Principles and Practice of Constraint Programming (CP 2005)

WORKSHOP PROCEEDINGS:

Symmetry and Constraint Satisfaction Problems

October 1*st*, 2005
Melia Hotel, Sitges SPAIN

Karen E. Petrie Meinolf Sellmann
Jean-Francois Puget (Eds.)

Symmetry and Constraint Satisfaction Problems

Fifth International Workshop
Sitges (Barcelona), Spain, 1st October 2005
Proceedings

Held in conjunction with the
Eleventh International Conference on
Principles and Practice of
Constraint Programming (CP 2005)

Programme Committee

Table of Contents

Some improvements in symmetry elimination in not-equals binary constraint networks

Belaïd Benhamou and **Mohamed Réda Saïdi**
Laboratoire des Sciences de l'Information et des Systmes (LSIS)
Centre de Mathmatiques et d'Informatique
39, rue Joliot Curie - 13453 Marseille cedex 13, France
email:Belaid.Benhamou@cmi.univ-mrs.fr, saidi@cmi.univ-mrs.fr

Abstract

Detecting and eliminating symmetrical values when solving CSPs reduces drastically the search space. Symmetrical values with a given value are in a sense redundant, their removal simplifies the problem without affecting its consistency. Verifying the conditions of symmetry is in general a hard task, but in not-equals binary constraint networks, these conditions can be simplified. In this paper, we show how some symmetrical value are eliminated when the instantiation of the current variable to a value of its domain fails. We give a simple sufficient condition proving that these eliminated values are symmetrical with the value of the current variable. A Linear time complexity algorithm witch verifies this sufficient condition and which detects symmetrical values is proposed. These symmetries are exploited in a simplified forward checking method adapted to solve not-equals CSPs. This method is experimented on both randomly generated instances of graph coloring and Dimacs graph coloring benchmarks. The obtained results shows that symmetry elimination is a considerable improvement for solving graph coloring.

1 Introduction

Constraint Satisfaction Problems (CSPs) are at the heart of many applications in artificial intelligence, such as in visualization of scenes in CAO, scheduling problems, program verification, model checking. Several problems are expressed and solved in the framework of constraints.

A CSP is a set of constraints where each constraint involves a subset of the CSP variables. The main question is how to assign each variable a value of its domain without violating a constraint. This problem is known to be Np-complete.

In practice several methods and techniques are proposed. The backtracking search method and its improvements [Haralik and Elliot, 1980; Sabin and Freuder, 1997] are often used to solve CSPs. Sometimes, the backtracking method can be improved by simplifying the problem by verifying some k-consistencies [Freuder, 1978], or by using the techniques of CSP decomposition [Jégou, 1990; 1993; Dechter and Pearl, 1987; 1989] or those of symmetry elimination [Freuder, 1991; Benhamou, 1994; Crawford *et al.*, 1996; Focacci and Milano, 2001; Fahle *et al.*, 2001; Puget, 2001].

We investigate in this article symmetry in *not-equals binary constraints* CSPs. In theory, there is no matter to restrict our study to this framework, since each CSP can be reduced to a not-equals CSP. Graph coloring fits in this framework and solving not-equals CSPs is in general an NP-complete problem. Besides, in practice, this framework is quite expressive, it covers a broad range of problems in artificial intelligence, such as time-tabling and Scheduling, Register Allocation in compilation, and cartography [A.Ramani *et al.*, 2004].

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

We show in this article how the symmetry condition given in [Benhamou, 2004] is weakened in the case of failing to instantiate a variable with a value of its domain during the search. We give a more simplified symmetry condition which leads to a symmetry detection algorithm whose efficiency is better and which detects more symmetries than the algorithm defined in [Benhamou, 2004].

The rest of this article is organized as follow: Section 2 gives a brief background on CSPs. In section 3 we discuss the symmetry notion and show how the symmetry condition given in [Benhamou, 2004] is weakened in order to detect efficiently new symmetries in not-equal CSPs. We show in section 4 how the symmetry property is exploited in a Forward Checking backtracking method adapted to not-equals CSPs. In section 5 we evaluate and compare the effectiveness of our result by carrying experiments on both Dimacs graph coloring benchmarks and randomly graph coloring generated instances. Section 6 concludes the work.

2 CSPs formalism

A CSP (Constraint Satisfaction Problem) as defined in [Montanari, 1974; Mackworth, 1977] is a quadruple $P = (X, D, C, R)$ where: $X = \{X_1, ..., X_n\}$ is a set of n variables; $D = \{D_1, ..., D_n\}$ is the set of finite discreet domains associated to the CSP variables, D_i includes the set of possible values of the CSP variable X_i; $C = \{C_1, ..., C_m\}$ is a set of m constraints involving some subsets of the CSP variables.

A binary constraint is a constraint which involves two variables; $R = \{R_1, ..., R_m\}$ is a set of relations corresponding to the constrains of C, R_i represents the list of value tuples permitted by the constraint C_i. A CSP P can be represented by a constraint graph $G(X, E)$ where the set of vertices X is the set of the CSP variables and each edge of E connects two variables involved in the same constraint $C_i \in C$.

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

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

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

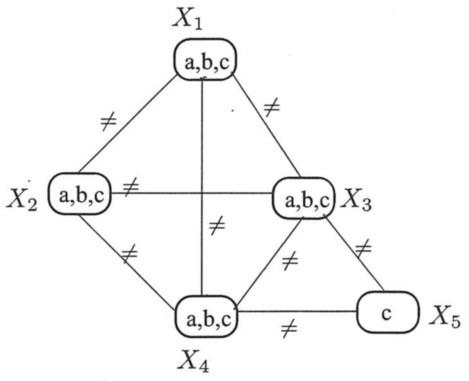

Figure 1: Constraint graph of a NECSP

3 Symmetry in CSPs

We recall in this section some definitions and properties first introduced into [Benhamou, 1994] which we will use to prove our result on symmetry in NECSPs. Some proofs are omitted for space reason.

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 $\forall d_i \in D_i$, $\sigma(d_i) \in D_i$.*

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

In other words, a symmetry of a CSP $\mathcal{P} = (V, D, C, R)$ is a permutation of domain values that leaves the CSP $\mathcal{P}$ invariant, i.e. $\sigma_R(R_{ij}) = R_{ij}$ [1].

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 $\sigma(c_i) = b_i$*

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: $I/\sigma[v_i] = \sigma(I[v_i]), \forall i \in [1, n]$. The followin property 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* 1** *See [Benhamou, 1994].*

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* 2** *See [Benhamou, 1994].*

Corollary 3.1 *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.*

Corollary 3.1 allows to remove symmetrical values with a value $d_i \in D_i$ without affecting the CSP consistency if the value d_i is already shown to not participating in any solution of the CSP.

3.1 Symmetry in NECSPs

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

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

***Proof* 3** *See [Benhamou, 2004].*

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

[1] σ_R: is the generalization of σ to the relations of R

3.2 The weakened symmetry condition

Before introducing the new symmetry property, we define the notion of assignment trees and failure trees corresponding to the enumerative search method used to prove the consistency of the considered CSP.

Definition 3.4 *We call an assignment tree of a CSP $\mathcal{P}$, a tree which gathers the history of all the variable assignments made during its consistency proof, where the nodes represent the variables of the CSP and where the edges outcomming from a node X_i are labeled by the different values used to instantiate the corresponding CSP variable X_i.*

The assignment tree of a CSP is sensitive to the order in which the variables are instantiated during its consistency checking. The root of the tree is the first variable in the ordering. In the sequel, we consider only assignment trees corresponding to the Forward Checking [Haralik and Elliot, 1980] method. Forward Checking method is an improvement of the classical CSP backtraking which performs a look ahead filtering consisting in removing in the domains of the not yet instantiated variables, the values which do not support the instanciation of the current variable.

Example 3.1 *Take the CSP of Figure 1 and apply a forward checking process on it w.r.t the variable ordering $\{X_1, X_2, X_3, X_4, X_5\}$. Figure 2 illustrates the assignment tree of the considered CSP.*

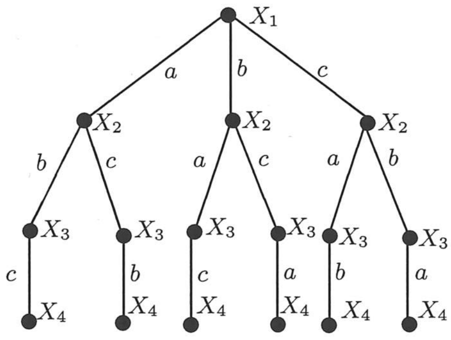

Figure 2: Assignment tree

In an assignment tree of a CSP, a path connecting the root of the tree to a node defines a partial instantiation of the CSP. The variables of the partial instanciation are the nodes of the considred path. The last node of the path corresponds to the last affected variable in the instantiation.

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

Definition 3.5 *Let T be an assignment tree corresponding to a consistency proof of a CSP $\mathcal{P}$, $I = (a_1, a_2, ..., a_i)$ an inconsistent partial instanciation of the variables $X_1, X_2, ..., X_i$ corresponding to the path $\{X_1, X_2, ..., X_i\}$ in T. We call a failure tree of the instanciation I, the sub-tree of T noted by $T_{I=(a_1,a_2,...,a_i)}$ such that:*

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

Example 3.2 *Let us consider the assignment tree of the figure 2 corresponding to the CSP of Example 2.1. If we take as a partial instantiation $I = (b, a)$ which assigns X_1 to the value b and X_2 to the value a, then the failure tree $T_{I=(b,a)}$ of the instanciation I is shown in the figure 3 (the part in a box).*

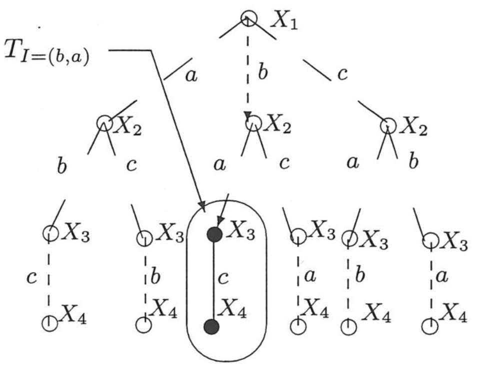

Figure 3: failure tree of the partial instanciation $I = (b, a)$

We can now give the new symmetrey property which represents the principal contribution of this work. The main idea is to weaken the symmetry condition of theorem 3.2 when an inconsistent partial instantiation is generated during the search.

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

***Proof* 4** *Let $\mathcal{P}'(V', C', D', R')$ be a sub-CSP of the CSP $\mathcal{P}(V, C, D, R)$ such that $V' = Var(T_{I=(a_1,...,a_{i-1},a_i)}) \cup X_i$ and $C' \subseteq C$, $D' \subseteq D$ and $R' \subseteq R$ are the restrictions of C, D, and R to the variables of V'. By the hypothesis, $T_{I=(a_1,...,a_{i-1},a_i)}$ is a failure tree of I in $\mathcal{P}$. This implies that the assignment of X_i to the value a_i leads to a failure in $\mathcal{P}'$. In other words, a_i do not participate in any solution of $\mathcal{P}'$. By the hypothesis, both values a_i and b_i appear in the same domains of the variables of $Var(T_{I=(a_1,...,a_{i-1},a_i)})$. This means that a_i and b_i appear in the same domains of non-instantiated variables of the CSP $\mathcal{P}'$. By application of Theorem 3.2 we deduce that the* *value b_i is symmetrical to a_i in $\mathcal{P}'$. By application of corollary 3.1, we deduce that the value b_i do not participate in any solution of $\mathcal{P}'$. This implies that the partial instanciation $J = I_0 \bigcup \{b_i\} = (a_1, ..., a_{i-1}, b_i)$ is inconsistent in $\mathcal{P}$. (QED)*

This new property of symmetry is a weakening of the condition of Theorem 3.2 in the case of instanciation leading to an inconsistency. The case of inconsistent instantiation is important, because it allows to prune the consistency proof tree of a CSP $w.r.t$ Corollary 3.1.

Some symmetries not captured by Theorem 3.2 can result from this new condition. Let us consider for instance the CSP of Figure 1. If we take the inconsistent partial instanciation $I = (b, a)$ of the variables X_1 and X_2, then the two values a and c of the domain of the current variable X_2 are symmetrical by application of the new theorem 3.3, whereas they are not symmetrical by application of the theorem 3.2. The branch corresponding to the assignment of X_2 to c is not explored in the consistency proof tree thanks to Theorem 3.3. This defines a symmetry cut which we use in the section 4 to shorten the proof tree of CSP consistency checking.

Remark 3.1 *Theorem 3.2 can be used to compute symmetrical solutions, whereas theorem 3.3 cannot, it is used only to avoid some redundant partial inconsistent instantiations.*

3.3 Symmetry detection

Now we deal with the symmetry detection problem. To be symmetrical, values have to satisfy the weakened symmetry condition of theorem 3.3. The algorithm sketched in Figure 4 computes the symmetrical values with a value a_i of a given domain $w.r.t$ the condition of theorem 3.3. These values form the class of symmetry of a_i which we denote by cl(a_i)).

```
procedure $weak\_symmetry(a_i \in D_i, Var(T_{I=(a_1,...,a_i)})$,
var cl($a_i$):class);
input: a value $a_i \in D_i$, a set of variables $Var(T_{I=(a_1,...,a_i)})$
Output: the class $cl(a_i)$ of symmetrical values to $a_i$.
begin
    cl($a_i$):={$a_i$}
    for each $d_i \in D_i$-{$a_i$} do
        if for each domain $D_k$ of variables of
        $Var(T_{I=(a_1,...,a_i)})$
        we have
            ($a_i \in D_k$ and $d_i \in D_k$)
            or ($a_i \notin D_k$ and $d_i \notin D_k$)
        then cl($a_i$):=cl($a_i$)∪{$d_i$}
end
```

Figure 4: The algorithm of search for symmetries in NECSPs

Complexity:
Let n be the number of variables of the NECSP, and d the size of the largest domain. It is easy to see that the algorithm of Figure 4 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. This algorithm has a linear complexity $w.r.t$ the NECSP size.

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

4 Exploiting symmetry in NECSPs

Now, we show how the symmetry property given in Theorem 3.3 is exploited to increase the efficiency of NECSP backtracking algorithms. For efficiency reasons, we implement a *Simplified Forward Checking* method (denoted by SFC) adapted to NECSPs which we want improve by adding the symmetry property.

The principle of the Forward Checking [Haralik and Elliot, 1980] is based on filtering the domains of the non-instantiated variables $w.r.t$ the instantiated one. When the current variable v_i is instantiated with a value d_i, the domains of the non-instantiated variables (called future variables) having a constraint with v_i are filtered such that all the values which do not support the value d_i of the current variable v_i are removed.

In the case of NECSPs, the filtering is simplified. It consists only in removing the value d_i from the domains of the future variables having a constraint with v_i. This results in a *Simplified Forward Checking* which we considered in our implementation.

If during the filtering, a domain of a future variable v_j becomes empty, Forward Checking stops the filtering, its effects are undone, and the current variable v_i is instantiated with a new value from its domain. If all possible instantiations for the current variable v_i are inconsistent then a backtracking is made and the variable v_{i-1} becomes the current variable. Otherwise, if v_i could be instantiated with no constraint violation then the next variable to instantiate is selected among the future variables, and the same process is repeated.

Forward Checking can have different performances according to the variable ordering in the instantiation. Several heuristics exist, in practice, the one minimizing the ratio

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

where $Degree(v_i)$ denotes the number of constraints of the initial CSP in which the variable v_i is involved, is one of the most effective to chose the next variable to instantiate. This heuristic is known under the name DomDeg. We use it in SFC to select the future variable to instantiate.

