except for very specific cases occurring in structured instances, there is no regularity of permutation symmetries across the instances of a benchmark. The symmetries introduced by a 2-cycle $(i\ j)$ are simply cut by enforcing the constraint $x_i \leq x_j$ (we did not apply any reasoning to strengthen the constraints). Phase shifts (e.g., on variable x_l) are cut by posting the constraint $x_l = 0$. Symmetry-breaking constraints are combined and enforced in such a way that the resulting feasible solutions are the symmetry class representatives.

⁵The decomposition is not unique.

The effect of the symmetry-breaking constraints on the search graph can be directly analyzed by comparing the properties of the search graph with and without symmetry-breaking constraints. Since the search space is completely enumerated, only small size instances can be considered. In the next section, we present and discuss experimental results on the case study defined.

3 Experimental results

In [Prestwich, 2001; 2002] it has been conjectured that symmetry-breaking has a negative effect on local search-based techniques. From the standpoint of the model of local search process defined in section 2.1, we can formulate the hypothesis that symmetry-breaking constraints perturb the search graph in such a way that the algorithm effectiveness is reduced. The perturbation on the search graph can be of two kinds:

- topology perturbation
- shape perturbation (BA, local/global optima, etc.)

The negative effect could be originated from one or both the kinds of perturbation. In order to test this, we analyze the modification introduced in the search graph by symmetry-breaking constraints.

3.1 Topology perturbation

The topology of the search graph corresponding to the model with symmetries is highly regular (indeed, it is an hypercube) and each node has a degree equal to n (see an example in Fig.2). The graph is connected, therefore there is a path between any node and any global optimum. More important, given the structure of the graph, there are many alternative paths connecting every pair of nodes. Such a regular and redundant structure is particularly suitable for local search. ‘Irregular’ graph topologies, such as scale-free and small-world graphs [Watts and Strogatz, 1998; Strogatz, 1998; Barabasi, 2002], strongly affect the graph exploration process. As a consequence, it is natural to look for special topologies in graphs derived by models with symmetry-breaking constraints. An example of the resulting search graph is drawn in Fig.3, that is the counterpart of the one depicted in Fig.2. The outcome of our simulations, in which we tested graphs up to 10 variables⁶, can be summarized as follows. In the search graph related to the model with symmetry-breaking constraints, we observe that:

1. Node degree varies among nodes;
2. node degree frequency has a bell-shape, analogous to the Gaussian distribution characterizing random graphs (a typical case is reported in Fig.4);
3. the width of the bell-curve increases as the number of symmetries increases and it is independent of the relative number of 2-cycles and phase shifts.

Effect 1 is not surprising, since the constraints cut some parts of the search space, so they are very likely to introduce

⁶Since the analysis of the graph is complete, the instance size is strongly limited.

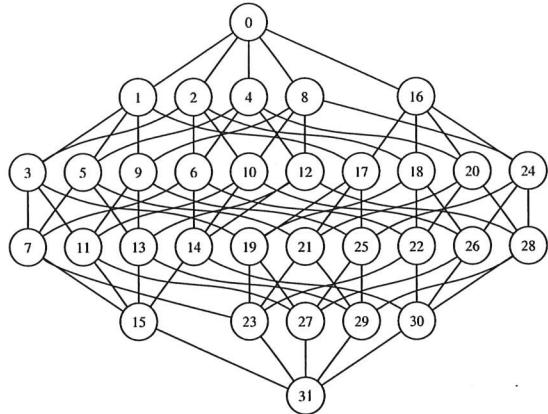


Figure 2: Search graph corresponding to an instance of size 5. The node degree is the same for every node and it is equal to 5.

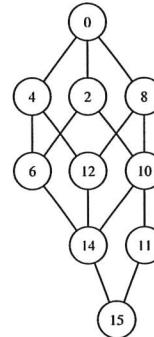


Figure 3: Search graph corresponding to an instance of size 5, with the following symmetries: (1 2), (1 4) and a phase shift involving variable x_5 . The maximum node degree is 4 and the minimum is 2.

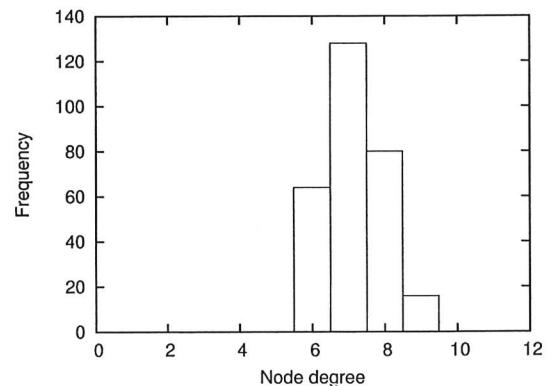


Figure 4: Node degree frequency of a search graph corresponding to an instance of size 10, with the following symmetries: (8 10), (6 8), (3 4), (6 10), (7 9). The maximum node degree is 9 and the minimum is 6.

differences in the node degree. However, the second effect is extremely important, since it definitely exclude the possibility that symmetry-breaking constraints introduce a graph topology that negatively affect local search behavior – at least in our model. In fact, the node degree frequency is such that the reachability of global optima is not dramatically perturbed, as can be also confirmed by observing the properties of random graphs [Newman *et al.*, 2001]. In fact, such graphs are connected, the average path length connecting any pair of nodes is quite short, there are many alternative paths connecting any pair of nodes and there are no bottlenecks. Finally, effect 3 just shows that, even if the number of symmetries increases, the graph topology keeps its basic characteristics. It is important to note that the graph is still connected and there are no disconnected regions.

Concluding, we can definitely reject the conjecture that local search effectiveness and efficiency are reduced by topological modification of the search space.

3.2 Basin of attraction perturbation

The second way symmetry-breaking affects the search space is by perturbing its shape. In particular, we studied how the basins of attraction of global optima vary as a consequence of symmetry-breaking constraints. The primary effect of symmetry-breaking is to reduce the number of optimal solutions, which seems to negatively influence the efficiency of local search [Clark *et al.*, 1996]. Nevertheless, this effect should be counterbalanced by the parallel reduction of local minima⁷.