Theorem 3.3 allow to prune k-1 branches in the search tree if there are k symmetrical values and one of them is shown to not participating in any solution. If $SymClass(d_i)$ denote the class of values of the domain D_i symmetrical to d_i, then we consider only the value d_i, since the other values of $SymClass(d_i)$ are redundant.

```
Procedure SFC-sym-weak($D, I, k, Var(T_I)$);
input: a set of domains $D$, $I = (d_1, ..., d_k)$ a partial instantiation
of variables $\{v_1, ..., v_k\}$; $k$ the index of the current variable and $Var(T_I)$
the set of variables of the failure tree $T_I$ (at beginning $Var(T_I)$ is empty).
var empty:boolean;
begin
    if $k = n$ the $[d_1, d_2, \ldots, d_k]$ is a solution, stop
    else
    begin
        empty:=false;
        for each $v_i \in V$, such as $C_{ik} \in C, v_i \in$ 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;
                        add($v_i, Var(T_I)$);
                        weak_symmetry($d_k \in D_k$,$Var(T_I)$,SymClass($d_k$));
                        $D_k$=$D_k - SymClass(d_k)$;
                        empty:=true;
                    end
                end
        if not(empty) then
        begin
            add($v_k, Var(T_I)$);
            $v_{k+1}$=next-variable($v_k$)
            repeat
                take $d_{k+1} \in D_{k+1}$
                $D_{k+1}$=$D_{k+1} - d_{k+1}$
                $I = [d_1, d_2, \ldots, d_k, d_{k+1}]$;
                SFC-sym-weak($D, I, k+1, Var(T_I)$);
                weak_symmetry($d_{k+1} \in D_{k+1}$,$Var(T_I)$,SymClass($d_{k+1}$));
                $D_{k+1}$=$D_{k+1} - SymClass(d_{k+1})$;
            until $D_{k+1} = \emptyset$
        end
    end
end
```

Figure 5: SFC method combined with the new symmetry detection algorithm

4.1 Combining trivial symmetry with the detected one

Some trivial symmetrical values can be exploited without effort of detection. Indeed, all the values which form the intersection of the domains of a NECSP are trivially symmetrical.

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

***Proof* 5** *The proof is trivial, since all the values in Ω appear in all the domains, thus the condition of Theorem 3.2 is hold.*

The trivial symmetries are very important, since during search, the values of the subset Ω which are not used in the partial instantiation remain symmetrical at each node of the search tree. We consider only one of them and the others are, in fact, removed . These symmetries are very useful for solving graph coloring where all the variables have identical domains. There is one domain D_i=$\{0, \ldots, c-1\}$ of c values and Ω=D_i in this case. If during the search all the first values $\{0, \ldots, mdn\}$ of the ordered finite domain $D_i = \{0, \ldots, c-1\}$ (with $0 \leq mdn \leq c-1$) are used in the partial instantiation I, then the values of the part $\{mdn+1, \ldots, c-1\}$ remain symmetrical. All the values of the domain D_i are trivially symmetrical before starting variable instantiation ($mdn = 0$).

Such trivial symmetries are 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 $\{mdn+1, \ldots, c-1\}$ of trivial symmetrical values. This subset becomes empty when mdn reaches the value $c-1$. On the other hand, the subset $\{0, \ldots, mdn\}$ of the used values increases and become quickly identical to the whole domain D_i.

A lot of symmetries exist between the values of the part $\{0, \ldots, mdn\}$ which we detect by using the symmetry procedure of figure 4. The trivial symmetry of the part $\{mdn+1, \ldots, c-1\}$ and the one we detect on the part $\{0, \ldots, mdn\}$ 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 at given search node, then $c - mdn - 1 + k$ symmetry cuts can be made when both kind of symmetry are associated to prune the search space.

Figure 5 sketches the SFC procedure augmented by the symmetry property of theorem 3.3 and the DomDeg heuristic (notation SFC-sym-weak). The structure $future(v_i)$ encodes the set of non-instantiated variables remaining after the instantiation of v_i and *next-variable* a function which encodes the (DomDeg) heuristic. In the sequel SFC will denote the SFC method augmented by the DomDeg heuristic.

5 Experiments

We will now evaluate the performances of our implementation. The tests are made on both randomly generated graph coloring instances and some graph coloring benchmarks of the 2^{nd} challenge of Dimacs (`http://dimacs.rutgers.edu/Challenges/`). The graph coloring problem consists in coloring the n vertices of a graph with m colors, such that no two adjacent vertices have the same color. This problem is trivially expressed as a NECSP. We will test and compare both the Simplified Forward Checking augmented by the symmetry property defined in [Benhamou, 2004] (SFC-sym) and the Simplified Forward Checking augmented by the advantage of the symmetry property of theorem 3.3 (SFC-sym-weak). The complexity indicators are the number of nodes and CPU time. The source codes are written in C and compiled on a machine equipped with an AMD Athlon XP 2200+processor with 512 MB.

5.1 Random graph coloring problems

Random graph coloring problems are generated according to the parameters:(1) n the number of vertices (the variables), (2) Cls the number of colors (the domain values) and (3) d the density which is a number between 0 and 1 expressing the ratio:

$$d = \frac{number\ of\ constraints}{total\ number\ of\ possible\ constraints}$$

For each test corresponding to some fixed value of the parameters n, Cls and d, a sample of 100 instances are randomly generated and the measures (CPU time, nodes) are taken in average.

Tables 1 and 2 give the performances of the two methods SFC-sym and SFC-sym-weak on random graph coloring

Pb	SFC-sym		SFC-sym-weak		
Cls	Nodes	Time	Nodes	Time	%Cons
13	69	0.0	67	0.0	0.0
14	479	0.0	467	0.0	0.0
15	7014	0.0	6843	0.0	0.0
16	181448	2.8	177024	2.7	17.0
17	1894312	32.5	1837245	32.0	45.0
18	288708	4.6	279559	4.5	100.0
19	7760	0.1	7389	0.1	100.0
20	252	0.0	239	0.0	100.0
21	102	0.0	101	0.0	100.0

Table 1: Graph coloring instances with $n = 100$ and $d = 0.5$

Pb	SFC-sym		SFC-sym-weak		
Cls	Nodes	Time	Nodes	Time	%Cons
35	114061	2.6	110578	2.6	0.0
36	1645219	39.7	1585537	38.4	0.0
37	5327514	110.3	5152504	106.8	15.0
38	18553369	374.8	17992800	365.5	55.0
39	2638689	52.7	2520893	50.25	95.0
40	437893	7.15	423600	6.9	100.0
41	163319	2.4	157073	2.35	100.0
42	14743	0.2	14008	0.1	100.0

Table 2: Graph coloring instances $n = 100$ and $d = 0.9$

problems. The number of variables is fixed to $n = 100$ and $d = 0.5$ for the instances of Table 1 and $d = 0.9$ for the ones of Table 2 . Both tables give the number of colors (Cls) of the problem, the number of nodes (Nodes), the CPU time in seconds (Time) for both methods and the percentage of consistency (%Cons). We can see that SFC-sym-weak, generates in general less nodes than SFC-sym and spend less time than SFC-sym to solve the problems. This proves that SFC-sym-weak detects and eliminate more symmetries than SFC-sym and the detection is faster in SFC-sym-weak. The gain is not very important on the random problems, but the symmetry behavior is shown and there is no case where SFC-sym is faster than SFC-sym-weak.

5.2 Dimacs graph coloring benchmarks

Table 3 shows the results of the two methods on some graph coloring benchmarks of Dimacs. These problems are known to be hard. We seek for each problem the minimal number k of colors needed to color the vertices of a given graph (the chromatic number). The search of the chromatic number consist in proving the consistency of the problem with k colors (existence of a k-coloration of the graph); and in proving its inconsistency when using $k-1$ colors. The symbol "?" means that the corresponding method does answer the question in 12 hours.

We can remark that only SFC-sym-weak is able to solve the problems "mulsol.i.1", "fpsol2.i.3", and "fpsol2.i.2", and SFC-sym-weak outperforms drastically SFC-sym on the problems "miles1500", and "fpsol2.i.1". On the other problems both methods compare.

Pb		SFC-sym		SFC-sym-weak		
instance	k	Nodes	Time	Nodes	Time	Consistency
queen8_8	8	1284400	8.5	1265496	8.4	no
queen8_8	9	5869838	34.3	5820278	34.4	yes
le450_5a	4	23	0.0	18	0.0	no
le450_5a	5	125546	4.7	124275	4.8	yes
le450_5b	4	6	0.0	6	0.0	no
le450_5b	5	9898	0.4	9869	0.4	yes
myciel5	5	32133	0.12	31504	0.12	no
myciel5	6	46	0.0	46	0.0	yes
mulsol.i.1	48	?	?	70162	1.83	no
mulsol.i.1	49	196	0.0	196	0.0	yes
mulsol.i.4	30	3454	0.0	3454	0.0	no
mulsol.i.4	31	184	0.0	184	0.0	yes
fpsol2.i.1	64	4353805	149.76	9189	0.7	no
fpsol2.i.1	65	495	0.3	495	0.3	yes
fpsol2.i.2	29	?	?	384612	14.5	no
fpsol2.i.2	30	450	0.1	450	0.1	yes
fpsol2.i.3	29	?	?	384612	13.4	no
fpsol2.i.3	30	424	0.1	424	0.1	yes
zeroin.i.1	48	152	0.0	53	0.0	no
zeroin.i.1	49	210	0.0	210	0.0	yes
zeroin.i.3	29	49	0.0	31	0.0	no
zeroin.i.3	30	205	0.0	205	0.0	yes
school1	13	36923	1.9	13199	0.7	no
school1	14	107498	5.5	47877	2.7	yes
school1_nsh	13	63	0.0	56	0.0	no
school1_nsh	14	734	0.0	589	0.0	yes
miles750	30	50890	0.4	167	0.0	no
miles750	31	127	0.0	127	0.0	yes
miles1500	72	6030066	104.8	14079	0.3	no
miles1500	73	127	0.0	127	0.0	yes
1-Fullins_4	4	21876	0.1	17667	0.1	no
1-Fullins_4	5	92	0.0	92	0.0	yes
2-Fullins_3	4	29099	0.1	5806	0.0	no
2-Fullins_3	5	51	0.0	51	0.0	yes

Table 3: Dimacs graph coloring benchmarks

6 Conclusion

In this work we weakened the symmetry condition and improved the symmetry property in not-equals constraint networks when an inconsistent partial instantiation is generated. We implemented a more efficient symmetry search algorithm which detects more symmetries and exploit the new symmetry property in a a Forward Checking backtracking algorithm adapted to NECSPs. Experiments are carried on both random generated graph coloring problems and Dimacs graph coloring benchmarks. The obtained results show that the new symmetry property is a real improvement for solving NECSPs.

Further investigation consists first in combining some clique search algorithms, or CSP decomposition methods with our method to improve graph coloring solving. An other interesting point is to extend the new symmetry property to general CSPs.

References

[A.Ramani *et al.*, 2004] A.Ramani, F.A.Aloul, I.L.Markov, and K.A.Sakallak. Breaking instance-independent symmetries in exact graph coloring. *In Proceeding of Design Automation and Test in Europe*, pages 324–329, 2004.

[Benhamou, 1994] B. Benhamou. Study of symmetry in constraint satisfaction problems. *In the working notes of the workshop PPCP'94*, 1994.

[Benhamou, 2004] B. Benhamou. Symmetry in not-equals binary constraint networks. *SymCon'04 : 4th International Workchop on Symmetry and Constraint Satisfaction Problems*, 2004.

[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.

[Dechter and Pearl, 1987] R. Dechter and J. Pearl. The cycle-cutset method for improving search performance in ai applications. *Proceedings of the Third IEEE Conference on AI Applications*, 1987.

[Dechter and Pearl, 1989] R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38:353– 366, 1989.

[Fahle *et al.*, 2001] T. Fahle, S. Schamberger, and M. Sellmann. Symmetry breaking. In *International conference on constraint programming*, volume 2239 of *LNCS*, pages 93–108. Springer Verlag, 2001.

[Focacci and Milano, 2001] 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.

[Freuder, 1978] E.C. Freuder. Synthesing constraint expressions. *CACM*, 21(11):958–966, 1978.

[Freuder, 1991] E.C. Freuder. Eliminating interchangeable values in constraints satisfaction problems. *Proc AAAI-91*, pages 227–233, 1991.

[Haralik and Elliot, 1980] R. M. Haralik and G. L. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence 14*, pages 263–313, 1980.

[Jégou, 1990] F. Jégou. Cyclic-clustering : A compromise between tree-clustering and cycle-cutset method for improving search efficiency. *ECAI1990*, pages 369–371, 1990.

[Jégou, 1993] F. Jégou. Decomposition of domains based on the micro-structure of finite constraint-satisfaction problems. *AAAI1993*, pages 731–736, 1993.

[Mackworth, 1977] A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence 8*, pages 99–118, 1977.

[Montanari, 1974] U. Montanari. Networks of constraints : Fundamental properties and applications to picture processing. *Information Science 7*, pages 95–132, 1974.

[Puget, 2001] J.F. Puget. Symmetry breaking revisited. In *International conference on constraint programming*, volume 2470 of *LNCS*, pages 446–461. Springer Verlag, 2001.

[Sabin and Freuder, 1997] D. Sabin and E. Freuder. Understanding and improving the mac algorithm. In *CP97*, pages 167–181, 1997.

Algebraic Structure Helps in Finding and Using Almost-Symmetries

Igor L. Markov
Department of EECS, The University of Michigan, Ann Arbor, MI 48109-2122
imarkov@eecs.umich.edu

Abstract

Many successful uses of symmetries in discrete computational problems rely on group-theoretical properties. For example, large sets of permutational symmetries can be compactly represented by small sets of generators and manipulated using stabilizer-chain algorithms. While almost-symmetries may be more numerous than symmetries, no group-like properties or efficient algorithms are currently known for them. In this work, we identify an algebraic structure formed by almost-symmetries and, using it, develop a complete "life-cycle" for almost-symmetries, including their discovery, compact representation and symmetry-breaking.

1 Introduction

A structure-preserving reversible transformation of a structured object is often called a symmetry (or *automorphism*), with prime examples being (1) a permutation of vertices of a given graph, that maps edges to edges and preserves vertex labels, (2) a permutation of variables in a system of constraints, that preserves the system, (3) a permutation of possible values of some variables, such as the simultaneous negation of two Boolean (or two integer) variables, (4) a permutation of input and output variables of a Boolean function that leaves the function invariant. Computational applications often involve finite domains. For example, identifying a pair of symmetric variables x and y in an equation allows one to introduce the additional constraint $x \leq y$ so as to reduce the amount of searching by up to 25%, or closer to 50% for non-Boolean variables. Combining many such symmetries sometimes reduces the complexity of search, proofs or refutations from exponential to polynomial, both in provable lower bounds [18] and empirical performance [1]. Recent progress in understanding and manipulating symmetries accelerates the solution of large, practical Boolean equations [1], instances of Integer Linear Programming [3; 4], and other discrete problems. This is often accomplished through the use of fast symmetry detection [20; 8] and *symmetry-breaking* techniques [7; 13; 2]. A broad range of affected applications include (i) formal verification of microprocessors and software, (ii) optimal scheduling and planning, (iii) protein folding and other optimization problems in natural sciences. A rich literature exists on semantic symmetries of Boolean functions and describes many uses in synthesis and optimization of digital circuits [21, ibid]. For example, in a given VLSI layout one may permute the input pins of a single AND gate or a whole 64-bit multiplier to improve wiring congestion by uncrossing wires [6].

Identifying and using a greater variety of symmetries often improves computational efficiency. Hence, it is natural to relax the notion of symmetry and deal with *almost-symmetries* which retain useful properties, occur more often and have greater impact. Such extensions have already been studied for tilings and partial differential equations in Mathematics [22; 12], quasicrystals and approximate conservation laws in Physics [16; 22], stars and planets in Astrophysics [23], molecular configurations in Chemistry [17; 9] and computational problems in Artificial Intelligence [15].

To be useful, the effort to exploit symmetries or almost-symmetries should be justified by tangible benefits, especially when symmetry-agnostic processing is possible [14; 8]. Such trade-offs are poorly understood for almost-symmetries. In Section 2 we show that for regular symmetries, computational improvements can only be achieved by using the algebraic structure of symmetries. The latter observation is driving much of our work on almost-symmetries as outlined below.

Existing techniques for using symmetries tend to be undermined by the *freedom of choice*. Namely, all computational techniques cited above assume *simultaneous* constraints, as in Boolean satisfiability, but are not sensitive to *disjunctive* constraints involving multiple variables [11]. In another example, the behavior of digital circuits is often left underspecified on invalid inputs (resolved later so as to simplify the design), which leads to Boolean functions with *don't-cares*. In Section 3 we study possible generalizations of symmetries in these cases. It turns out that relevant almost-symmetries cannot be composed as freely as regular symmetries and therefore do not form groups, cosets, semi-groups, monoids, groupoids, or other named algebraic structures. However, any meaningful properties carried by almost-symmetries can potentially be interpreted as a type of structure. We identify such an algebraic structure in Section 4 and show how it helps to compactly represent almost-symmetries, facilitating symmetry-finding in Section 5 and almost-symmetry-breaking in Section 7. In Section 6 we discuss possible generalizations and applications of almost-symmetries.