Our experiments show that the relative size of basins of attraction of global optima is reduced in the model with symmetry-breaking constraints. We compared the relative size of global optima BA of instances of different size and different number of symmetries. Table 1 reports the statistics concerning the search graph with and without symmetries (results are averaged over 20 instances). The first two columns of the table report the number of variables and the number of symmetries, respectively. The third column is computed as follows. The ratio of the total size of the global optima BA and the number of feasible states is computed for both the model with symmetry-breaking constraints ($\Omega_s^* = |I_s^*|/|S_s|$) and without them ($\Omega^* = |I^*|/|S|$). The reported value is the average of the ratio Ω_s^*/Ω^* (along with the standard deviation in the fourth column). Very important are columns five and six, in which we reported the percentage of times Ω_s^* is smaller (col.5) or larger (col.6) than Ω^* . $\Omega_s^* < \Omega^*$ means that the fraction of nodes from which the algorithm can reach a global optimum is higher in the model without symmetry-breaking constraints. For example, consider the row corresponding to 10 variables and 1 symmetry: in the 60% of the instances, the percentage of states leading to a global optimum is lower in the model with symmetry-breaking constraints, and in the 30% is higher (for the remaining 10% is equal). When the total size of global optima BA is higher in the model with symmetries, it means that symmetry-breaking

⁷In some models, symmetry-breaking constraints introduce new local optima [Prestwich, 2002]. This is not the case for our model, that should be considered as a ‘best’ case.

constraints reduce the probability of reaching the global optima (at least in the deterministic case). We can observe that, in most of the cases, $\Omega_s^* \leq \Omega^*$. Therefore, despite the search space reduction achieved by symmetry-breaking constraints, a deterministic local search, such as DII, has not a higher probability of reaching a global optimum. Indeed, in most of the cases the inequality is strict.

The analysis we made only considered the deterministic case, however the results are an important clue about the general effect of symmetry-breaking constraints also concerning the stochastic case. In fact, even stochastic local search methods have a strong greedy component, that characterizes DII. Therefore, even if they can move away from ‘wrong’ basins of attraction, their efficiency is lessen by the reduction of global optima BA.

In summary, the experimental results enable us to reject the hypothesis that symmetry-breaking affects local search by perturbing the topology of the search space. Furthermore, we have some interesting clues concerning the effect of symmetry-breaking on the shape of basins of attraction.

4 Discussion and future work

In this work, we have presented a model to study the effect of symmetry-breaking constraints on local search and we have exemplified this methodology on a case study. The search landscape explored by a local search algorithm can be seen as a labeled graph and the algorithm execution as a path on this graph. Moreover, since relevant properties of the search process can be modeled via dynamical system theory, concepts such as attractors and basin of attractions can be used to characterize the algorithm execution. This standpoint enables us to study the effect of symmetry-breaking constraints on local search by analyzing their impact on the search landscape and, in particular, on its topology and its shape. We have reported a preliminary study based on an example of symmetric problem, characterized by permutation symmetries. Results exclude that symmetry-breaking constraints perturb the search graph topology in such a way that local search is penalized (at least for this case study). We have also found some interesting results indicating that the global optima basin of attractions are reduced in the model with symmetry-breaking constraints. This may be the reason why these constraints have a negative effect on local search behavior.

The problem model we used, while quite general and plausible as an abstraction of a real problem, has some limits, though. First of all, since the instances are randomly generated and only expressed by the composition of 2-cycles and phase shifts, the model does not capture the generality of cases that can occur in real-world problems. Furthermore, the experiments should be extended to study in detail the structure of the search graph, for instance by considering the reduction of number of optimal solutions. We should also experience with real problems and larger size instances and taking into account diverse local search algorithms, also stochastic.

An important observation concerns the rejection of the hypothesis on the perturbation of the search graph topology. This result enables us to conjecture that robust and

adaptive local search methods, such as Variable Neighborhood Search⁸, might not be negatively affected by symmetry-breaking constraints.

The problem of dealing with symmetries when a local search technique is applied is of considerable relevance, since stochastic approximate algorithms are often used to attack real-world problems. Very few works in the literature tackled this issue and there is a lot of room for future research. Besides studies on models, practitioners may need guidelines on both modeling and algorithm design. For instance, it would be extremely interesting trying to apply local search on models with symmetry-breaking constraints and to use these constraints in a tabu search fashion or by relaxing them whenever a diversification step is required. Finally, it would be of practical relevance the discovery of criteria to decide when using a model with symmetry-breaking constraints, or – at the other extreme – a super-symmetric model.

Acknowledgments

I thank Michela Milano for discussions and suggestions on this work. I also thank the anonymous reviewers for helpful comments.