2 Necessary Background

Below we review a representative modern paradigm for exploiting *syntactic symmetries* in search and discrete optimization [1; 2], and then contrast it with *semantic (functional) symmetries* and related notions of *interchangeability* and *substitutability* [24].

Finding and representing symmetries efficiently. Consider an instance of constraint-programming or combinatorial optimization where all constraints must be simultaneously satisfied. To identify its semantic symmetries, all variables, constraints and optimization objectives are represented by labeled vertices in a graph $\mathcal{G}$ where occurrences of variables are represented by edges, e.g., a forest of parse trees (uses of hypergraphs or graph gadgets are allowed and do not affect our discussion). The graph is constructed in such a way that the group H_{sym} of symmetries of the original problem is isomorphic to the graph's group of automorphisms $Aut(\mathcal{G})$, which can be found by advanced software tools such as NAUTY [20] or SAUCY [8]. $Aut(\mathcal{G})$ is captured in a compact form by an unordered list of group generators, which cannot be much larger than $|\mathcal{G}|$. Having a group isomorphism, rather than an arbitrary one-to-one mapping between two sets, ensures that sets of generators of $Aut(\mathcal{G})$ map onto those of H_{sym}. Also, NAUTY and SAUCY would not have been applicable if symmetries did not form groups.

Using symmetries in search. For a given symmetry, a *symmetry-breaking predicate* (SBP) is a set of additional constraints added to the original constraints to prevent redundant search through symmetric branches. For multiple symmetries, compatibility is ensured by lex-leader SBP constructions [7] which have recently been improved [2]. It is often best to generate SBPs for group generators only [1], thus adding very few constraints (breaking fewer symmetries does not affect correctness). In practice, the SBPs for symmetries π_1 and π_2 make the addition of an SBP for products of π_1 and π_2 (and products of their powers) essentially unnecessary [2]. Group properties are essential in other paradigms for using symmetries, where efficiency calls for algorithms based on stabilizer chains [14].

Functional symmetries & don't cares. A Boolean function f with n input bits can be captured by a truth table with 2^n lines and one output column filled with 0s and 1s. Permutational and negational symmetries of f can be captured as symmetries of its truth table, or as automorphisms of the n-dimensional Boolean cube whose vertices represent the lines of the truth table and are labeled 0 or 1 (see further optimizations in [6]). Such symmetries can be generalized to non-Boolean domains and in that form subsume Freuder's notions of interchangeability and substitutability [10], allowing simultaneous variable permutations and value substitutions even when a particular representation of f is not symmetric. Application-derived functions are often partially defined, i.e., some of the output values can be resolved later. Such *output don't-care conditions* can be captured by asterisks in the truth table or by yet-undefined vertex labels in the Boolean cube.

Disjunctive constraints [11] can be modeled by graphs with removable edges, also suggesting that we study almost-symmetries of graphs first.

3 Possible Notions of Almost-symmetries

Semantic symmetries include syntactic ones and form the same algebraic structure. However, using all semantic symmetries is often computationally impractical. The same can be expected for almost-symmetries.

In order to relax the notion of symmetry, one considers transformations that do not necessarily preserve the structure in question. One known possibility is that of *conditional symmetries* that arise in the course of constraint-solving algorithms based on backtracking [13; 24]. Such algorithms work by assigning trial values to variables, which simplifies the original problem instance and potentially facilitates new symmetries. Given that conditional symmetries may exist in an exponential number of different contexts, it may be difficult to batch symmetry-detection and symmetry-breaking with as much efficiency as for unconditional symmetries. A typical conditional symmetry is applicable in a considerably smaller *scope* than a regular symmetry and therefore holds less promise to accelerate constraint-solving.[1] Yet, much of what we propose in this paper works for conditional symmetries as well.

Almost-symmetries are commonly defined as symmetries of slightly modified objects [15]. For example, adding or removing one constraint can make the overall set of constraints more symmetric. This may seem like a generalization of conditional symmetries, whose conditions can be viewed as constraints of a special kind. However, conditional symmetries rely on the *interpretation* of their conditions, i.e., assigned values are used to simplify constraints, whereas almost-symmetries are often formulated entirely in terms of *syntactic* manipulations on the set of constraints and promise greater computational efficiency. We note that such differences are only meaningful in the context of specific search procedures and can be ignored at first. Therefore we focus on labeled graphs, except in Section 7.

A popular example of an almost-symmetry of a graph is a symmetry of the graph derived by adding or removing a small number of edges [19]. However, we are first going to warm up by studying a different relaxation of graph symmetries that deals with vertex labels instead of edges, it is motivated by functions with don't-cares (see Section 2). From now on we are going to perceive vertex labels as colors — recall that a graph symmetry is a permutation of vertices that maps each vertex to a vertex of the same color and maps every pair of vertices connected by an edge to another such pair. The color-related limitation can be relaxed by introducing the *chameleon* color: a chameleon vertex can be mapped to a chameleon vertex or a vertex of any regular color. Additionally, a vertex of a regular color can be mapped to a chameleon vertex.[2]

[1]Symmetry-breaking for two symmetric Boolean variables reduces the solution space by 25%. However, when such a symmetry is conditional on fixed values of three other Boolean variables, the 25% reduction applies only to 1/8 of the solution space, thus the overall reduction is only by $3\frac{1}{3}$%.

[2]Our definition extends to several chameleon colors compatible with different sets of regular colors, and even a hierarchy of chameleon colors. However, most of the discussion that follows would still apply to such extensions.

4 Algebraic Structure in Almost-symmetries of Graphs

As long as the n vertices of the graph remain fixed, almost-symmetries are permutations from S_n. Thus

- products of almost-symmetries are unambiguous,
- the identity permutation is an almost-symmetry,
- each almost-symmetry has a unique inverse.

Unfortunately, almost-symmetries are not closed under compositional product. For example, consider a 3-vertex graph with no edges, the permutation π_1 that swaps the blue vertex v_1 with the chameleon vertex v_2, and π_2 that swaps v_2 with the red vertex v_3. The product $\pi_1 \cdot \pi_2$ maps the blue vertex v_1 to the red vertex v_3, which is forbidden. Worse, a power of an almost-symmetry may not be an almost-symmetry. We cannot fix all products, but we can fix the powers.

We further restrict almost-symmetries to those automorphisms of the underlying unlabeled graph $\mathcal{G}^*$ with the property that each cycle contains vertices of no more than one regular color.

This constraint is justified by considerations of group theory and symmetry-breaking. First, since regular symmetries are almost-symmetries and use the same compositional product, they form a *subgroup* of whatever structure almost-symmetries form. To this end, it is natural to seek additional subgroups. However, if powers of π are not almost-symmetries, then π cannot be in any subgroup of almost-symmetries. Second, given two almost-symmetries that have different contexts and do not form a product, their symmetry-breaking predicates can be extended by preconditions and conjoined (Section 7). However, this mechanism fails on powers of an almost-symmetry.

With their compositional product defined only partially, almost-symmetries do not form *groups*, *semigroups* or *monoids*. They do not form *cosets* because any coset containing the identity permutation $(\,)$ must be a subgroup. We can also try *groupoids* (also known as a *virtual groups*), which allow for partially-defined products but requires associativity. The latter property states that $\forall\, \alpha, \beta, \gamma$, if either $(\alpha \cdot \beta) \cdot \gamma$ or $\alpha \cdot (\beta \cdot \gamma)$ is defined, then both are defined and equal (we ignore other axioms of a groupoid for now). Given that the compositional product of almost-symmetries is inherited from S_n, there can be no problems with equality. However, it is possible that only one of the two products is defined. In the above example with vertices v_1, v_2, v_3 and permutations $\pi_1 = (12), \pi_2 = (23)$, the product $\pi_1 \cdot \pi_2$ was not a valid almost-symmetry. Now consider the products $(\pi_1 \cdot \pi_1) \cdot \pi_2$ and $\pi_1 \cdot (\pi_1 \cdot \pi_2)$. Since $\pi_1 \cdot \pi_1 = (\,)$, the former is defined ($= \pi_2$), but $(\pi_1 \cdot \pi_2)$ is not.

Before we reveal an algebraic structure compatible with these properties, we offer the following observation. For any given color-based almost-symmetry of a graph, there is *a specialization of colors* for the graph's chameleon vertices that turns the almost-symmetry into a regular symmetry. Indeed, each cycle can include vertices at most one regular color, to which all chameleon vertices in this cycle can be specialized. If *all* vertices in the cycle are chameleon-colored, specialize all of them to any existing color.

Now consider all possible color specializations of the chameleon vertices (which are just partitions of this vertex set into as many cells as we have regular colors). In each case we obtain a regular labeled graph with a group of symmetries, and each almost-symmetry is contained in at least one of those groups. To rephrase,

The set of almost-symmetries is a set-theoretic union of subgroups of S_n.

In an *irredundant* union-of-subgroups expression no subgroups can be skipped. Greedy removal of redundant subgroups from an expression ensures irredundancy, but not the smallest size. Even solving the (implicit) set-covering problem for given subgroups may not produce a union with fewest subgroups possible. [3]

Observe that all relevant subgroups are contained in the automorphism group $Aut(\mathcal{G}^*)$ of the unlabeled (colorless) graph $\mathcal{G}^*$ and contain all automorphisms of $\mathcal{G}^*$ whose cycles do not mix chameleon vertices with regular vertices. Such permutations form a sub-group that can be recovered in two steps: (1) construct the labeled graph $\mathcal{G}^\sharp$ by specializing all chameleon vertices to a color that has not been used before, (2) finding $Aut(\mathcal{G}^\sharp)$. The algebraic structure described above is visualized in Figure 1. In practice the intersection of subgroups can be larger than $Aut(\mathcal{G}^\sharp)$, e.g., consider two disconnected vertices v_1 (blue) and v_2 (chameleon), for which $Aut(\mathcal{G}^\sharp) = \{(\,)\}$ but the only non-trivial almost-symmetry (12) generates a larger subgroup.

Our analysis suggests that solutions to the graph almost-automorphism problem can be represented by subgroup generators arranged in potentially-overlapping unordered lists — one lists per subgroup of S_n in the union-of-subgroups structure.

Edge-based almost-symmetries discussed earlier also form a union of subgroups of S_n, taken over the various modifications of the original graph, and all powers of such almost-symmetries remain almost-symmetries. Furthermore, edge-based and color-based relaxations of graph symmetries can be combined, and the resulting almost-symmetries still form a set-theoretic union of subgroups of S_n.

[3]Consider $\mathbb{Z}_2 \times \mathbb{Z}_2$, the symmetry group of the letter H represented by the union of its three two-element subgroups.

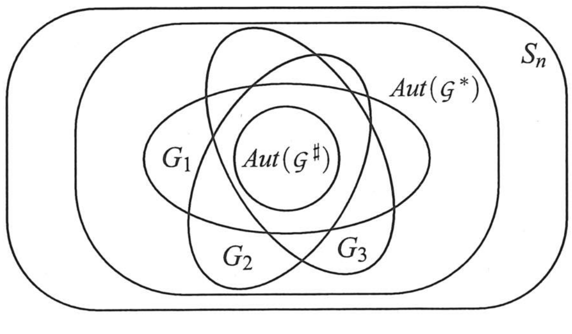

Figure 1: Algebraic structure in almost-symmetries — a set-union of subgroups: $Aut(\mathcal{G}^\sharp) \subseteq G_1 \cup G_2 \cup G_3 \subseteq Aut(\mathcal{G}^*)$. All shapes represent groups or subgroups, and illustrate containment relations geometrically.

5 Finding Almost-Symmetries

To find almost-symmetries and represent them compactly by lists of lists of permutations, we extend a common graph-automorphism algorithm, used in solvers NAUTY [20] and SAUCY [8], to handle color-based almost-symmetries. Since color specializations are partitions of the set of chameleon vertices, we are looking to capture all almost-symmetries by a small set of such partitions. Fortunately, a great deal of partition refinement is already performed the above graph-automorphism algorithm, which we describe next.

First, vertices of a given graph are differentiated by degree, creating an initial partition to separate vertices that cannot possibly be symmetric. The *cells* of this partition will be gradually split further. In particular, *immediate refinement* is based on external adjacencies, e.g., two vertices of same degree cannot be symmetric if one is adjacent to a vertex of degree m and the other is not adjacent to any vertex of degree m [8, Figure 1]. Such refinements can be exhausted by an efficient procedure due to Hopcroft. After that the algorithm resorts to (*traditional*) branching by, conceptually, picking a non-singleton cell and mapping its lowest-indexed vertex v_i to another vertex v_j (for more details, see [20; 8]). This may trigger another round of immediate partition refinement — neighbors of v_i can only map to neighbors of v_j. The overall algorithm proceeds by alternating between branching and refinement until all vertices in some cells are mapped to other vertices (or themselves), which allows one to test the resulting permutation for being a symmetry of the original graph [8, Figure 2]. Confirmed symmetries are accumulated,[4] and the algorithm backtracks to explore other branches of the search tree. With appropriate pruning [20], the algorithm ignores all branches leading only to symmetries expressible as compositions of accumulated symmetries. This ensures that group generators at the output are irredundant and can implicitly express an exponential number of symmetries in polynomial space.

Our extended algorithm leverages existing partition-refinement techniques and interleaves traditional branching with branching on colors of chameleon vertices. Since each branching on a chameleon vertex may create different subgroups in the union-of-subgroups structure, we seek to delay such branching and decrease the branching factor. The algorithm repeatedly applies rules from the following prioritized list.

1. Since all almost-symmetries of a given graph $\mathcal{G}$ are in $Aut(\mathcal{G}^*)$, apply an existing graph-automorphism algorithm to $\mathcal{G}^*$ (i.e., ignore vertex colors) until it needs branching or terminates.

2. Any cell with vertices of more than one regular color, but no chameleon vertices, must be split immediately, possibly triggering further partition refinement. Cells containing chameleon vertices cannot be split based on internal vertex colors, but can be refined based on external adjacencies.

[4]For graphs in engineering applications it is relatively rare to reject potential symmetries at this stage, but such *bad leaves* are common for highly symmetric Cayley graphs. With no bad leaves, the algorithm runs in polynomial time.

3A. If a cell contains only chameleon vertices, then specialize all vertices to one arbitrary color.

3B. If a cell contains chameleon vertices and vertices of one regular color, then specialize all chameleon vertices to this regular color.

4. In a non-trivial cell without chameleon vertices, invoke traditional branching in the hope that some cells with chameleon vertices will be refined through adjacencies. This rule does not use traditional branching in cells with unspecified chameleon vertices due to difficulties with color assignment, implied in some, but not all branches.

5. Branch on chameleon vertices in cell j with the smallest branching factor as follows:

- Specialize all k_j chameleon vertices in cell j at once — otherwise splitting and traditional branching will not work (Rules 2 and 4).
- Assign only c_j regular colors used in cell j.
- Select j to minimize branching factor $c_j^{k_j}$.

Symmetry generators accumulated since the last branching on chameleon vertices can only be used in that branch, hence we output a new subgroup upon returning from the lowest-level chameleon branch.

Rule 2 (immediate refinement) and Rules 3A, 3B (dominant colors) perform constraint propagation, interleaved with two types of branching (Rules 4 and 5). Pruning by accumulated symmetry ensures that generators of each subgroup are irredundant. For many graphs, constraint propagation alone will specialize all chameleon vertices, and for most randomly-generated graphs no branching will be invoked at all. However, even when almost-symmetries form a group, chameleon branching may be necessary, as shown in Figure 2. This example also shows that our algorithm may produce redundant unions-of-subgroups and therefore needs post-processing. However, in general, delayed branching on chameleon vertices and the minimization of branching factors lead to more compact union-of-subgroup expressions. The core algorithm above allows a number of engineering improvements, e.g., its decoupled branching on chameleon vertices can honor color-based constraints and preferences.

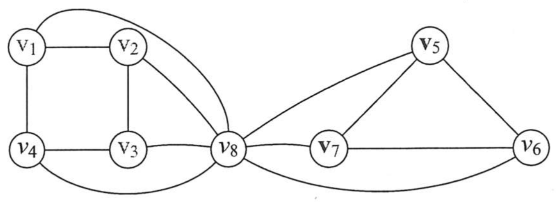

Figure 2: A graph with three red vertices (v_1, v_2, v_3), two blue vertices (v_5, v_7) and three chameleon vertices (v_4, v_8, v_6). Rule 1 separates v_8 from other vertices, and Rule 3A colors v_8 red. The remaining cell cannot be split (Rule 2), and traditional branching is not allowed in it (Rule 4). Therefore, Rule 5 must be applied with branching factor 4 — on two vertices (v_4,v_6) with two colors. Three of those branches produce non-trivial almost-symmetries after traditional branching (Rule 4), but one of the resulting subgroups contains two others. Indeed, every almost-symmetry becomes a symmetry when v_4 is colored red and v_6 is colored blue.

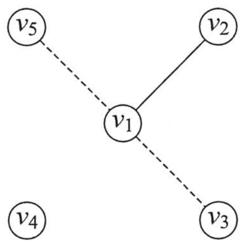

Figure 3: A graph with one regular and two chameleon edges illustrating difficulties in ensuring that powers of almost-symmetries are, too, almost-symmetries. Consider a clockwise 45°-rotation about the center.

6 Generalizations and Applications

Here we discuss other potential notions of almost-symmetry in graphs, with an eye on algebraic structure.

To bridge the gap between color-based and edge-based almost-symmetries in graphs one can introduce *chameleon edges* whose end-vertices may be mapped into other pairs of vertices that are either connected by edges or not. In other words, we classify all pairs of vertices into regular edges, chameleon edges, and non-edges, so as to prohibit mapping edges to non-edges and vice versa. Just as in the case of color-based almost-symmetries, ensuring that all powers of an almost-symmetry are, too, almost-symmetries is tricky. For example, consider the five-vertex graph in Figure 3 with one regular edge and two chameleon edges. The 4-cycle (2345) satisfies the constraints imposed so far, but its square $(24)(35)$ swaps the regular edge (v_1,v_2) with the non-edge (v_1,v_4), which is prohibited. To ensure that powers of almost-symmetries do not map edges to/from non-edges, one can impose an additional restriction similar to that in Section 4.

For a given vertex-permutation $\pi \in S_n$, define its action on pairs of vertices by means of $\pi_e(v_i,v_j) := (\pi(v_i),\pi(v_j))$ so that $\pi_e \in S_{\binom{n}{2}}$. We prohibit each permutation π whose π_e has at least one cycle containing both *an edge and a non-edge.*

According to this restriction, a chameleon edge in a cycle without edges (in a particular almost-symmetry) can be resolved into a non-edge, and in a cycle with edges — into an edge. This subsumes the popular view of graph almost-symmetries as symmetries of modified graphs [19]: if a given edge can be modified, it should be viewed as a chameleon edge. One can additionally *constrain* the number of instantiated edges or handle chameleon edges with *preference* for non-edges/edges. Such almost-symmetries, too, form unions of subgroups.

Almost-symmetries of functions with don't-cares are isomorphic to almost-symmetries of labeled graphs, as explained in Section 2. We now discuss disjunctive constraints using the DNF-SAT instance $ac+bc$ as example. This formula has one non-trivial permutational symmetry (ab), with SBP $(a \leq b) = a' + b$. We can also swap *either* (a with c) *or* (b with c), but not both. The obvious almost-symmetry-breaking predicate $(a \leq c)+(b \leq c) = (a'+c)+(b'+c) = a'+b'+c$ removes the non-solution 110 allowed by $a'+b$.

7 Almost-symmetry-breaking

Existing approaches to exploiting symmetries in search include pre-processing with symmetry-breaking predicates (SBPs), symmetry-breaking during search [13; 14] and symmetry-breaking by dominance. The first approach is largely independent of the search algorithm, while the last two require implementation changes and typically have greater overhead [4]. Since in this work we only seek to demonstrate a complete and sufficiently general "life-cycle" for almost-symmetries, we focus on pre-processing from now on. We do hope that the discussion below will also be useful in future work targeting specific search algorithms.

As mentioned in Section 2, the group isomorphism $H_{sym} \simeq Aut(\mathcal{G})$ is important when computing symmetries of non-graph objects — it allows one to pull back group generators that are critical to efficient symmetry-breaking [1]. We now define isomorphism of almost-symmetries so that descriptions of almost-symmetries in terms of generators always map to valid descriptions.

An isomorphism of almost-symmetries is a one-to-one mapping γ such that $\forall$ almost-symmetries π_1,π_2, their product $\pi_1 \cdot \pi_2$ is defined if and only if the product $\gamma(\pi_1)\cdot\gamma(\pi_2)$ is defined, in which case we require that $\gamma(\pi_1\cdot\pi_2) = \gamma(\pi_1)\cdot\gamma(\pi_2)$.

Isomorphism-of-symmetries proofs for graph constructions (that model constraints and objective functions) [1; 3] all extend to almost-symmetries. Omitting details, each such graph construction defines an isomorphism of containing S_k groups (for k initial variables and k graph vertices), and this mapping remains an isomorphism on every subgroup in the union. Hence, if either of the two permutations $\gamma(\pi_1)\cdot\gamma(\pi_2)$ and $\pi_1\cdot\pi_2$ is in a valid subgroup, then so is the other.

When defining SBPs, the key issue is not to prohibit *all* good solutions. To aid in this, we build global SBPs from known lex-leader SBPs [2] for every generator of almost-symmetries. In particular, generator SBPs can be conjoined within subgroups because the respective almost-symmetries can be freely composed. When the union of subgroups is derived from a disjunctive constraint, we can OR *all* subgroup SBPs because the lex-smallest assignment satisfying a disjunctive term will satisfy one of subgroup SBPs. Yet, for a general CSP, a unique overall solution may violate all subgroup SBPs, which suggests additional conditions per subgroup G_i. We propose to pick lex-leaders only among those assignments that enable almost-symmetries in G_i.

For almost-symmetries $g_{i1}, g_{i2}, \ldots, g_{im}$ from subgroup G_i, we build their SBP as $\Psi(G_i) := (\Phi_{G_i} \Rightarrow (\wedge_j \psi(g_{ij}))$ where $\psi(g_{1j})$ is a lex-leader SBP for the permutation g_{ij} and Φ_{G_i} is a pre-condition. *The overall SBP for $\cup_i G_i$ then $\wedge_i \Psi(G_i)$.*

For function f, Φ_{G_i} is (ideally the weakest) specialization of don't-cares that turns $g_{ij}, \forall j$ into symmetries. For a constraint graph, if turning g_{ij} into symmetries requires adding (or removing) an edge, then Φ_{G_i} expresses the new constraint represented by this edge (or its negation). When a good precondition is hard to build, we can skip G_i by assuming $\Phi_{G_i} = 0, \Psi(G_i) = 1$.

8 Conclusions and Ongoing Work

In this paper we have studied almost-symmetries of several kinds, demonstrating that they possess the structure of a set-theoretic union of subgroups of S_n. We explained how almost-symmetries can be compactly represented, computed for a labeled graph and used for symmetry-breaking.

Open questions span abstract algebra, computational complexity, algorithm design as well as applications in search and optimization. While groups (of symmetries) admit a complete abstract characterization and have many known properties, such a theory is not yet available for the union-of-subgroups structure (can any union-of-subgroups appear as a set of almost-symmetries?). Regarding worst-case complexity, recall that graph-automorphism is in NP, but is unlikely to be NP-complete. It cannot currently be solved in polynomial time, except in the bounded-degree case [5]. While all graph automorphisms can be captured in poly-space by group generators, almost-symmetries may require exponential space even in the bounded-degree case — an example is given in the Appendix. Yet, we believe that application-derived and "average-case" graphs can be solved quickly, as is the case for graph automorphism [8]. Extending symmetry-breaking during search [13; 14; 4] and by dominance to almost-symmetries is another interesting direction.

Our discussion of algebraic structure in almost-symmetries of graphs also applies to functions with don't-cares and to disjunctive constraints. Other types of almost-symmetries may carry the union-of-subgroups structure with union taken over the allowed variants of the underlying object. The subgroups then contain regular symmetries of the variant objects. A computational challenge is to minimize the number of subgroup terms in the union. Almost-symmetries can be captured using group generators — instead of the usual list of generators all of which are compatible, one needs a list of lists of generators, which are guaranteed compatible only within a given list. This leads to a generalization of existing symmetry-breaking predicates by means of preconditions. Unlike well-known conditional symmetries, almost-symmetries do not depend on the variable ordering and may be entirely syntactic, which makes them potentially more numerous and easier to find. On the other hand, conditional symmetries also form unions of subgroups and can be viewed as a type of semantic almost-symmetries.

Appendix: Exp-sized Union of Subgroups

Consider $n+2$ disconnected vertices: v_1 is red, v_2 is blue, and n vertices are chameleon. All color specializations are indexed by $j = 0..2^n - 1$ such that 1s in the binary expansion of j correspond to red vertices. With $\#j$ red vertices and $n - \#j$ blue vertices, $G_j \simeq S_{\#j} \times S_{n-\#j}$. No element of G_j for any j maps $v_1 \mapsto v_2$, hence this is an invariant of $\cup_j G_j$. However, $\forall\ i \neq j$, the minimal subgroup containing G_i and G_j contains a permutation that maps $v_1 \mapsto v_2$. Hence in any union-of-subgroups expression for almost-symmetries of the graph in question, each G_j must be contained in a separate term. Therefore, any list of lists of generators representing such an expression requires $\Omega(2^n)$ space.

References

[1] F. A. Aloul, A. Ramani, I. L. Markov, and K. A. Sakallah, "Solving Difficult Instances of Boolean Satisfiability in the Presence of Symmetry", *IEEE Trans. on CAD*, Sep. 2003, pp. 1117-1137.

[2] F. A. Aloul, I. L. Markov, and K. A. Sakallah, "Efficient SymmetryBreaking for Boolean Satisfiability," in Proc. *IJCAI* '03, pp. 271-282.

[3] F. A. Aloul, A. Ramani, I. L. Markov and K. A. Sakallah, "Symmetry-Breaking for Pseudo-Boolean Formulas", in Proc. *ASPDAC* '04, pp. 884-887.

[4] F. A. Aloul, A. Ramani, I. L. Markov and K. A. Sakallah, "Dynamic Symmetry-Breaking for Improved Boolean Optimization", *ASPDAC* '05, pp. 445-450.

[5] L. Babai and E. M. Luks, "Canonical Labeling of Graphs", in Proc. *STOC'83*, pp. 171-183.

[6] K.-H. Chang, I. L. Markov and V. Bertacco, "Post-Placement Rewiring and Rebuffering by Exhaustive Search For Functional Symmetries", Proc. *ICCAD*'05.

[7] J. Crawford, M. Ginsberg, E. Luks and A. Roy, "Symmetry-breaking Predicates For Search Problems", *Int'l Conf. Principles of Knowledge Represent. & Reasoning* (KR) '96, pp. 148-159.

[8] P. T. Darga, M. H. Liffiton, K. A. Sakallah, and I. L. Markov, "Exploiting Structure in Symmetry Detection for CNF", in Proc. *DAC* '04, pp. 530-534. `http://vlsicad.eecs.umich.edu/BK/SAUCY/`

[9] J. D. Dunitz, "Symmetry Arguments in Chemistry", *Proc. Nat'l Acad. Sci.* USA 1996; 93: 14260-14266.

[10] E. C. Freuder, "Eliminating Interchangeable Values in Constraint Satisfaction Problems", in Proc. *AAAI* '91, pp. 227-233.

[11] E. C. Freuder and P. D. Hubbe, "Using Inferred Disjunctive Constraints to Decompose Constraint Satisfaction Problems", in Proc. *IJCAI* '93, pp. 254-260.

[12] W. Fushchych, W. Shtelen, "On Approximate Symmetry and Approximate Solutions of Nonlinear Wave Equations With Small Parameters", *J. Physics A*, 1989, vol. 22, L887-L890.

[13] I. P Gent, B. M. Smith, "Symmetry Breaking in Constraint Programming", in Proc. *ECAI*'00, pp. 599-603.

[14] I. Gent, W. Harvey, T. Kelsey, "Groups and Constraints: Symmetry breaking during search", in Proc. *CP* '02, LNCS 2470, pp. 415-430, Springer.

[15] P. Gregory and A. Donaldson, "Concrete Applications of Almost-Symmetry", in *Workshop on Almost-Symmetry in Search*, pp. 1-5, Comp. Sci. TR-2005-201, Univ. of Glasgow, 2005.

[16] D. J. Gross, "The Role of Symmetry in Fundamental Physics", *Proc. Nat'l Acad. Sci.* USA 1996; 93: 14256-14259.

[17] M. E. Kellman, "Symmetry in Chemistry: From the Hydrogen Atom to Proteins", *Proc. Nat'l Acad. Sci.* USA 1996; 93: 14287-14294.

[18] B. Krishnamurthy, "Short Proofs For Tricky Formulas", *Acta Informatica*, vol. 22, pp.327–337, 1985.

[19] D. Long and M. Fox, "Restoring Symmetries in Almost-Symmetric Graph Structures", in *Workshop on Almost-Symmetry in Search*, pp. 6-13, Comp. Sci. TR-2005-201, Univ. of Glasgow, 2005.

[20] B. D. McKay, "Practical Graph Isomorphism", *Congressus Numerantium* 30('81), pp. 45-87.

[21] A. Mishchenko, "Fast Computation of Symmetries in Boolean Functions", *IEEE Trans. on CAD*, Nov. 2003, pp. 1588-1593.

[22] P. J. Steinhardt "New Perspectives on Forbidden Symmetries, Quasicrystals, and Penrose Tilings", *Proc. Nat'l Acad. Sci.* USA 1996; 93: 14267-14270.

[23] V. Trimble, "Astrophysical symmetries", *Proc. Nat'l Acad. Sci.* USA 1996; 93: 14221-14224.

[24] Y. Zhang and E. Freuder, "Conditional Interchangeability and Substitutability", in Proc. *SymCon* '04.

Comparison of Symmetry Breaking Methods in Constraint Programming

Karen E. Petrie and Barbara M. Smith
Cork Constraint Computation Center
University College Cork
Cork, Ireland
k.petrie@4c.ucc.ie,b.smith@4c.ucc.ie

Abstract

Symmetry in a Constraint Satisfaction Problem can cause wasted search, which can be avoided by adding constraints to the CSP to exclude symmetric assignments or by modifying the search algorithm so that search never visits assignments symmetric to those already considered. One such approach is SBDS (Symmetry Breaking During Search); a modification is GAP-SBDS, which works with the symmetry group rather than individual symmetries. There has been little experience of how these techniques compare in practice. We compare their performance in finding all *graceful labellings* of graphs with symmetry. For these problems, GAP-SBDS is faster than SBDS unless there are few symmetries. When simple symmetry-breaking constraints can be found to break all the symmetry, GAP-SBDS is slower; if the constraints break only part of the symmetry, GAP-SBDS does less search and is faster. Eliminating symmetry has allowed us to find all graceful labellings, or prove that there are none, for several graphs whose gracefulness was not previously known.

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. Symmetries may be found between variables or values or variable/value combinations. An example of value symmetry can be found in graph colouring problems where the different colours are interchangeable.

Symmetries can give rise to redundant search, since subtrees may be explored which are symmetric to subtrees already explored. This difficulty has been seen in the practical solving of real world constraint problems [Henz, 2001]. To avoid this redundant search, constraint programmers have designed methods, which try to exclude all but one in each equivalence class of solutions.

A common way to reduce or eliminate symmetry is to add constraints to the CSP, to exclude some or all symmetric equivalents. Ideally, the new constraints should be satisfied by only one assignment in any symmetry equivalence class. It is often hard to find simple symmetry constraints. Crawford *et al.* [Crawford *et al.*, 1996] give a systematic method for generating symmetry breaking constraints, when the problem only contains variable symmetry. This method encompasses writing down a solution to the CSP and then calculating what effect each of the problem symmetries has on this solution. Lexicographic ordering constraints are then imposed, to ensure that the original solution is smaller than any of its symmetric equivalents. There is no systematic procedure to calculate symmetry breaking constraints when the symmetry does not just affect the variables, but sometimes such constraints can be identified. Symmetry breaking constraints can interact badly with the search strategy. It may be that amongst the solutions in a symmetry equivalence class that do not satisfy the new constraints is one that would be found earlier, given the search strategy being used, than any of the solutions that can still be found. Hence, the constraints can still allow wasted search, and if only one solution is required, adding symmetry-breaking constraints can be worse than doing nothing.

An alternative is to adapt the search algorithm so that constraints are added during search to prevent exploration of assignments. Symmetry Breaking During Search (SBDS), as can be seen in previous chapters, adds such constraints on backtracking. SBDS requires a function for each symmetry in the problem describing its effect on the assignment of a value to a variable. 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 such situations Gent *et al.* [Gent *et al.*, 2002] linked SBDS (in ECL^{i}PS^e) with GAP (Groups, Algorithms and Programming) [GAP, 2000], a system for computational discrete algebra and in particular computational group theory. GAP-SBDS allows the symmetry group rather than its individual elements, to be described.