References

- [Adamic and Huberman, 2000] L. A. Adamic and B. A. Huberman. Power-law distribution of the world wide web. *Science*, 287, 2000.
- [Aloul *et al.*, 2002] F. A. Aloul, A. Ramani, I. L. Markov, and K. A. Sakallah. Solving difficult SAT instances in the presence of symmetry. In *SAT 2002 – Fifth International Symposium on the Theory and Applications of Satisfiability Testing*, Cincinnati, Ohio, USA, 2002.
- [Bar-Yam, 1997] Y. Bar-Yam. *Dynamics of Complex Systems*. Studies in nonlinearity. Addison-Wesley, 1997.
- [Barabasi, 2002] A.-L. Barabasi. *Linked: The New Science of Networks*. Perseus Books Group, 2002.
- [Blum and Roli, 2003] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, September 2003.
- [Clark *et al.*, 1996] D. A. Clark, J. Frank, I. P. Gent, E. MacIntyre, N. Tomov, and T. Walsh. Local search and the number of solutions. In *Principle and Practice of Constraint Programming – CP96*, volume 1118 of *Lecture Notes in Computer Science*, pages 119–133. Springer-Verlag, 1996.
- [Crawford *et al.*, 1996] 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.
- [Devaney, 1989] R. L. Devaney. *An introduction to chaotic dynamical systems*. Addison-Wesley, second edition, 1989.
- [Ebner *et al.*, 2001] M. Ebner, M. Shackleton, and R. Shipman. How neutral networks influence evolvability. *Complexity*, 7(2):19–33, 2001.
- [Gallian, 2001] J. A. Gallian. *Contemporary Abstract Algebra*. Houghton Mifflin, 2001.
- [Hordijk, 1996] W. Hordijk. A measure of landscapes. *Evolutionary Computation*, 4(4):335–360, 1996.
- [Kauffman, 1993] S. A. Kauffman. *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press, 1993.
- [Kauffman, 1995] S. A. Kauffman. *At home in the universe*. Oxford University Press, New York, 1995.
- [Merz and Freisleben, 1999] P. Merz and B. Freisleben. Fitness landscapes and memetic algorithm design. In David Corne, Marco Dorigo, and Fred Glover, editors, *New Ideas in Optimization*, Advanced topics in computer science series. McGraw-Hill, 1999.
- [Newman *et al.*, 2001] M.E.J. Newman, S.H. Strogatz, and D.J. Watts. Random graphs with arbitrary degree distributions and their applications. *Phys. Rev. E*, 64, 2001.
- [Nowicki and Smutnicki, 1996] E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job-shop problem. *Management Science*, 42(2):797–813, 1996.
- [Popper, 2002] K. Popper. *The Logic of Scientific Discovery*. Routledge, 2002. First edition in English 1977.
- [Prestwich, 2001] S. Prestwich. First-solution search with symmetry breaking and implied constraints. In *Proc. of CP'01 Workshop on Modelling and Problem Formulation*, 2001.
- [Prestwich, 2002] S. Prestwich. Supersymmetric modeling for local search. In *Proc. of SymCon'02 Workshop on Symmetry and Constraint Satisfaction Problems*, 2002.
- [Puget, 2002] J. F. Puget. Symmetry breaking revisited. In *Principle and Practice of Constraint Programming – CP02*, volume 2470 of *Lecture Notes in Computer Science*, pages 446–451. Springer-Verlag, 2002.
- [Reeves, 1999] C. R. Reeves. Landscapes, operators and heuristic search. *Annals of Operations Research*, 86:473–490, 1999.
- [Stadler, 1996] P. F. Stadler. Landscapes and their correlation functions. *Journal of Mathematical Chemistry*, 20:1–45, 1996. Also available as SFI preprint 95-07-067.
- [Strogatz, 1998] S.H. Strogatz. Exploring complex networks. *Nature*, 410:268–276, 1998.
- [Watts and Strogatz, 1998] D.J. Watts and S.H. Strogatz. Collective dynamics of ‘small-world’ networks. *Nature*, 393:440–442, 1998.

⁸For a survey on local search algorithms and metaheuristics in general, see [Blum and Roli, 2003].

Table 1: Statistics of the relative size of total basins of attraction of global optima. Statistics are computed over 20 randomly generated instances (see description of the model in section 2). $\Omega_s^* = |I_s^*|/|S_s|$ and $\Omega^* = |I^*|/|S|$, where I_s^* and S_s (resp. I^* and S) are the total size of global optima BA and the number of feasible states in the search space in the model with symmetry-breaking constraints (resp. without).

n	m	$\langle \Omega_s^* / \Omega^* \rangle$	$\sigma(\Omega_s^* / \Omega^*)$	frac. $\Omega_s^* < \Omega^*$	frac. $\Omega_s^* > \Omega^*$
5	1	0.98	0.03	0.50	0.05
5	2	1.00	0.09	0.45	0.15
5	3	0.97	0.05	0.45	0.05
5	4	1.01	0.11	0.20	0.30
6	1	1.00	0.08	0.50	0.20
6	2	0.96	0.08	0.45	0.15
6	3	1.02	0.11	0.25	0.35
6	4	1.01	0.15	0.45	0.20
7	1	0.99	0.04	0.40	0.15
7	2	0.98	0.08	0.60	0.10
7	3	0.98	0.06	0.65	0.25
7	4	1.00	0.13	0.60	0.30
7	5	0.96	0.14	0.55	0.15
8	1	0.99	0.02	0.50	0.15
8	2	0.99	0.05	0.60	0.35
8	3	1.00	0.12	0.50	0.35
8	4	0.97	0.06	0.75	0.10
8	5	1.12	0.70	0.55	0.15
8	6	0.96	0.08	0.50	0.35
9	1	1.01	0.03	0.40	0.35
9	2	0.99	0.06	0.55	0.25
9	3	0.99	0.07	0.60	0.35
9	4	0.97	0.07	0.55	0.20
9	5	0.96	0.18	0.65	0.20
9	6	1.01	0.12	0.55	0.35
9	7	0.99	0.09	0.50	0.35
9	8	1.01	0.20	0.55	0.35
9	9	1.03	0.19	0.45	0.40
10	1	0.99	0.03	0.60	0.30
10	2	0.99	0.03	0.50	0.40
10	3	0.97	0.05	0.80	0.20
10	4	0.98	0.06	0.65	0.35
10	5	0.98	0.09	0.55	0.40
10	6	1.02	0.16	0.55	0.25
10	7	1.12	0.28	0.30	0.60
10	8	0.95	0.14	0.50	0.25
10	9	1.16	0.71	0.65	0.30
10	10	1.00	0.15	0.60	0.30

Conditional interchangeability and substitutability *

Yuanlin Zhang [†] and Eugene C. Freuder

Cork Constraint Computation Center
 University College Cork
 Cork, Republic of Ireland
 {y.zhang, e.freuder}@4c.ucc.ie

Abstract

In a constraint satisfaction problem, two values of a variable are interchangeable if every solution involving one value remains a solution with the value replaced by the other one. Although interchangeability occurs in many problems, there are also problems where little interchangeability occurs. In this paper, we study *conditional interchangeability and substitutability* for problems where values are interchangeable or substitutable under certain conditions.

1 Introduction

A *binary Constraint Satisfaction Problem (CSP)* consists of a set of variables, each of which has a finite domain, and a set of binary constraints on these variables. A solution to a CSP is an assignment of values to variables such that all constraints are satisfied.