GAP-SBDS allows the symmetry to be handled more efficiently than in SBDS; the elements of the group are not explicitly created, as is required in the original SBDS. On the other hand, GAP-SBDS has the overhead of the communication between ECL^{i}PS^e and GAP. Furthermore, the symmetry-breaking constraints posted on backtracking are constructed dynamically from the stabiliser rather than being pre-defined in the symmetry functions as in SBDS. It can be expected that GAP-SBDS will be a better choice than SBDS when the symmetry group is large, if only because it becomes impractical to list explicitly the individual elements of the group. However, for small symmetry groups, SBDS can be faster.

In [Gent *et al.*, 2002], limited experiments with GAP-SBDS are reported; it is much faster than the original SBDS on the Alien Tiles problem [Gent *et al.*, 2000], where the symmetry group has 1152 elements. Experiments with two Balanced Incomplete Block Design problems (BIBDs) showed that it can successfully handle symmetry groups with millions of elements, which SBDS clearly cannot do. However, symmetry constraints gave better results than GAP-SBDS on the two problems considered, except for variable orderings incompatible with the constraints.

More experience of these techniques is required in order to identify which is appropriate for a given situation. If simple constraints can be found that will break some of the symmetry in a CSP, the CP users needs to know whether they should be used, or whether one of the techniques that break symmetry during search should be applied. In this paper, symmetry breaking is investigated in a class of graph labelling problems. SBDS is compared with GAP-SBDS; and GAP-SBDS with constraints to break the symmetry.

As well as providing further experience of these techniques in practice, constraint programming has proved to be a valuable technique for investigating these problems. Eliminating symmetry has allowed many new results to be found, which are presented throughout this chapter.

2 Graceful Graphs

A labelling f of the nodes of a graph with q edges is *graceful* if f assigns each node a unique label from $\{0, 1, ..., q\}$ 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, ..., q.) Figure 1 shows an example. The study of graceful graphs is an active area of graph theory. Gallian [Gallian, 2004] surveys graceful graphs, i.e. graphs which have a graceful labelling, and lists the graphs whose status is known. [Gallian, 2004] is the latest version of a dynamic

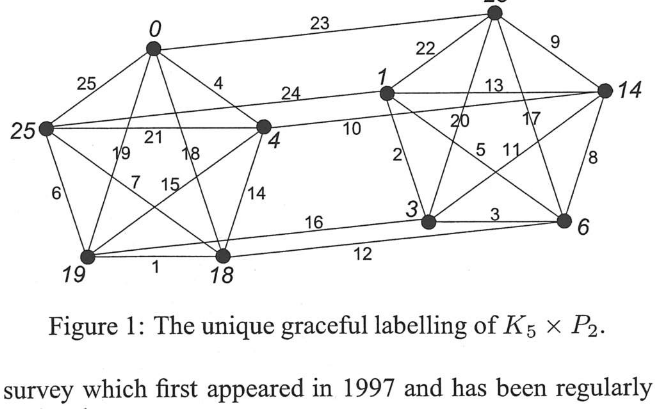

Figure 1: The unique graceful labelling of $K_5 \times P_2$.

survey which first appeared in 1997 and has been regularly updated.

Finding a graceful labelling of a given graph, or proving that one does not exist, can be expressed as a CSP. Specific cases have previously been considered; for instance, the all-interval series problem (problem 007 in CSPLib, at http://www.csplib.org) is equivalent to finding a graceful labelling of a path, and Lustig & Puget [I.J.Lustig and Puget, 2001] found a graceful labelling of a graph discussed in section 4.

There are two kinds of symmetry in the problem of finding a graceful labelling of a graph: first, there may be symmetry in the graph. For instance, if the graph is a clique, any permutation of the vertex labels in a graceful labelling is also graceful. If the graph is a path, P_n, the labels $x_1, x_2, ..., x_n$ can be reversed to give an equivalent labelling $x_n, ..., x_2, x_1$. We might call this type of symmetry geometric. The second type of symmetry is that we can replace every vertex label x_i by its complement $n - x_i$. We can also of course combine each geometric symmetry with the complement symmetry.

The advantage of this class of problem is that graphs can be picked with interesting symmetries. In general this is a necessary property in the choice of problems for symmetry evaluation. Hence, different symmetry-breaking approaches applied to different kinds of symmetry can be compared. Moreover, a particular type of symmetry, e.g. vertex permutations, can be chosen, and the number of symmetries varied, e.g. the number of vertices. This allows comparison of SBDS and GAP-SBDS on a range of problems, starting with ones that SBDS can easily handle and going on to ones that it cannot.

2.1 Modelling Decisions

A basic CSP model has a variable for each node $x_1, x_2, ..., x_n$, each with domain $0, 1, ..., q$ and a variable for each edge $d_1, d_2, ..., d_q$, each with domain $1, 2, ..., q$. The constraints of the problem are: if edge k joins nodes i and j then $d_k = |x_i - x_j|$; $x_1, x_2, ..., x_n$ are all different; and $d_1, d_2, ..., d_q$ are all different. The node variables are used as the search variables as they give the best results both when and when not considering symmetry breaking. They are also the simplest set to consider symmetry breaking over. When undertaking a comparative study, it is important to find a model that is efficient enough to solve the problem in reasonable time; but simple enough not to mask any empirical differences between symmetry breaking methods. The model must also hold for all instances of the problem. All modelling decisions were made after experimental testing, using SBDS to find all graceful labellings of $K_4 \times P_2$.

All Different

ECL^{i}PS^e provides two different levels of propagation for the *all_different* constraint, it can either be treated as a set of binary $\neq$ constraints, or a *global all_different* with higher propagation. Results of testing both constraints over the edge variables and node variables are shown in table 1.

In accordance with these results we use the *global all_different* on the edge variables and the $\neq$ 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 far more possible values than variables.

Version of *all_different* constraint	bt	sec.
$\neq$ *all_diff* on nodes & $\neq$ *all_diff* on edges	3188	91.94
Global *all_diff* on nodes & $\neq$ *all_diff* on edges	3184	91.56
$\neq$ *all_diff* on nodes & Global *all_diff* on edges	147	12.53
Global *all_diff* on nodes & Global *all_diff* on edges	147	12.41

Table 1: Comparison of *all_different* constraints finding all graceful labellings of $K_4 \times P_2$.

Variable	$K_4 \times P_2$		$K_5 \times P_2$	
Ordering Heuristic	bt	sec.	bt	sec.
lex	147	12.53	4172	1325.69
SDF	175	12.54	5781	1531.52

Table 2: Comparison of different variable ordering heuristics using SBDS for finding all graceful labellings.

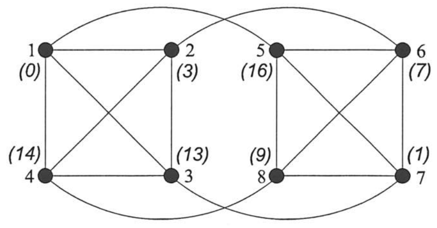

Figure 2: The graph $K_4 \times P_2$.

3 Search Decisions

Once the model is fully developed, it can be considered which search heuristic should be used. The remit for this is similar to that of choosing the model. We need a search heuristic that leads to efficient testing of the symmetry breaking methods, whilst not masking the different performances. In [I.J.Lustig and Puget, 2001] Lustig and Puget found smallest domain first (SDF) variable ordering to be far superior, so we compared this to a static variable ordering over both $K_4 \times P_2$ (the variable ordering used for this graph is shown in Figure 2) and $K_5 \times P_2$. The results can be found in table 2.

These results show the static ordering to be using less search than smallest domain first when using SBDS. In general, static variable ordering heuristics are a good choice when comparing symmetry breaking methods as symmetry breaking constraints can often conflict with dynamic heuristics. In fact, symmetry breaking constraints can conflict with static variable orderings as well, but in that case it is usually possible to find a reasonable heuristic by empirical testing within the study. Static search orders also provide a better test bed for dynamic search methods, as any differences in the search tree are due to earlier pruning due to the symmetry breaking method, and not because the dynamic search method causes a different search path to be taken.

4 $K_m \times P_2$ Graphs: SBDS *v.* GAP-SBDS

The graph shown in Figure 2, with the node numbering used in the CSP and one of its graceful labellings, is the cross-product of the clique K_4 and the path P_2: it consists of two copies of K_4, with corresponding vertices in the two cliques also forming the vertices of a path P_2. Lustig & Puget [I.J.Lustig and Puget, 2001] found a graceful labelling of this graph; it was not previously known to be graceful. However, they looked for only one graceful labelling of the graph and did not break any of the symmetry.

The symmetries of the graph are, first, intra-clique permutations: any permutation of the 4-cliques which acts on both in the same way. For instance, we can transpose nodes 1 and 2 and simultaneously 5 and 6. Second, inter-clique permutations: the labels of the first clique (nodes 1, 2, 3, 4) can be interchanged with the labels of the corresponding nodes (5, 6, 7, 8) in the second. These can also be combined with each other and with the complement symmetry. Hence, the size of the symmetry group is $4! \times 2 \times 2$, or 96.

GAP-SBDS requires the symmetry group of the problem as input. In SBDS, a function is required for every symmetry other than the identity: i.e. 95 functions for $K_4 \times P_2$. GAP is used to output the required functions, using the same generators as for GAP-SBDS. As described in [Gent *et al.*, 2000], GAP was used in a similar fashion to produce the symmetry functions for the Alien Tiles problem.

Some of the symmetry in the $K_4 \times P_2$ problem can alternatively be eliminated using constraints. Several different strategies have been devised for eliminating or reducing the symmetry, in order to compare the original SBDS and GAP-SBDS.

A: All the symmetry can be eliminated (i.e. the full symmetry group of 96 elements) using SBDS or GAP-SBDS.

B: Alternatively, it might be decided not to eliminate the complement symmetries; at worst this will double the number of solutions. This leaves just the graph symmetry group, i.e. 48 symmetries.

C: Once the complement symmetry has been ignored, the inter-clique symmetry can be eliminated by adding constraints to the CSP, provided that they do not interfere with permuting the node labels within the cliques. A permissible constraint is that the smallest node label in the first clique is less than the smallest in the second clique. Since there is a node with value 0, this means that it must be in the first clique. With this constraint, SBDS or GAP-SBDS need only eliminate the 24 remaining symmetries.

D: the 24 remaining symmetries consist of all permutations of the subsets $\{1,5\}$, $\{2,6\}$, $\{3,7\}$ and $\{4,8\}$ of the variables. This is a generalisation of symmetry due to variables being indistinguishable, and in this special case, the symmetry can be eliminated by just the transpositions of the variables, or sets of variables, being permuted. Here, each transposition acts on two of the variable subsets, e.g. one swaps the labels of nodes 1 and 2, and of nodes 5 and 6. In SBDS, a function

Graph	Strategy + no. of syms.		SBDS BT	SBDS sec.	GAP-SBDS BT	GAP-SBDS sec.
$K_3 \times P_2$	A	24	6	0.25	9	0.54
	B	12	16	0.24	16	0.59
	C	6	16	0.21	16	0.58
	D	3	16	0.2	-	-
$K_4 \times P_2$	A	96	147	12.9	165	8.3
	B	48	369	13.1	369	14.3
	C	24	369	11.0	369	14.1
	D	6	369	10.6	-	-
$K_5 \times P_2$	A	480	4172	1356	4390	382
	B	240	9889	929	9889	793
	C	120	9889	659	9889	783
	D	10	9889	629	-	-

Table 3: Comparison of different levels of symmetry breaking using SBDS or GAP-SBDS for finding all graceful labellings of $K_n \times P_2$.

can be provided for each of the six transpositions. This cannot be done in GAP-SBDS, however, because the subset of transpositions is not closed under composition and so is not a subgroup of the full symmetry group. As in strategy C, the inter-clique symmetry is broken by a constraint.

These different ways of dealing with symmetry in $K_4 \times P_2$ using SBDS and GAP-SBDS are compared in Table 3. Results are also given for $K_3 \times P_2$, with 24 symmetries, and $K_5 \times P_2$, with 480. There are 4 non-isomorphic graceful labellings of $K_3 \times P_2$ and 15 of $K_4 \times P_2$. $K_5 \times P_2$ has a unique graceful labelling (shown in Figure 1). Strategies B, C and D only break the graph symmetries and so find twice as many solutions. If we do nothing to break symmetry, there are 1440 solutions for $K_4 \times P_2$. $K_3 \times P_2$ and $K_4 \times P_2$ were already known to be graceful, but the number of non-isomorphic graceful labellings, and the results for $K_5 \times P_2$, are new.

For each graph, when all but the complement symmetries are eliminated, i.e. for strategies B and C, SBDS and GAP-SBDS incur the same search effort, although the runtime differs considerably. In strategy A, SBDS does less search than GAP-SBDS; this seems to be due to the lazy evaluation of $g(A)$ in GAP-SBDS, to delay imposing constraints, which results sometimes in missing some constraint propagation. However, it is not clear why this results in a difference in search only when the complement symmetries are involved. The best strategy for SBDS is D, where the number of symmetry functions is smallest. Increasing the number of symmetry functions severely affects the running time of SBDS, to the extent that strategy A is much the slowest strategy for SBDS on the largest problem, even though the number of backtracks is more than halved. The running time of GAP-SBDS, on the other hand, is much less affected by the number of symmetries, and its best strategy is to break all the symmetry and hence benefit from the reduction in search.

For each problem, strategy C is faster for SBDS than for GAP-SBDS, showing that for a sufficiently small number of symmetries (120 for $K_5 \times P_2$) it is faster to have the individual symmetries expressed explicitly than as a group, and so avoid the overheads of interacting with GAP. However, it does not depend solely on the size of the symmetry group: for $K_4 \times P_2$, strategy A is faster for GAP-SBDS than for SBDS, although there are only 96 symmetries.

These symmetry-breaking strategies can be extended to the graph $K_4 \times P_3$. This has a third 4-clique whose nodes are joined pairwise to the second. The 704 non-isomorphic graceful labellings of this graph have been found. As with $K_5 \times P_2$, $K_4 \times P_3$ was not previously known to be graceful. GAP-SBDS, breaking all the symmetry, was compared with SBDS using strategy D, i.e. the best strategy for each method. The symmetries are exactly as in $K_4 \times P_2$, but instead of swapping the first and second clique, the first and third are swapped. In using strategy D (6 SBDS functions + a constraint), the constraint must now be that the smallest node label in the first clique is less than the smallest node label in the third, since the node labelled 0 may now be in the central clique. In the best cases for each strategy GAP-SBDS takes 386,068 backtracks and 21150 sec; SBDS takes 844,629 backtracks and 32700 sec.

This reinforces the conclusion that for these problems the best strategy for GAP-SBDS is faster than the best for SBDS, except for the small $K_3 \times P_2$ problem. It is better to break all the symmetry using GAP-SBDS than to break only half using SBDS, even though only a small number of symmetry functions is required using strategy D.

5 Symmetry-Breaking Constraints

Alternatively breaking the symmetry of $K_m \times P_2$ using constraints added to the CSP could be considered. For all except the combinations of the complement symmetry and the graph symmetries, this is straightforward. The systematic procedure given by Crawford *et al.* [Crawford *et al.*, 1996] for generating such constraints can be followed: write down a solution $(x_1, x_2, ..., x_n)$ to the CSP and the effect on this solution of every symmetry in the problem, and then impose constraints ensuring that the original solution is lexicographically smaller than any of its symmetric equivalents. In theory, this might lead to one constraint for every symmetry, but often the constraints can be simplified so that many symmetries are eliminated by the same constraint. For instance, in $K_4 \times P_2$, any symmetry which transposes the labels of nodes 1 and 2 (and hence also of nodes 5 and 6) is eliminated by the constraint $x_1 < x_2$. (The task of finding constraints is simplified by the fact that x_1 and x_2 cannot be equal.) This is explained more fully by Puget [Puget, 2004].

The constraints $x_1 < x_2$, $x_2 < x_3$, $x_3 < x_4$ exclude permutations within the cliques and $x_1 < x_5$, $x_1 < x_6$, $x_1 < x_7$, $x_1 < x_8$ exclude swapping the first clique with the second and permuting both. Since the constraints imply that $x_1 = 0$, we can add this and then only need $x_2 < x_3$, $x_3 < x_4$. Hence, three constraints eliminate half the symmetry, i.e. 48 elements of the symmetry group.

These constraints can be compared with those added during search by SBDS. When a symmetry function is had in SBDS for every graph symmetry (strategy B) the effect will be similar to adding a constraint to the model for every symmetry, and will result in many duplicated constraints added on backtracking. Hence, strategy B is roughly comparable to using Crawford *et al.*'s procedure without doing any simplification and amalgamation of the resulting constraints. Strategy D, with only 6 symmetry functions, is roughly comparable to the three simplified constraints which break all the graph symmetries. The constraints are slightly quicker than strategy D for $K_4 \times P_2$ and $K_5 \times P_2$, and take almost exactly the same number of backtracks. However, SBDS has an additional advantage in being independent of the variable ordering. The procedure for deriving the symmetry constraints assumes that the variables will be assigned in the order $x_1, x_2, ..., x_n$; if they are not, finding solutions may be delayed. On the other hand, SBDS will always find the first solution, with respect to the variable ordering, in any symmetry equivalence class.

In principle, all the symmetry could be eliminated by adding constraints to the CSP. However, the combinations of the complement symmetry and the graph symmetries require many more, and more complex, constraints than the graph symmetries. For instance, the solution shown in Figure 2, (0, 3, 13, 14, 16, 7, 1, 9), can be transformed to (16, 13, 3, 2, 0, 9, 15, 7) by taking the complement, then to (0, 9, 15, 7, 16, 13, 3, 2) by swapping the two cliques, and finally to (0, 7, 9, 15, 16, 2, 13, 3), by a permutation of the cliques. The final solution satisfies the constraints already added.

This symmetry transforms the general solution $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8)$ to $(q - x_5, q - x_8, q - x_7, q - x_6, q - x_1, q - x_4, q - x_2, q - x_3)$. Following [Crawford *et al.*, 1996], constraints could be add to ensure that the first solution is lexicographically less than the second. ECL^{i}PS^e allows this to be stated as a single constraint, which is equivalent to (and in other constraint programming systems would have to be expressed as) the set of constraints: $x_1 \leq q - x_5$; if $x_1 = q - x_5$ then $x_2 \leq q - x_8$; if $x_1 = q - x_5$ and $x_2 = q - x_8$ then $x_3 \leq q - x_7$; if $x_1 = q - x_5$ and $x_2 = q - x_8$ and $x_3 = q - x_7$ then $x_4 \leq q - x_6$. This is a cumbersome way of excluding just one symmetrically equivalent solution. Unless we can combine and simplify the constraints arising from different symmetries, which would require time and effort even if possible, we need one such set of constraints for every one of the 48 remaining symmetries.

To break these 48 symmetries in SBDS, we provide a symmetry function for each. This effectively automates the conditional constraints and again gives independence of the variable order. GAP-SBDS, of course, does the same job, but using the group and not the individual elements.

There is a way to break the complement symmetry, using a simple constraint, which is oulined by Beutner and Harboth [Beutner and Harboth, 2002]. If a graph G is graceful it must have an edge labelled q, so two adjoint nodes must be labelled 0 and q. There also must be an edge labelled $q - 1$, which means that either, there are two adjoint nodes labelled 0 and $q - 1$ or, two adjoint nodes labelled 1 and q; by constraining the former, rather than the latter the complement symmetry is broken. It is not possible to derive this constraint from the approach outlined in [Crawford *et al.*, 1996]. So experimentation with this constraint included, would not easily generalise to other problem classes. Hence, this constraint is not considered further within this thesis. However, in the next sections we do consider further the role of symmetry constraints in graceful graphs.

6 GAP-SBDS *v.* Constraints

As in the last section, for the graphs considered here some but not all of the symmetry can easily be broken by adding constraints to the CSP. $K_4 \times K_3$ is composed of three 4-cliques and four 3-cliques and is shown in figure 3 with the node numbering used in the CSP. It was not previously known to be graceful: figure 3 shows one of its graceful labellings.

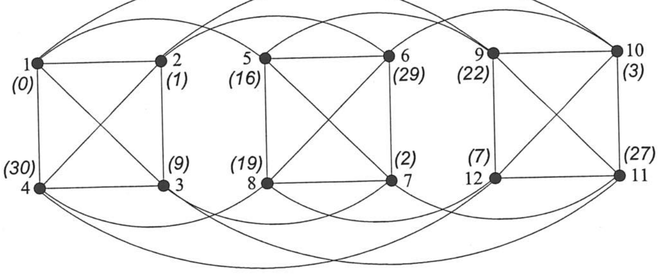

Figure 3: The graph $K_4 \times K_3$.

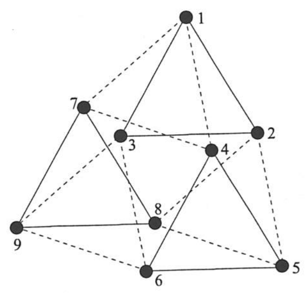

Figure 4: The graph $K_3 \times K_3$.

The variables of the CSP could also be represented by a 3 × 4 matrix $\begin{pmatrix} x_1 & x_2 & x_3 & x_4 \\ x_5 & x_6 & x_7 & x_8 \\ x_9 & x_{10} & x_{11} & x_{12} \end{pmatrix}$ in which the rows represent the 4-cliques and the columns represent the 3-cliques. The graph symmetry can be translated into row and column symmetry in the matrix, i.e. given any solution, permuting the rows and/or the columns gives another. Flener *et al.* [Flener *et al.*, 2002] show that if a CSP can be represented by a matrix of variables with row and column symmetry, and all the values in the matrix are required to be distinct, as here, then the row and column symmetry can be broken by constraints that the largest value should be in the bottom-right corner of the matrix and that the last row and last column should be ordered. In this case, $x_{12} = 30$ (the number of edges) and since there must be an edge joining the nodes labelled 0 and 30, either $x_4 = 0$ or $x_9 = 0$.

Because of the complement symmetry, breaking the row and column symmetry does not break all the symmetry of the problem; however, we can use GAP-SBDS to break all 288 symmetries ($4! \times 3! \times 2$).

Finding all graceful labellings of this graph takes too long using ECL^{i}PS^e, and we restricted the search to solutions in which the edge with value 30 occurs in a 4-clique rather than a 3-clique. When using the symmetry constraints, this means that $x_9 = 0$, since $x_{12} = 30$. GAP-SBDS found the 17 non-isomorphic solutions in 10.6 hours runtime, but only 29 solutions had been found using the symmetry constraints after allowing 50% more time. (Since the complement symmetries are not being broken, there are 34 possible solutions.)

A possible factor in the poor performance of the symmetry constraints is that they conflict with the variable ordering. The constraints have been used as stated in [Flener *et al.*, 2002] and they do not seem an obviously bad choice. However, the graph symmetries can be broken by forcing *any* corner element of the matrix to be the maximum element of the matrix, and ordering the row and column containing the corner. To investigate these different symmetry constraints, while keeping the same variable ordering, a smaller graph, $K_3 \times K_3$, shown in figure 4 is considered, which can be represented by a 3×3 matrix. This is not graceful: any graph in which every node has even degree and the number of edges is congruent to 1 or 2 (mod 4) is known to be not graceful.

As shown in figure 4, the graph is made up of two sets of triangles. As well as permutations within each set of triangles, corresponding to row and column permutations in the matrix $\begin{pmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{pmatrix}$ One set of triangles can be exchanged for the other, corresponding to transposing the matrix. Hence, the problem has $3! \times 3! \times 2 \times 2$ symmetries, or 144; constraints to eliminate the row and column symmetries of the matrix will only eliminate 36. In table 4, GAP-SBDS and symmetry-breaking constraints are compared, in proving that the graph is not graceful. GAP-SBDS is used with the full symmetry group of 144 elements, and with subgroups. Those elements involving interchanging the two sets of triangles are omitted first; since $K_4 \times K_3$ does not have this symmetry, these results are comparable to those for the larger graph. Secondly, the complement symmetries are also omitted, giving a subgroup of size 36; GAP-SBDS is then breaking the same symmetries as the constraints.

GAP-SBDS, breaking:		BT	sec.
	144 symmetries	1393	68
	72 symmetries	2651	137
	36 symmetries	5513	207
Symmetry-breaking constraints, with maximum element at:	top-left	5499	144
	top-right	7050	184
	bottom-left	5008	148
	bottom-right	8276	241

Table 4: Comparison of GAP-SBDS and symmetry breaking constraints in proving that $K_3 \times K_3$ is not graceful.

The row and column symmetries of the corresponding matrix are broken as for $K_4 \times K_3$, and then the results of constraining the maximum element to be in each of the four corners in turn are compared. (Instead a corner element could be forced to be the *minimum* element in the matrix, and change the ordering constraints on the row and column containing the corner appropriately. However, in ECL^{i}PS^e this gives exactly the same number of backtracks for this problem.)

Table 4 shows that with symmetry-breaking constraints, the least search is done when the maximum element of the matrix is constrained to be in the bottom-left corner, and that the bottom-right corner (as used for $K_4 \times K_3$) is the worst choice. It seems to us that this behaviour would be hard to predict, and that *a priori* any of the four choices seem reasonable. When GAP-SBDS breaks the same symmetry as the constraints, its performance is comparable: it does better than the worst choice of constraints, but worse than the best. When it breaks more symmetry than the constraints, i.e. when it is given the full symmetry group of 144 elements, or the subgroup of 72 elements, it does less search and is faster.

In the $K_4 \times K_3$ problem, similarly, GAP-SBDS breaks all the symmetry and the constraints only half. The experience with the $K_3 \times K_3$ graph suggests that this, rather than a conflict with the variable ordering, is the reason for the poor performance of the symmetry constraints relative to GAP-SBDS, and that changing the constraints to be more compatible with the variable ordering would not improve their performance sufficiently to beat GAP-SBDS.

Gent *et al.* [Gent *et al.*, 2002] compared GAP-SBDS and symmetry constraints in solving two BIBDs. As with the graphs considered in this section, BIBDs can be represented as matrices of variables with row and column symmetry. It is not usually possible to find simple constraints which are guaranteed to break all the row and column symmetry in a matrix; the graceful graph problems are an exception because of the requirement that all the values must be different. In the BIBDs, constraints were imposed to order the rows and columns lexicographically. Such constraints are not usually able to break all the symmetry, and from the results here it should therefore be expected that GAP-SBDS would be faster. However, the constraints did break all the symmetry in the two BIBD instances considered. The problems were solved faster using the constraints than with GAP-SBDS, but it is not clear whether this is because the constraints did break all the symmetry in these instances, or whether the size of the symmetry group (millions of elements) also played a part.

7 Double-wheel Graphs

The final category of graphs that have been investigated consist of two wheels, W_m, with a common hub: they are referred to as DW_m. They could be composed as $(C_m \bigcup C_m)+K_1$, i.e. two copies of the cycle C_m, with each vertex joined to a central point. Figure 5 shows DW_5 with a graceful labelling.

This class of graphs was selected because *all* the symmetry can be broken using simple constraints; and hence it allows comparison between GAP-SBDS and symmetry constraints in a problem class where it is expected that the constraints will do well. It is believed that the gracefulness of these graphs has not previously been investigated: it has been shown that DW_3 is not graceful, whereas DW_4 and DW_5 are, with 44 and 1216 non-isomorphic labellings respectively.

The two cycles, C_m, each have rotation and reflection symmetries. These can be eliminated by adding, for the cycle consisting of nodes 2, 3, 4, ..., $m+1$, the m constraints $x_2 < x_3$,

		GAP-SBDS		Constraints	
Graph	Symmetries	BT	sec.	BT	sec.
DW_3	144	48	1.95	21	0.34
DW_4	256	1053	36.1	911	13.3
DW_5	400	33622	1609	26115	539
	Ordering	BT	sec.	BT	sec.
DW_4	1,2,6,7,8,9,3,4,5	963	29.3	570	11.0
DW_4	1,2,3,4,5,6,7,8,9	1053	36.1	911	13.3
DW_4	2,3,4,5,6,7,8,9,1	1328	54.1	1190	49.6
DW_4	9,8,7,6,5,4,3,2,1	1328	53.0	1311	84.5

Table 5: Comparison of GAP-SBDS and symmetry breaking constraints for finding all graceful labellings of double-wheel graphs.

$x_2 < x_4$, ..., $x_2 < x_{m+1}$ and $x_3 < x_{m+1}$. Similar constraints can be added for the second cycle. The labels of the two cycles can also be interchanged: we eliminate that symmetry by a constraint that the smallest node label in the first cycle is less than the smallest in the second. With the other constraints, this simplifies to $x_2 < x_{m+2}$.

The central node (node 1) is unaffected by the graph symmetries. This allows the complement symmetries in these graphs to be easily broken. Given any solution and its complement, one has $x_1 \leq q/2$ and the other $x_1 \geq q/2$. However, x_1 cannot be equal to $q/2$: node 1 would then be connected to every other node, and in particular to those nodes labelled 0 and q, and so there would be two edges labelled $q/2$. Hence, the constraint $x_1 < q/2$ ensures that we do not get both a solution and its complement and so eliminates the complement symmetries.

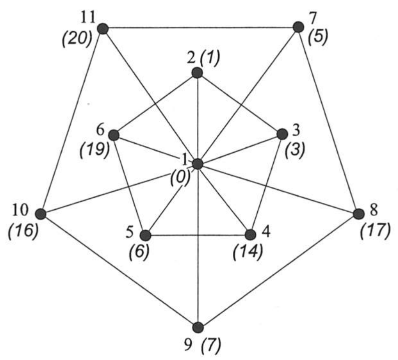

Figure 5: The double wheel DW_5, with a graceful labelling.

Hence, for the graphs in this class, all the symmetry is eliminated by adding $2m + 2$ constraints to the CSP. These constraints could be derived using the procedure from [Crawford *et al.*, 1996] described earlier, assuming that variables will be assigned in the order $x_1, x_2, ..., x_{2m+1}$. On the other hand, the symmetry group has $2m \times 2m \times 2 \times 2$, or $16m^2$, elements, so GAP-SBDS must handle a group of this size to eliminate all the symmetry.

Table 5 compares symmetry-breaking constraints and GAP-SBDS for these graphs. Constraints are the better choice, when the variables are assigned in the usual order $x_1, x_2, x_3, x_4, ...$ However, the difference in speed is not as great as might be expected, given that in DW_5, say, GAP-SBDS is handling a group of 400 elements, whereas only 12 constraints are required for the same task. Clearly, GAP-SBDS is dealing with a large group very efficiently.

Symmetry-breaking constraints are sensitive to the variable ordering, and in Table 5 the effect of changing the variable ordering for graph DW_4 is shown. The results are repeated for the original ordering, and results are given for a better ordering (found by trial and error) and for orderings with the central node variable assigned last. GAP-SBDS is much less affected by the variable ordering than the symmetry-breaking constraints are. In fact, 2,3,4,5,6,7,8,9,1 is symmetrically equivalent to 9,8,7,6,5,4,3,2,1 (and many other orderings) in GAP-SBDS, and hence the number of backtracks is the same, whereas the symmetry-breaking constraints have the effect of differentiating between these orderings. It is notable that the worst ordering takes much longer with constraints than with GAP-SBDS. Although a user might be unlikely to choose such a bad ordering in this case, it is a risk of using constraints to break symmetry. Gent *et al.* [Gent *et al.*, 2002] similarly showed that in their BIBD experiments, the lexicographic ordering constraints performed very poorly given the wrong variable and value ordering, whereas GAP-SBDS was much more robust.

8 Finding One Solution

It is sometimes thought that breaking symmetry is only important when finding all solutions. However, even when only one solution is required, finding one can require searching large subtrees which contain no solution and hence symmetry in the CSP can still lead to wasted search. (Of course, if a problem has no solution, then it makes no difference whether the intention was to find one or all.)

For some of the smaller graphs, finding a single solution requires the same search effort whether or not we break any of the symmetry and irrespective of whether we use SBDS, GAP-SBDS or constraints (although the runtime varies).This is true, for instance, for $K_4 \times P_2$ and $K_4 \times P_3$. However, for $K_5 \times P_2$ and $K_4 \times K_3$ finding a solution takes a significant amount of search, and symmetry breaking saves a lot of wasted search, as shown in table 6. Here, the symmetry breaking constraints eliminate only the graph symmetries, and not the complement symmetries.

In the two cases shown in table 6, both GAP-SBDS and symmetry constraints show a significant saving over having no symmetry breaking, for both graphs. However, SBDS does not always compete with having no symmetry breaking in terms of runtime. In the $K_4 \times K_3$ graph, where this is the case, there are 288 symmetries which relates to more of an overhead for SBDS, than finding just the first solution can justify. In general, these results confirm that both dynamic and static symmetry breaking can be used to good effect when searching for one solution, on relatively large problems, with a proportionally large symmetry group.