Given a constraint satisfaction problem, two values of a variable are *interchangeable* [Freuder, 1991] if every solution involving one value remains a solution when the value is replaced by the other one. Derived from this concept of interchangeability are concepts of substitutability, partial interchangeability, and functional interchangeability etc. These variations are present in many real-world problems and help to solve, abstract, and compile CSPs [Freuder and Sabin, 1997; Rainer Weigel, 1999]. However, for some problems, there is little or no interchangeability. For example, consider a CSP with only one constraint $x < y$ with $x, y \in [1..10]$. No values of x (or y) are interchangeable, but if we know that $y > 6$, the values from 1 to 6 of x are interchangeable with each other. In other words, under certain conditions, some values of a variable become interchangeable.

In this paper, we introduce conditional interchangeability (CI) and conditional substitutability (CS), and their restricted version to neighboring variables. We also present alternative ways to express the 'condition' and compare this work with the existing work.

*This work has received support from Science Foundation Ireland under Grant 00/PI.1/C075.

[†]Current address: Department of Computer Science, Texas Tech University, Lubbock, Texas. Email: yzhang@cs.ttu.edu

2 Conditional interchangeability and substitutability

A CSP is usually represented as a triple (V, D, C) , where V is a set of variables $\{x_1, \dots, x_n\}$, D a set of domains $\{D_1, \dots, D_n\}$, and C a set of constraints. A constraint between x and y is denoted by c_{xy} . In this paper, the values of a variable are referred to by letters of a, b, c, d, e, f, \dots d is also used to denote the size of the maximum domain in a CSP. We assume that a constraint is given explicitly by a set of allowed tuples. For simplicity, $x.a$ refers to value a of variable x .

Definition 1 A condition on a set of variables X is defined as a set of constraints on these variables X .

The idea of conditional interchangeability is to consider the interchangeability under a solution space restricted by certain conditions, as shown by the example of $x < y$ in the first section whose solution space is reduced by the condition $y > 6$.

Definition 2 Given a CSP (V, D, C) and a variable x , two values a, b of x are conditionally interchangeable (CI) under condition Con iff they are interchangeable in $(V, D, C \cup Con)$.

In this section, a condition is assumed to consist of primitive constraints of the form $x = a$ where x is a variable and a a value, and logical conjunctions and disjunctions.

Example 0 Assume a CSP with a list of variables x, y, z has a solution space

$$\{(a, a, a), (a, b, c), (b, b, c), (a, c, c), (b, c, c), (b, b, b)\}.$$

A tuple, e.g., (a, b, c) , is an instantiation of the variables x, y, z in that order. The values a and b of x are not interchangeable due to the existence of the solutions (a, a, a) and (b, b, b) . After exposing the condition of $y = b \wedge z = c$ or $y = c \wedge z = c$, these two solutions are excluded, and thus values a, b of x are interchangeable. It can be written as

$$(y = b \wedge z = c \vee y = c \wedge z = c) \rightarrow x.a \equiv x.b$$

where \equiv denotes interchangeability.

It is straightforward to verify the following observation.

Proposition 1 Given a condition, CI is reflective, symmetric, and transitive.

As such, CI provides a way to put the values of a variable into equivalent groups.

Substitutability is defined in [Freuder, 1991] as follows. Given a CSP and two values a and b of a variable x , a is substitutable for b if any solution involving b remains a solution after a is substituted for b . Similar to the interchangeability under conditions, an otherwise non-substitutable value could become substitutable under some condition.

Definition 3 Given a CSP (V, D, C) and two values a and b of a variable x , a is conditionally substitutable for b under a condition Con iff it is substitutable for b in $(V, D, C \cup Con)$.

Consider Example 0 again. Neither a nor b of x is substitutable for the other. After introducing the condition

$$y = b \wedge z = c \text{ or } y = c \wedge z = c \text{ or } y = a \wedge z = a,$$

solution (b, b, b) is excluded and thus a is substitutable for b . However, under this condition, b is not substitutable for a and consequently not interchangeable with b . The fact that b is conditionally substitutable for a is denoted by

$$(y = b \wedge z = c \vee y = c \wedge z = c \vee y = a \wedge z = a) \rightarrow a \preceq b,$$

where \preceq means “substitutable for”.

Conditional substitutability describes a relationship between values of a variable, and has the following property.

Proposition 2 Given a condition Con , conditional substitutability is reflexive and transitive.

The following stronger substitutability will be useful later.

Definition 4 Given a CSP and a variable x , a value a of x is completely substitutable if it is substitutable for any other value in the domain of x .

The conditional version of this concept is given below.

Definition 5 Given a CSP (V, D, C) and a variable x , a value a of x is conditionally completely substitutable under condition Con iff it is completely substitutable in $(V, D, C \cup Con)$.

Since the solution space of a CSP is not known a priori, it is usually not easy to identify the (conditionally) interchangeable values in the problem. In the next section, we study the conditional interchangeability of the values of a variable x by considering only the constraints between x and its neighbors.

3 Conditional Neighborhood Interchangeability

We first explain some notations on the neighborhood of a variable. Given a constraint c_{xy} , value $a \in D_x$ is compatible with $b \in D_y$ if $(a, b) \in c_{xy}$ and a is also called a support of b . Two variables are neighbors if there is a constraint between them. $N(x)$ is used to denote an ordered list of all neighboring variables of x : $< x_1, \dots, x_l >$. Given a CSP and a variable x , the neighborhood subproblem on x refers to the problem of variable x , its neighbors, and the constraints between x and its neighbors. Note that the subproblem of x does not include any constraints between its neighbors. Once a CSP is restricted to a neighborhood subproblem, it is easy to recognize the conditions for the values to be interchangeable.

Definition 6 Given a CSP, two values a and b of a variable x are conditionally neighborhood interchangeable (CNI) under condition Con iff under Con , a and b are interchangeable with respect to the neighborhood subproblem on x .

The conditional interchangeability has the following relationship with CNI.