	SBDS		GAP-SBDS		Symmetry constraints		No symmetry breaking	
Graph	BT	sec.	BT	sec.	BT	sec.	BT	sec.
$K_5 \times P_2$	3368	998	3381	300	5138	324	30010	1850
$K_4 \times K_3$	25698	3150	25698	1930	12087	1100	40702	2360

Table 6: Finding one graceful labelling of $K_5 \times P_2$: comparison of symmetry breaking methods and no symmetry breaking.

9 Conclusions

An experimental comparison has been carried out of a number of symmetry-breaking techniques on graceful graph problems. These problems allow graphs with different symmetry to be chosen. The symmetry of the graph also combines with the complement symmetry, which doubles the size of the symmetry group.

Two techniques have been compared which break symmetry during search, namely SBDS and GAP-SBDS. In comparing SBDS and GAP-SBDS, our experiments with the $K_m \times P_l$ graceful graph problems have confirmed that increasing numbers of symmetries seriously affect the speed of SBDS. If the symmetry group is small enough (in one case, 120 elements), SBDS is faster than GAP-SBDS. The boundary is not clearly defined, and it appears to depend on other factors such as the type of symmetry and the problem being solved, rather than just the number of symmetries. However, since GAP-SBDS is slower on small problems, but still acceptable, whereas SBDS is unusable on large problems, GAP-SBDS is the better general choice, from the point of view of performance.

GAP-SBDS has also been compared with adding constraints to the CSP to eliminate symmetry. For finding all solutions to these graph problems, if simple symmetry-breaking constraints can be devised to eliminate all the symmetry in a problem, they are faster than GAP-SBDS. However, if there is a choice between breaking some of the symmetry using constraints or breaking all of it using GAP-SBDS, then GAP-SBDS does less search and runs faster. This may be generally true when the constraints break at most half of the symmetry, as in the problems considered here. GAP-SBDS has the additional advantage of being independent of the search order; symmetry constraints are based on the assumption of a particular search order and deviating from that order can delay finding solutions. When just looking for one solution the comparison between symmetry breaking constraints and GAP-SBDS is not so clear. However, both methods can provide an improvement both in search effort and runtime over no symmetry breaking being undertaken.

With good symmetry breaking, constraint programming is a valuable tool for finding all graceful labellings of symmetric graphs or proving that they are not graceful. This investigation has produced several new results on graceful graphs, and those for $K_m \times P_n$ graphs are included in the latest version of Gallian's survey [Gallian, 2004].

References

[Beutner and Harboth, 2002] D. Beutner and H. Harboth. Graceful labelings of nearly complete graphs. *Results in Mathematics*, 41:34–39, 2002.

[Crawford *et al.*, 1996] J. Crawford, M. L. Ginsberg, E. Luks, and A. Roy. Symmetry-breaking predicates for search problems. In Luigia Carlucci Aiello, Jon Doyle, and Stuart Shapiro, editors, *KR'96: Principles of Knowledge Representation and Reasoning*, pages 148–159. Morgan Kaufmann, San Francisco, California, 1996.

[Flener *et al.*, 2002] P. Flener, A.M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Breaking row and column symmetries in matrix models. In P. Van Hentenryck, editor, *Proc. of CP'02*, LNCS 2470, pages 462–476. Springer, 2002.

[Gallian, 2004] J. A. Gallian. A Dynamic Survey of Graph Labeling. *The Electronic Journal of Combinatorics (DS6)*, 2004. (`http://www.combinatorics.org/Surveys`).

[GAP, 2000] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.2*, 2000. (`http://www.gap-system.org`).

[Gent *et al.*, 2000] I.P. Gent, S.A. Linton, and B.M. Smith. Symmetry breaking in the alien tiles puzzle. Technical Report APES-22-2000, APES Research Group, October 2000. Available from http://www.dcs.st-and.ac.uk/~apes/apesreports.html.

[Gent *et al.*, 2002] I. P. Gent, W. Harvey, and T. Kelsey. Groups and constraints: Symmetry breaking during search. In P. Van Hentenryck, editor, *Proc. of CP'02*, LNCS 2470, pages 415–430. Springer, 2002.

[Henz, 2001] M. Henz. Scheduling a major college basketball conference—revisited. *Operations Research*, 49(1):163–168, 2001. ←

[I.J.Lustig and Puget, 2001] 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. ←

[Puget, 2004] J.-F. Puget. Breaking symmetries in all different problems. In *Proceedings SymCon-04: Symmetry and Constraint Satisfaction Problems*, pages 71–78, 2004.

Breaking symmetries in symmetric matrices

Jean-Francois Puget
ILOG
9 Avenue de Verdun
94253 Gentilly Cedex, France
puget@ilog.fr

Abstract

The lex^2 constraint is a well known way to break many of the variable symmetries in matrix models that have both full row and column symmetry. We derive in this paper a similar set of symmetry breaking constraints for symmetric matrices, i.e. matrices M such that $M = M^t$. We apply our new constraint to a combinatorial problem. We show that this is more efficient than previous work.

1 Introduction

Adding symmetry breaking constraints is one of the oldest ways of breaking variable symmetries for constraint satisfaction problems (CSPs) [Puget, 1993]. For instance, it is shown in [Crawford et al., 1996] 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 is however possible to remove many symmetries using a polynomial number of constraints. For instance, when the variables of the CSP can be represented as a $m \times n$ matrix X such that any row or column permutation is a symmetry, we say that the CSP is a two dimensional *matrix model* with full row and full column symmetry. In such case, the size of the symmetry group is $|G| = m!n!$ elements. For matrix models, it is possible to add a lexicographic order on both the rows and the columns [Lubiw, 1987] [Flener et al., 2002]. We denote this set of constraints by lex^2. It is also shown in [Flener et al., 2002] that although lex^2 breaks many symmetries, it does not break all of them.

We consider in this paper another kind of matrix models where the matrix of variables is symmetric, i.e. is such that $X = X^t$. This means that the matrix is square and equal to its transpose. Such matrices arise for instance as incidence matrices of graphs. When we search for a graph on n vertices satisfying some property, we might want to solve this with a CSP where there is a $0-1$ variable x_{ij} for each pair of nodes (i,j): the variable x_{ij} equals 1 if and only if there exists an edge between i and j. The variable x_{ij} equals the variable x_{ji} by definition. Therefore the 2 dimensional matrix of the variables x_{ij} is symmetric. Moreover, this matrix has some symmetries. Indeed, any permutation of the vertices of the underlying graph is a symmetry of the problem. A permutation of the vertices is translated into a permutation of the rows and of the columns of the matrix. For instance, for a graph with 4 vertices, the matrix of variables can be depicted as the one in figure 1a. When we permute the first two vertices, we permute the first two rows as well as the first two columns. The result is shown in figure 1.b.

$$\boxed{\begin{array}{cccc} A & B & C & D \\ B & E & F & G \\ C & F & H & I \\ D & G & I & K \end{array}} \qquad \boxed{\begin{array}{cccc} E & B & F & G \\ B & A & C & D \\ F & C & H & I \\ G & D & I & K \end{array}}$$

(a) (b)

Figure 1: Some symmetric matrices.

Another case of symmetric matrices appears in quasi group completion problems: variable x_{ij} stands for the product of i by j. If multiplication is commutative, i.e. when $x_{ij} = x_{ji}$, then the multiplication table is represented by a symmetric matrix.

This paper is organized as follows. Section 2 contains definitions and notations used in the rest of the paper. We show in section 3 that the lex^2 constraint is valid for symmetric matrices. We apply this result to a Ramsey problem in section 4. Experimental results show that this symmetry breaking constraint is much more efficient than previously proposed ones. We conclude in section 5.

2 Notations

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 variables of a graph with integers, such that any variable symmetry is completely described by a permutation of the labels of its variables. This is formalized as follows.

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 $[1^\sigma, 2^\sigma, \ldots, n^\sigma]$. The product of two permutations σ and θ is defined by $i^{(\sigma\theta)} = (i^\sigma)^\theta$.

A *constraint satisfaction problem* $\mathcal{P}$ (CSP) with m variables is a tuple $\mathcal{P} = (\mathcal{X}, \mathcal{V}, \mathcal{D}, \mathcal{C})$ where $\mathcal{X}$ is a finite set of variables $(x_i)_{i \in I^n}$, $\mathcal{V}$ is a finite set of values, $\mathcal{D}$ a finite set of finite sets $(\mathcal{D}_i)_{i \in I^n}$, and every constraint in $\mathcal{C}$ is a subset of the cross product $\bigotimes_{i \in I^m} \mathcal{D}_i$ such that $\mathcal{D}_i \subseteq \mathcal{V}$ for all i. Without loss of generality, we can assume that $\mathcal{V} = I^n$ for some n.

A *literal* is a statement of the form $x_i = j$ where $j \in \mathcal{D}_i$.

An *assignment* is a set of literals, one for each variable of the CSP. A *partial* assignment is a subset of an assignment.

A *solution* to $(\mathcal{V}, \mathcal{D}, \mathcal{C})$ is an assignment that is consistent with every member of $\mathcal{C}$.

A *symmetry* is a bijection from literals to literals that maps solutions to solutions. The symmetries of a CSP form a group : the product of two symmetries is a symmetry, the identity is a neutral element, and every symmetry has an inverse symmetry.

A *variable symmetry* is a symmetry g such that there is a permutation σ of I^m such that $(x_i = j)^g = (x_{i^\sigma} = j)$. In such case, we will denote g by σ:

$$(x_i = j)^\sigma = (x_{i^\sigma} = j) \qquad (1)$$

A *value symmetry* is a symmetry g such that there exists a permutation θ of I^n such that $(x_i = j)^g = (x_i = j^\theta)$. In such case we will denote g by θ:

$$(x_i = j)^\theta = (x_i = j^\theta) \qquad (2)$$

Our definition of symmetries is similar to the one used in [Kelsey et al., 2004].

The transpose of a matrix M is denoted M^t.

A CSP is a symmetric matrix model if the variables of the CSP can be represented in a 2 dimensional matrix $X = (x_{ij})_{i \in I^m, j \in I^n}$ such that $X = X^t$, i.e. if the following holds :

$$\forall i, j \in I^n, \; x_{ij} = x_{ji}$$

We say that this CSP has full row symmetry (equivalently full column symmetry), if the following holds: Given any permutation σ of I^n, then the permutation $\rho(\sigma)$ that maps x_{ij} to $x_{i^\sigma j^\sigma}$ is a variable symmetry. We say that $\rho(\sigma)$ is *induced* by σ. The symmetry amounts to appy the permutation σ to both rows and columns.

3 Breaking Variable Symmetries in Symmetric Matrices

Adding constraints is one of the oldest methods for reducing the number of variable symmetries of a CSP[Puget, 1993]. In [Crawford et al., 1996], it is shown that all the variable symmetries of any CSP can be broken by the following constraints.

$$\forall \sigma \in G, \; \mathcal{V} \preceq \mathcal{V}^\sigma \qquad (3)$$

where V is the vector of the variables of the CSP, and $\preceq$ is the lexicographic ordering relation. Unfortunately, there may be an exponential number of symmetries σ. Several researchers have proposed ways to state only a polynomial number of constraints [Aloul et al., 2003] [Shlyakhter, 2002] [Flener et al., 2002] [Puget, 2005]. In most of these cases the symmetry breaking constraints do not remove all symmetries, but they remove most of them. The purpose of this paper is to derive such a polynomial set for symmetric matrices with row symmetry.

Let us consider an example for the sake of clarity. We consider a 6×6 symmetric matrix depicted in figure 2. The vector of the variables of the problem is ABCDEFGHIJKLMNO. The diagonal for the matrix is empty. The results we will derive can be adapted to cover the case for non empty diagonals as well.

$$\begin{array}{|cccccc|}
\hline
 & A & B & C & D & E \\
A & & F & G & H & I \\
B & F & & J & K & L \\
C & G & J & & M & N \\
D & H & K & M & & O \\
E & I & L & N & O & \\
\hline
\end{array}$$

Figure 2: A 6×6 symmetric matrix.

Let us consider the permutation of the third and fourth rows. Then the third and fourth columns are permuted as well. This yields the matrix depicted in figure 3.

$$\begin{array}{|cccccc|}
\hline
 & A & C & B & D & E \\
A & & G & F & H & I \\
C & G & & J & M & N \\
B & F & J & & K & L \\
D & H & M & K & & O \\
E & I & N & L & O & \\
\hline
\end{array}$$

Figure 3: After permuting third and fourth rows (and columns).

The lex constraint for this symmetry is:

ABCDEFGHIJKLMNO $\preceq$ ACBDEGFHIJMNKLO

This can be simplified into by removing identical variables occurring at identical indices. This gets rid of variables ADEHIJO. We chose to keep J for reasons that will become clear later. This yields the simplified constraint :

BCFGJKLMN $\preceq$ CBGFJMNKL

This can be further simplified by removing pairs of variables already appearing at a lower index. For instance, the first variable of the first vector is B, and C is the first variable of the second vector. This means that the set {B, C} is compared at the first index. Then any occurrence of that set can be removed from the constraint. This set occurs again at the second index since C is the second variable of the first vector, and B is the second variable of the second vector. These variables can be removed from both vectors without modifying the constraint. Getting rid of all duplicated sets yields :

BFJKL $\preceq$ CGJMN

This says that the third row is lexicographically smaller than the fourth row! This also says that the third column is lexicographically smaller than the fourth column. We kept the variable J in both vectors in order to get this nice result.

This example can be turned into a proof of the following result. We denote by σ_{kl} the permutation induced by the permutation of rows k and l, and permutation of columns k and l.

Lemma 1. *Given a matrix X such that $X = X^t$, and given $k < l$ then, if σ_{kl} is a variable symmetry, equation (3) implies that row k is lexicographically smaller than row l.*

As a consequence, if any row permutation is a symmetry, then we can order all the rows lexicographically by repeated application of lemma 1 to permutations of two consecutive rows. Since the matrix is symmetric, this is equivalent to order lexicographically the columns. Therefore, we have proved the following result:

Theorem 2. *Given a symmetric matrix X with full row symmetry, then equation (3) implies the lex^2 constraints on X.*

The above proof is inspired by a the proof of the validity of the lex^2 constraint given in [Shlyakhter, 2002] for 2 dimensional matrices with full row and column symmetry. A similar proof was later given in [Kiziltan, 2004]. The allperm constraint of [Frisch et al., 2003] was derived in a similar way as well.

We have proved that the lex^2 constraint was valid for symmetric matrices. It is known that the lex^2 constraint does not break all symmetries for non symmetrical matrices. Is this the case for symmetric matrix? The answer is yes unfortunately. We can even use the negative example given in [Flener et al., 2002] to construct a negative example for symmetric matrix. Indeed, any 2 dimensional matrix can be used to define a symmetric matrix as follows.

Given a $m \times n$ matrix A with full row and column symmetry, we construct the matrix depicted in figure 4. The $m \times m$ upper left quarter is filled with 0 except for the diagonal, as well as the $n \times n$ lower quarter. The upper right quarter is the matrix A. The lower left quarter is the transpose of A. The diagonal of this matrix is empty by definition.

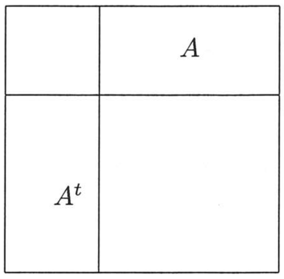

Figure 4: A symmetric matrix.

Any permutation of the first m rows of this matrix is a symmetry, because A has a full row symmetry. Therefore, we can order lexicographically the first m rows by repeated application of lemma 1 to permutations of two consecutive rows. This amounts to lexicographically order the rows of A. Similarly, any permutation of the last n rows is a symmetry because A has full column symmetry. Therefore, we can order lexicographically the last n rows by repeated application of lemma 1 to permutations of two consecutive rows. This amounts to lexicographically order the rows of A^t. In turn, this is equivalent to lexicographically order the columns of A. We have proved that staing lex^2 on the matrix of figure 4 is equivalent to stating the lex^2 matrix for A.

Let us look at the matrices given in figure 5. These matrices were given in [Flener et al., 2002]. Both have rows and columns lexicographically ordered. One can move from one to the other by swapping the first two rows and the last two columns. This shows that lex^2 does not break all symmetries.

$$\underset{\text{(a)}}{\begin{array}{|ccc|}\hline 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \\ \hline\end{array}} \qquad \underset{\text{(b)}}{\begin{array}{|ccc|}\hline 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \\ \hline\end{array}}$$

Figure 5: Two matrices with rows and columns ordered.