Proposition 3 Two values of a variable are conditionally interchangeable under a condition Con if they are conditionally neighborhood interchangeable under Con .

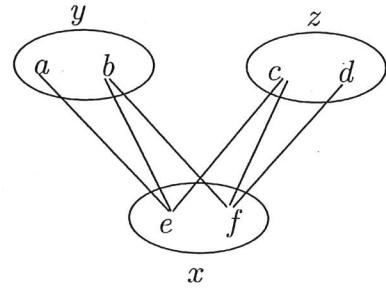


Figure 1: A CSP instance. The big circle represents a domain of a variable. The alphabets inside a circle are values. Two values are compatible if there is an edge between them. The constraint between two variables are exactly the set of all the edges connecting values of these variables.

Example 1 Consider the neighborhood subproblem, of a CSP, on x in Fig 1. values $e, f \in D_x$ are CNI under the condition $y = b \wedge z = c$. Hence,

$$y = b \wedge z = c \rightarrow e \equiv f.$$

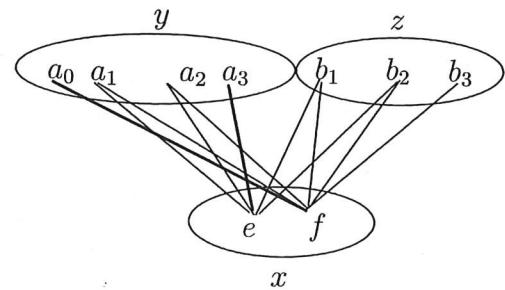


Figure 2: Values e and f are CNI under many conditions

Example 2 In Fig 2, e and f share many supports in y and z respectively. They are CNI under any one of the four instantiations of (y, z) : (a_1, b_1) , (a_1, b_2) , (a_2, b_1) , and (a_2, b_2) . We could have

$$y = a_1 \wedge z = b_1 \rightarrow e \equiv f,$$

$$y = a_1 \wedge z = b_2 \rightarrow e \equiv f,$$

$$y = a_2 \wedge z = b_1 \rightarrow e \equiv f,$$

$$y = a_2 \wedge z = b_2 \rightarrow e \equiv f.$$

Here we are interested in finding all conditions under which two values are CNI. Consider two values a and b of a variable x_i . Let $\langle x_1, \dots, x_l \rangle$ be the neighbors of x_i . For a neighboring variable $x_j (j \in 1..l)$, let $SS_j(\{a, b\})$ be the shared supports of a and b with respect to the constraint c_{ij} . Every tuple in $SS_1(\{a, b\}) \times \dots \times SS_l(\{a, b\})$ is a condition for a and b to be CNI. In the example above $SS_y(\{e, f\}) = \{a_1, a_2\}$ and $SS_z(\{e, f\}) = \{b_1, b_2\}$. The number of conditions can be exponential to the number of neighboring variables. Given the fact that all the conditions for a and b to be CNI are from a Cartesian product of their shared supports, the conditions can be simplified as

$$x_1 \in SS_1 \wedge \dots \wedge x_l \in SS_l \rightarrow a \equiv b. \quad (1)$$

Specifically, in Example 2, we have

$$y \in \{a_1, a_2\} \wedge z \in \{b_1, b_2\} \rightarrow e \equiv f. \quad (2)$$

It can be shown that the shared supports of two values provide the “weakest” condition under which they are CNI.

Proposition 4 *To represent the weakest condition for two values of x to be CNI, we need a space of size $O(|N(x)|d)$ where d is the size of the maximum domain of the problem of concern.*

Now let us turn to the neighborhood substitutability under conditions.

Definition 7 *Given a CSP and two values a and b of a variable x , value a is conditionally neighborhood substitutability (CNS) for b under condition Con iff under Con, a is substitutable for b with respect to the neighborhood subproblem on x .*

Example 3 Consider the CSP in Fig 3. Neither e nor f of variable x is substitutable for the other. If we restrict y to be in $\{a_1, a_2, a_3\}$, i.e., $y \in \{a_1, a_2, a_3\}$, value e is substitutable for f . Since e and f share the same supports in z , it is not necessary to put any condition on z for e to be substitutable for f . The substitutability of a for b under the condition above can be expressed as

$$y \in \{a_1, a_2, a_3\} \rightarrow e \preceq f.$$

Is there a condition under which e is completely substitutable (i.e., substitutable for f and g of x)? For this example, it can be verified that

$$y \in \{a_1, a_2, a_3\} \wedge z \in \{b_1, b_2\} \rightarrow e \preceq f \wedge e \preceq g.$$

In fact, we can identify a general condition for any value a of a variable x to be completely substitutable: If each neighboring variable of x contains only values that are supports of a , a is then substitutable for any other value of x .

Proposition 5 *Given a CSP and a value a of a variable x , for any neighboring variable x_j of x , let S_j be the set of supports for a with respect to the constraint on x and x_j . a is completely substitutable under the condition*

$$x_1 \in S_1 \wedge x_2 \in S_2 \wedge \dots \wedge x_l \in S_l.$$

Conditional neighborhood substitutability implies conditional substitutability.

Proposition 6 *Given a CSP and two values a and b of x , if a is conditionally neighborhood substitutable for b under condition Con, a is conditionally substitutable for b under condition Con.*

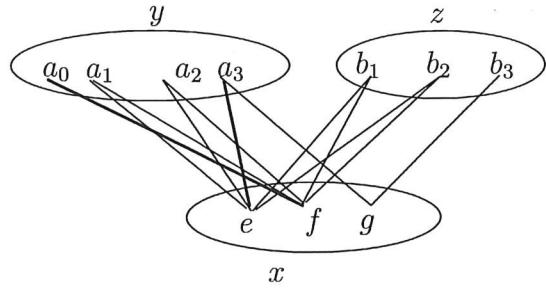


Figure 3: More complicated constraint c_{yx}

4 Extension of conditional interchangeability and substitutability

The interchangeability and substitutability of *values* of a variable can be generalized to those of *instantiations* of a set of variables. Since a set of variables can be regarded as one variable whose values are consistent instantiations of the original variables, it is not difficult to extend the conditional interchangeability and substitutability of values to those of instantiations.

Given a list of variables X , an instantiation of X is *consistent* iff it satisfies all the constraints involving only variables in X . We use \bar{a}, \bar{b}, \dots to refer to an instantiation of X .

We are interested in the set of variables each of which is outside X and is a neighbor of some variable in X . These variables are called neighbors of X and denoted by $N(X)$.

Two consistent instantiations of X are *interchangeable* iff every solution involving one instantiation remains a solution with the instantiation replaced by the other one.

Definition 8 *Given a CSP (V, D, C) and a set of variables X , let \bar{a} and \bar{b} be two consistent instantiations of X . \bar{a} and \bar{b} are conditionally interchangeable under condition Con iff they are interchangeable in $CSP(V, D, C \cup Con)$.*

The conditional substitutability of a consistent instantiation is defined below.

Definition 9 *Given a CSP (V, D, C) and a set of variables X , let \bar{a} and \bar{b} be two consistent instantiations of X . \bar{a} is conditionally substitutable for \bar{b} under condition Con iff it is substitutable for \bar{b} in $CSP(V, D, C \cup Con)$.*

Given a CSP, the neighborhood subproblem on X is the one consisting of X and $N(X)$, the constraints involving variables in X , and the constraints involving one variable in X and the other in $N(X)$. In other words, the subproblem on X includes constraints on X and constraints connecting a variable in X with a neighbor of X . Now we are able to list the neighborhood version of conditional interchangeability and substitutability.

Definition 10 *Given a CSP and two consistent instantiations \bar{a} and \bar{b} of a set of variables X , \bar{a} and \bar{b} are conditionally neighborhood interchangeable under condition Con iff under Con, \bar{a} and \bar{b} are interchangeable with respect to the neighborhood subproblem on X .*

Definition 11 Given a CSP and two consistent instantiations \bar{a} and \bar{b} of a set of variables X , \bar{a} is conditionally neighborhood substitutable for \bar{b} under condition Con iff under Con , \bar{a} is substitutable for \bar{b} with respect to the neighborhood subproblem on X .

In the following we discuss conditions to make instantiations interchangeable or substitutable.

For any consistent instantiation \bar{a} of X , and a neighbor $y_i \in N(X)$, let $S_i(\bar{a})$ be the set of values of y_i that are compatible to \bar{a} , i.e., each value of $S_i(\bar{a})$ and \bar{a} satisfy the constraints between y_i and any variable in X . Given two instantiations \bar{a} and \bar{b} of X and a neighbor y_i , their shared support set, denoted by $SS_i(\{\bar{a}, \bar{b}\})$, is the intersection of the support set of \bar{a} and that of \bar{b} . Assuming that $N(X) = \{y_1, y_2, \dots, y_m\}$, we are able to list the general conditions for neighborhood interchangeability

$$y_1 \in SS_1 \wedge y_2 \in SS_2 \wedge \dots \wedge y_m \in SS_m \rightarrow \bar{a} \equiv \bar{b}$$

where $SS_i, i \in 1..m$, refers to $SS_i(\{\bar{a}, \bar{b}\})$.

Under the condition

$$y_1 \in S_1(\bar{a}) \wedge y_2 \in S_2(\bar{a}) \wedge \dots \wedge y_m \in S_m(\bar{a}),$$

\bar{a} is completely substitutable, i.e., substitutable for every other consistent instantiation of X .

5 On the application of conditional interchangeability and substitutability

In the case of checking the satisfiability of a CSP, conditional interchangeability and substitutability may be used to prune the search space. The conditional interchangeability, as an equivalence relation, partitions the values of a variable into equivalent groups. Given a group of interchangeable values (under certain condition), we can choose to keep only one value of the group in the domain of the variable while not affecting the satisfiability of the original problem. The reason is that if there is any solution including another member of the group, it remains a solution if we replace the member by the value we choose to keep.

Consider a value a of a variable x . Assume x has l neighbors, for neighbor $x_i (i \in 1..l)$, S_i is the support set of a , and SS_i is the shared support set of a and another value b of x .

By the condition in (1), if we have

$$x_1 \in SS_1 \wedge \dots \wedge x_l \in SS_l,$$

we can keep a and prune b and all other values which are interchangeable with a under this condition.

By the condition in Proposition 5, conditional neighborhood substitutability results in

$$x_1 \in S_1 \wedge x_2 \in S_2 \wedge \dots \wedge x_l \in S_l \rightarrow x = a. \quad (3)$$

Concerning the pruning ability, the CNS is clearly more powerful than CNI. CNI can only remove values interchangeable with a while CNS removes all other values. Furthermore, each SS_i is a subset of S_i , implying that the CNS provides a weaker premise than CNI.

In the rest of this section, we study the relationship between the concepts here and those in the work reported in [Bowen and Likitvivatanavong, 2004; Prestwich, 2004; Chmeiss and Sais, 2003].

5.1 Domain transmutation

The work by Bowen and Likitvivatanavong embeds the idea of conditional neighborhood interchangeability without explicitly introducing condition. They introduce the concept of domain transmutation by splitting a value into two or merging two values in terms of CNI. In other words, [Bowen and Likitvivatanavong, 2004] creates virtual values for those conditions making values interchangeable.

Consider Example 1 (Fig 1). For values e and f of x , their shared supports are $SS_y = \{b\}$, $SS_z = \{c\}$. We know that

$$y \in SS_y \wedge z \in SS_z \rightarrow e \equiv f.$$

But we can not remove either e or f because they are interchangeable only when the condition holds. The method in [Bowen and Likitvivatanavong, 2004] simply introduces a new value, say g , whose support sets are exactly SS_y and SS_z . Now that some role of e has been assumed by y , we only need to figure out the *other* role e plays when the condition is not true (i.e., $y \neq b \vee z \neq c$). When $y \neq b$, e is supported by a of y and c of z , but when $z \neq c$, e is not supported by any value of z and thus all its supports in y are useless. See the picture in Fig. 4 for the supports of e . Similarly, when the condition is false, the supports of f are $\{b\}$ (for y) and $\{d\}$ (for z).

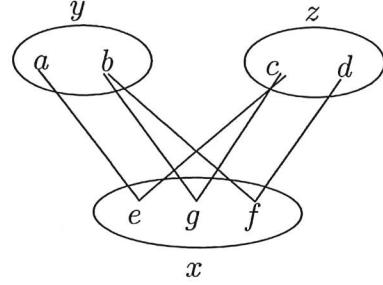


Figure 4: New value g is introduced and the constraints c_{xy} and c_{xz} are updated to reflect the conditional neighborhood interchangeability

5.2 “Interchangeability constraints”

Since the work in [Prestwich, 2004] is based on a different representation of constraints, to establish the connection, we first explain a “nogood” (disallowed) representation of a constraint.

Consider the constraint c_{yx} in Example 2 (Fig 2). The disallowed tuples by c_{yx} are:

$$\{(a_0, e), (a_3, f)\}.$$

That (a_0, e) is not allowed is expressed by \neq and logical connectives

$$y \neq a_0 \vee x \neq e.$$

Similarly, disallowing (a_3, f) is expressed as

$$y \neq a_3 \vee x \neq f.$$

To represent c_{yx} , we connect the two formulae above by and:

$$(y \neq a_0 \vee x \neq e) \wedge (y \neq a_3 \vee x \neq f). \quad (4)$$

Now a constraint is represented by a formula with conjunctive normal form, instead of a set of allowed tuples. Consequently, the conjunction of all constraints of CSP is still of conjunctive normal form. The CSP in Example 2 is represented as

$$\begin{aligned} & (y \neq a_3 \vee x \neq e) \wedge (y \neq a_0 \vee x \neq f) // c_{yx} \\ & \wedge \\ & (z \neq b_3 \vee x \neq e) // c_{zx} \end{aligned}$$

We are now able to introduce the idea to prune values in [Prestwich, 2004]. Given a variable x and a term $x \neq a$, we first select all conjuncts containing $x \neq a$ from all constraints (for simplicity, we consider only binary constraints here although non binary constraints can be treated similarly)

$$\begin{aligned} x \neq a \vee x_1 \neq a_{11} \\ \vdots \\ x \neq a \vee x_1 \neq a_{1n_1} \\ x \neq a \vee x_2 \neq a_{21} \\ \vdots \\ x \neq a \vee x_2 \neq a_{2n_2} \\ x \neq a \vee x_l \neq a_{l1} \\ \vdots \\ x \neq a \vee x_l \neq a_{ln_l} \end{aligned} \quad (5)$$

Note, in the formulae above, we group the conjuncts with the same variables together. For instance, the first group involves x and x_1 .

By assuming an ordering " \leq " on the values in the domain of x , Prestwich gives the following pruning "constraint" [Prestwich, 2004] for all b such that $a \leq b$,

$$\begin{aligned} & x_1 \neq a_{11} \wedge \cdots \wedge x_1 \neq a_{1n_1} \wedge \\ & x_2 \neq a_{21} \wedge \cdots \wedge x_2 \neq a_{2n_2} \wedge \cdots \wedge \\ & x_l \neq a_{l1} \wedge \cdots \wedge x_l \neq a_{ln_l} \rightarrow x = b. \end{aligned} \quad (6)$$

This constraint is named by Prestwich as **interchangeability constraints (IC)**. Its premise is the conjunctions of all the conjuncts in (5) with $x \neq a$ removed. Since conjuncts in (5) include *all* those involving $x \neq a$, if the premise of (6) is true, we can simply let x be a , which makes all other \neq 's on x (in the whole CSP of concern) true and thus the satisfiability of the whole problem is not affected. The IC can be strengthened as follows

$$\begin{aligned} & x_1 \neq a_{11} \wedge \cdots \wedge x_1 \neq a_{1n_1} \wedge \\ & x_2 \neq a_{21} \wedge \cdots \wedge x_2 \neq a_{2n_2} \wedge \cdots \wedge \\ & x_l \neq a_{l1} \wedge \cdots \wedge x_l \neq a_{ln_l} \rightarrow x = a. \end{aligned} \quad (7)$$

In fact, with the traditional representation of constraints in mind, $x_1 \dots x_l$ are exactly the neighbors of x , and $a_{i1} \dots a_{il}$ are the values which are not consistent with a of x . The constraint (7) could be written as

$$\begin{aligned} & x_1 \notin (D_1 - S_1) \wedge x_2 \notin (D_2 - S_2) \wedge \cdots \wedge \\ & x_l \notin (D_l - S_l) \rightarrow x = a \end{aligned} \quad (8)$$

where S_i is the support set of a of x with respect to x_i , as defined as before. Obviously, this pruning constraint is equivalent to (3) that is derived from CNS directly.

In summary, the conditional neighborhood substitutability facilitates stronger pruning and more intuition than IC's.

5.3 Generalized neighborhood substitutability

In this subsection, we switch back to the traditional representation of a constraint as a set of allowed tuples.

The generalized neighborhood substitutability (GNS) proposed in [Chmeiss and Sais, 2003] says that two values of a variable is GNS iff they share at least one support with respect to each neighboring variable.

There is a relationship between GNS and CNI.

Proposition 7 *Two values are GNS iff there exists a condition Con such that they are CNI under Con.*

However, when a value is CNS for another value, these two values might not be GNS. For example, consider e and g of x in Fig. 3. e is CNS for g , but they do not share any support in the domain of variable z and thus they are not GNS.

A "constraint" to prune search space is also proposed in [Chmeiss and Sais, 2003]. By assuming a total ordering on the values, the key component of that pruning constraint on a variable x can be translated, by using notations developed here, to

$$\begin{aligned} & x_1 \in S_1 \wedge x_2 \in S_2 \wedge \cdots \wedge x_l \in S_l \\ & \rightarrow (x = a_1 \vee x = a_2 \vee \cdots \vee x = a_m \vee x = a) \end{aligned} \quad (9)$$

where a is a value of x , $a_1 \dots a_m$ are the values smaller than a , $x_1 \dots x_l$ are the neighbors of x , and again S_i is the support set of a with respect to x_i .

It is interesting to observe that this constraint is equivalent to IC (6) if the same ordering on values is used in both constraints.

As we have been aware, under the premise of (9), a is completely substitutable for every other value of x , and thus the pruning constraint can be strengthened by letting $x = a$, equivalent to (3).

There is no obvious connection between the constraint (9) and GNS. But the relationship between the specific pruning constraints (3) and CNS is immediate.

6 Other related work

6.1 Partial interchangeability

Two values of a variable x are *partially interchangeable* [Freuder, 1991] with respect to a set of variables X iff any solution involving one implies a solution involving the other with possibly different values for X . Partial interchangeability is a special type of conditional interchangeability where the condition is on the assignments of X .

In this section, we present a result on a property of partial interchangeability. If x is not a neighbor of any variable in X , partial interchangeability is equivalent to interchangeability.

Proposition 8 *Given two values a and b of a variable x and a set X , if a and b are partially interchangeable with respect to X , and x is not a neighbor of X , then a and b are interchangeable.*

Proof. Consider any solution where x takes value a . Let \bar{a} be the list of values for X in the solution and a' the list of values for other variables in the solution. We represent the solution by a list (a, \bar{a}, a') . We need to prove that (b, \bar{a}, a') is also a solution.

Since a and b are partially interchangeable, there exist a list of values \bar{b} for X such that (b, \bar{b}, a') is a solution. The fact that x is not a neighbor of any variable in X implies that the neighbors (recalling the definition of the neighbors of a set of variables) of X have the same values in both solutions. Both \bar{a} and \bar{b} are consistent with the values taken by their neighbors. Hence, replacing \bar{b} in the second solution with \bar{a} , (b, \bar{a}, a') is still a solution.

Similarly, we can show that any solution containing b remains a solution by substituting a for b . Hence, they are interchangeable. \square

6.2 Context dependent interchangeability

Context dependent interchangeability (CDI) [Weigel *et al.*, 1996] is equivalent to conditional interchangeability. Instead of focusing on the neighborhood of a variable, [Weigel *et al.*, 1996] resorts to a rather sophisticated decomposition method to identify CDI values under certain “conditions”.

The identification of conditional neighborhood interchangeability is at least tractable for binary CSPs and could be a practical tool to prune the search space.

6.3 Domain partition

The idea in [Haselbock, 1993] is that although two values of a variable may not be neighborhood interchangeable, they could be (fully) interchangeable with respect to only one, rather than all, neighboring variable. Based on this observation, the domain of a variable is partitioned with respect to each constraint on it such that values in each partition are interchangeable with respect to a certain constraint. One immediate advantage of this method is that a filtering procedure (e.g., AC algorithms) could be implemented more efficiently by taking each partition as one value. A search procedure is also introduced to make use of the neighborhood interchangeability (but not CNI or CNS).

6.4 Inferred Disjunctive Constraints

The inferred disjunctive constraints [Freuder and Hubbe, 1993] make use of the complete substitutability of CNS and some other observations to decompose a CSP.

7 Conclusion

When two values of a variable are not (neighborhood) interchangeable or substitutable, there exists some “interchangeability” and “substitutability” among them under some condition. The condition is usually a restriction on the domain of each neighboring variable. we propose conditional (neighborhood) interchangeability and substitutability which could be used to prune search space. They further strengthen the pruning constraints and concepts proposed by Prestwich [Prestwich, 2004] and Chmeiss & Sais [Chmeiss and Sais, 2003]. They also offer a uniform perspective on the previous work (e.g., [Bowen and Likitvivatanavong, 2004;

Freuder and Hubbe, 1993; Prestwich, 2004; Chmeiss and Sais, 2003]). Prestwich has studied IC's, based on SAT solver. We are planning experiments to study the efficiency to prune the search space by using conditional neighborhood substitutability in a CSP solver.

8 Acknowledgement

We thank Chavalit Likitvivatanavong, Barry O'Sullivan, Steve Prestwich, and Nic Wilson for useful discussions on the topics covered in this material.

References

- [Bowen and Likitvivatanavong, 2004] James Bowen and Chavalit Likitvivatanavong. Domain transmutation in Constraint Satisfaction Problems. To appear in Proceedings of AAAI-04, 2004.
- [Chmeiss and Sais, 2003] A. Chmeiss and L. Sais. About neighborhood substitutability in CSPs. In *Proceedings of SymCon-03*, pages 41–45, Kinsale, Ireland, 2003.
- [Freuder and Hubbe, 1993] Eugene C. Freuder and Paul D. Hubbe. Using inferred disjunctive constraints to decompose constraint satisfaction problems. In *Proceedings of IJCAI-93*, pages 254–260, Chambery, France, 1993. AAAI press.
- [Freuder and Sabin, 1997] Eugene C. Freuder and Daniel Sabin. Interchangeability supports abstraction and reformulation for multi-dimensional constraint satisfaction. In *Proceedings of AAAI/IAAI-97*, pages 191–196, Providence, Rhode Island, 1997. AAAI press.
- [Freuder, 1991] E.C. Freuder. Eliminating interchangeable values in constraint satisfaction problems. In *Proceedings of AAAI-91*, pages 227–233, Anaheim, California, 1991. AAAI press.
- [Haselbock, 1993] A. Haselbock. Exploiting interchangeabilities in Constraint Satisfaction Problems. In *Proceedings of IJCAI-93*, pages 282–287, Chambery, France, 1993. IJCAI Inc.
- [Prestwich, 2004] Steven Prestwich. Exploiting full interchangeability. Manuscript, 2004.
- [Rainer Weigel, 1999] Boi Faltings Rainer Weigel. Compiling constraint satisfaction problems. *Artificial Intelligence*, 115(2):257–287, 1999.
- [Weigel *et al.*, 1996] R. Weigel, B. V. Faltings, and B. Y. Choueiry. Context in Discrete Constraint Satisfaction Problems. In *Proceedings of ECAI-96*, pages 205–209, Budapest, Hungary, 1996. John Wiley & Sons, Ltd.