We can now construct two symmetrical matrix, by applying the construct of figure 4 with each of the matrices in figure 5. This yields two symmetrical matrices given in figure 6. Both matrices have rows and columns lexicographically ordered. However, one can move from one to the other by swapping the first two rows (and two columns) and the last two columns (and last two rows).

$$\underset{\text{(a)}}{\begin{array}{|cccccc|}\hline & 0 & 0 & 0 & 0 & 1 \\ 0 & & & 0 & 1 & 0 \\ 0 & 0 & & 1 & 0 & 1 \\ 0 & 0 & 1 & & 0 & 0 \\ 0 & 1 & 0 & 0 & & 0 \\ 1 & 0 & 1 & 0 & 0 & \\ \hline\end{array}} \qquad \underset{\text{(b)}}{\begin{array}{|cccccc|}\hline & 0 & 0 & 0 & 0 & 1 \\ 0 & & 0 & 0 & 1 & 0 \\ 0 & 0 & & 1 & 1 & 0 \\ 0 & 0 & 1 & & 0 & 0 \\ 0 & 1 & 1 & 0 & & 0 \\ 1 & 0 & 0 & 0 & 0 & \\ \hline\end{array}}$$

Figure 6: Two matrices with rows and columns ordered.

This proves the following.

Theorem 3. *Given a symmetric matrix X with full row symmetry, then the lex^2 constraints on X does not break all symmetries.*

In [Kiziltan, 2004] other symmetry breaking constraints are studied, namely multiset ordering constraints. It is shown that one can use these constraints to order both rows and columns for matrices that have full row and full column symmetries. A similar argument shows that this is also true for symmetric matrices.

Theorem 4. *Given a symmetric matrix X with full row symmetry, then X has a symmetrical variant where the rows and the columns are mutlitset ordered.*

4 The Ramsey Problem

We will use as an example a problem which has been proposed in [Puget, 1993], namely the Ramsey problem. Consider the complete graph with n nodes (each node is connected to every other node). The problem is to color the edges of this graph with 3 colors, such that for any 3 nodes i, j, k, the three edges $(i,j)(j,k)(k,i)$ do not have all three the same color (but two of them can have the same color). For $n = 16$, this problem has a lot of solutions. For $n = 17$ there is no solution. It is not very difficult to write a program for finding solutions for $n \leq 16$. The challenge is to write a program which solves the problem for $n \leq 16$, and proves that there is no solution for $n = 17$. This problem can be modeled with a symmetric matrix X such that x_{ij} represents the color assigned to the edge (i, j). Since the graph is non oriented, we have that $x_{ij} = x_{ji}$. The color constraint states that

$$\forall i, j, k,\ x_{ij} \neq x_{ik} \vee x_{ik} \neq x_{ki} \vee x_{ki} \neq x_{ij}$$

In fact, using constraint reification, stronger constraints can be stated:

$$\forall i, j, k,\ (x_{i,j} = x_{i,k}) + (x_{i,k} = x_{k,i}) + (x_{k,i} = x_{i,j}) \leq 1$$

Indeed, in any triangle, we can have at most two edges of the same color.

This model has full row symmetry. It also has some value symmetries. In fact, any value permutation is a value symmetry here.

4.1 First Model

In order to break value symmetries, it was proposed in [Puget, 1993] to introduce extra variables counting the number of occurrences of each value. We therefore define the variable z_{ik} to be the number of occurrences of the value k in the i-th row of the matrix X. The channeling constraints between the variables z_{ik} and the variables x_{ij} are generalized cardinality constraints [Regin, 1996]. The z_{ik} variables are arranged in a $n \times c$ matrix Z ($c = 3$ in our example).

The following symmetry breaking constraints were proposed in [Puget, 1993]:

The first element of the Z matrix is the largest element of that matrix:

$$\forall i \in I^n, \forall k \in I^c,\ z_{11} \geq z_{ik} \tag{4}$$

The first row of the Z matrix is decreasing:

$$\forall k \in I^{c-1},\ z_{1k} \geq z_{1,k+1} \tag{5}$$

The colors of the first row are in increasing order :

$$\forall j \in I^n,\ x_{1j} \leq x_{1,j+1} \tag{6}$$

4.2 Second Model

The X matrix is symmetrical, therefore one could state lex^2 on it. Our second model is obtained by stating (4), (5) and lex^2. We need to show that these constraints are consistent.

The approach of [Crawford et al., 1996] was defined for Boolean variables and variable symmetries. We can extend it to arbitrary variables and arbitrary symmetries as follows. A solution is said to be canonical if it is minimal among all its symmetrical equivalent. More formally, given G a group of symmetries (containing variable symmetries, or values symmetries, or both) we define the orbit of a solution s to be the set:

$$s^G = \{s^\sigma | \sigma \in G\}$$

We say that a solution is canonical if it is minimal among its orbit:

$$\forall \sigma \in G,\ s \preceq s^\sigma$$

We say that a constraint is a symmetry breaking constraint if it removes some solutions but removes no canonical solutions. A symmetry breaking constraint is the following:

$$\forall \sigma \in G,\ \mathcal{V} \preceq \mathcal{V}^\sigma \tag{7}$$

This is very similar to (3). The difference is that now G may contain value symmetries in addition to variable symmetries. Stating these constraints remove all solutions except for the canonical ones.

Let us go back to our ramsey problem, where the vector of variables V is arranged in the X matrix. We have shown that lex^2 is implied by (3), hence it is implied by (7).

Assume that we have a canonical solution s. We have shown that s is consistent with lex^2. Therefore, the values in the first row of X for this solution are in increasing order. This row starts with $z_{1,1}$ occurences of 1, followed by occurences of other values. Assume s is not consistent with (4). Let us look at the values of the variables z_{ik}. There exists a variable $z_{i,k}$ such that $z_{i,k} > z_{11}$. Let us swap rows 1 and i, columns 1 and i, and colors 1 and k. Then let us order the columns and rows of the X matrix. This yields another solution where the first row starts with $z_{i,k}$ occurences of 1. Since $z_{i,k} > z_{1,1}$, this new solution is lexicographically saller than s, which is a contradiction. Therefore, (4) does not remove any canonical solution.

Assume s is a canonical solution that is not consistent with (5). Let us look at the values of the variables z_{ik}. If the first row of the Z matrix is non decreasing, then there exists k such that $z_{1k} < z_{1,k+1}$. Let c_k be the first column with value k. Then in the first row of the solution, there are $z_{1,k}$ values k, starting at index c_k, followed by $z_{1,k+1}$ values $k+1$. Then let us swap the colors k and $k+1$, and order the columns of X after that. This yields a new solution. In this solution, the first row is the same as the one in the other solution, except for the occurences of k and $k+1$. In the new solution, there are $z_{1,k+1}$ occurences of k, starting at c_k, followed by $z_{1,k}$ occurences of $k+1$. Since $z_{1,k+1} > z_{1,k}$, the second solution is lexicographically smaller. Therefore the first solution is not canonical, which is a contradiction. This proves that (5) does not remove any canonical solution.

We have proved that none of (4), (5) and $lex2$ remove canonical solutions. Therefore their conjunction will not remove any canonical solutions. Hence, our second model contains all canonical solutions.

We ran experiments using both models. Results are given in Table 1. For each model we give the number of solutions found, the number of backtracks and the running times. The second model is clearly superior. The only difference between the two models is the variable symmetry breaking constraint : lex^2 vs (6).

n	first model			second model		
	nsol	bt	time	nsol	bt	time
4	24	3	0.01	12	5	0
5	343	7	0.01	122	16	0.01
6	7,697	50	0.32	1,174	58	0.08
7	252,325	453	5.6	20,773	266	0.57
8	7,752,909	35,571	202	355,663	3,318	9.09
14				16,285	210,747	26.38
15				30	7,708	1.16
16	1,036,800	35,699,452	6,559	6	713	0.25
17	0	651	0.12	0	171	0.06

Table 1. Result for finding all solutions.

5 Conclusion

We have presented a simple symmetry breaking constraint for symmetric matrix models with full row symmetry. This symmetry breaking constraint is analogous to the lex^2 constraint for matrix models with full row and column symmetries. Experiments on a combinatorial problem show that these constraints break much more symmetries than previously published work.

References

[Aloul et al., 2003] F. Aloul, I. Markov, and K. Sakallah "Efficient Symmetry-Breaking for Boolean Satisfiability". *In proceedings of IJCAI 03*, Acapulco, Mexico, pp. 271-282, 2003.

[Crawford et al., 1996] Crawford, J., Ginsberg, M., Luks E.M., Roy, A. "Symmetry Breaking Predicates for Search Problems." In proceedings of KR'96, 148-159.

[Flener et al., 2002] 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

[Frisch et al., 2003] Alan M. Frisch, Chris Jefferson, and Ian Miguel "Constraints for Breaking More Row and Column Symmetries ". Proceedings of CP'03.

[Kelsey et al., 2004] Kelsey, T, Linton, SA, Roney-Dougal, CM. "New Developments in Symmetry Breaking in Search Using Computational Group Theory". *In Proceedings AISC 2004*. Springer LNAI. 2004.

[Kiziltan, 2004] Z.Kiziltan, *Symmetry Breaking Ordering Constraints* PhD dissertation, Uppsala University, March 2004.

[Lubiw, 1987] Anna Lubiw Doubly lexical orderings of matrices, SIAM J. on Computing 16, 1987, 854-879.

[Puget, 1993] Puget, J.-F.: "On the Satisfiability of Symmetrical Constraint Satisfaction Problems." Proceedings of ISMIS'93 (1993), 350–361.

[Puget, 2005] Puget, J.-F.: "Breaking symmetries in all different problems". To appear in proceedings of IJCAI 05.

[Regin, 1996] J-C. Regin: "Generalized Arc Consistency for Global Cardinality Constraint", AAAI-96 Portland, OR, USA, pp 209–215, 1996

[Shlyakhter, 2002] Ilya Shlyakhter, "Generating effective symmetry-breaking predicates for search problems." *Discrete Applied Mathematics*, 2002.

Sets of Symmetry Breaking Constraints

Barbara M. Smith
Cork Constraint Computation Centre, University College Cork, Ireland
b.m.smith@4c.ucc.ie

Abstract

[Puget, 2004] has shown that if the symmetry in a constraint satisfaction problem acts only on the variables and there is also an allDifferent constraint on the variables, the symmetry can be eliminated by adding a small number of constraints, linear in the number of variables. In this paper, Puget's procedure for finding a set of symmetry-breaking constraints is extended to find all possible distinct sets. It is shown that there can be exponentially many distinct sets of symmetry breaking constraints, leading to the need to choose between them. The choice depends on how the problem will be solved, and specifically on the variable order. Since a variable order can lead to a choice of symmetry-breaking constraints, it seems plausible that the same variable order should be used during search; however, experiments with a graceful graph problem do not show that pairing the symmetry-breaking constraints with a compatible variable order in this way leads to reduced search.

1 Introduction

If the symmetries of a CSP permute the variables, and not the values, there is a systematic procedure for generating symmetry-breaking constraints, given by Crawford, Ginsberg, Luks and Roy [Crawford *et al.*, 1996]. This procedure requires an order of the variables to be specified, and the symmetry breaking constraints that are derived depend on this order; hence in theory, if there are n variables, there could be $n!$ different sets of symmetry-breaking constraints. In practice, this does not happen; different variable orders can give exactly the same constraints, or different orders can yield equivalent constraints. Taking this into account, there may be only a few distinct sets of symmetry-breaking constraints that we can consider adding to the CSP, or perhaps only one.

This is true in general of these kinds of symmetry and symmetry-breaking constraints. In this paper, the question of how many distinct sets of symmetry-breaking constraints there can be is investigated, specifically for problems where there is an allDifferent constraint on the variables. Puget has shown [Puget, 2004] that in such a case, the symmetry can be broken with a small number of constraints (linear in the number of variables) and gives a procedure for generating them from the symmetry group. Here, the procedure is adapted to generate all the symmetrically distinct sets of constraints. As in [Puget, 2004], instances of the 'graceful graph' problem are used as examples, since these do have variable symmetry and an allDifferent constraint on the variables. Graceful graph problems also have a value symmetry, but for the present purposes that is ignored.

2 Graceful Graphs

A labelling f of the nodes of a graph with q edges is *graceful* if f assigns each node a unique label from $\{0, 1, ..., q\}$ 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, ..., q.) [Gallian, 2003] gives a survey of graceful graphs, i.e. graphs with a graceful labelling, and lists the graphs whose status is known.

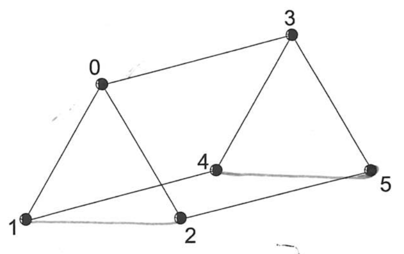

Figure 1: The graph $K_3 \times P_2$: the node numbers correspond to the indices of the CSP variables $x_0, x_1, ..., x_5$.

Figure 1 shows an example of a graph that has a graceful labelling. This graph is an instance of the class of graphs $K_m \times P_n$, consisting of n copies of a m-clique (K_m), with corresponding nodes in each clique joined by a path of length n (P_n). A possible model of this problem as a CSP has a variable for each node, in this case $x_0, x_1, ..., x_5$, corresponding to the node numbering given in Figure 1. The domain of x_i is $\{0..., 8\}$, since there are 9 edges. These are the search variables. We also have a variable for each edge, in this case $d_0, .., d_8$, with domains $\{1..., 9\}$. If edge k joins nodes i and j, we have a constraint $d_k = |x_i - x_j|$. The variables $x_0, x_1, ..., x_5$ are all different, as are $d_0, .., d_{14}$.

As discussed in [Petrie and Smith, 2003], symmetry in the resulting CSP arises in two ways. First there is the symmetry of the graph: in this case, we can permute the nodes within any clique as long as we permute the nodes in the other cliques in the same way; and we can reverse the order of the cliques, so that in this case, the first clique (nodes 0, 1, 2) swaps with the second clique (nodes 3, 4, 5). Second, in any assignment to the node variables, we can replace any value v by $q - v$; this symmetry will be ignored in this paper. For $K_m \times P_n$ graphs, we can eliminate the graph symmetry by adding constraints on the node variables to the model.

3 Examples of Symmetry Breaking Constraints

In this section, two examples are given to show that different variable orders do not necessarily yield different constraints, but may do.

Suppose that we have n variables and the symmetry group acting on them is S_n, i.e all permutations. A graceful graph problem with this symmetry is the problem of finding graceful labellings of a clique. It has been known for a long time, and is easy to show, that the clique K_n is graceful iff $n \leq 4$, so that solving the corresponding CSP is not very interesting in itself; nevertheless, it provides a good illustration.

The CSP model has a variable for each node, $x_0, x_1, ..., x_{n-1}$. If the procedure given in [Crawford *et al.*, 1996] is applied to this problem, taking the variables in lexicographic order, we get a set of symmetry-breaking constraints that can be simplified to $x_0 < x_1 < x_2 ... < x_{n-1}$. (The simplification involves dropping constraints that are implied by combinations of other constraints, through transitivity, e.g. $x_0 < x_2$.)

If we started from another variable order, say $x_{n-1}, x_{n-2}, ..., x_2, x_1, x_0$, we would get a different set of constraints, i.e. $x_{n-1} < x_{n-2} < ... < x_1 < x_0$. However, these constraints are equivalent to the first set: an element of S_n acts on the order $x_0, x_1, ..., x_{n-1}$ to produce $x_{n-1}, x_{n-2}, ..., x_2, x_1, x_0$ and also transforms the first set of constraints into the second. A search for a solution to the first CSP (i.e. with the first set of constraints added) could be transformed into an equivalent search in the second CSP, with the second set of constraints, by applying the same symmetry to the assignments made. Hence, although the $n!$ possible variable orders each yield a different set of symmetry breaking constraints, all these sets are effectively the same.

Now consider the problem of finding graceful labellings of the graph shown in Figure 1. The symmetry of the graph allows the labels of the nodes in each 3-clique to be permuted as long as the labels in the other clique are permuted in the same way; we can also swap the labels of the nodes in one clique with the corresponding nodes in the other clique; and we can combine these transformations. The symmetry group has 12 elements ($3! \times 2$).

The variable order $x_0, x_1, ..., x_5$ would give the symmetry breaking constraints $x_0 < x_1; x_0 < x_2; x_0 < x_3; x_0 < x_4; x_0 < x_5$ and $x_1 < x_2$. In fact, any variable order in which x_0 and x_1 are placed first (as well as some other orders) will give exactly the same constraints. Also, as with labelling K_n, there are variable orders which will give different but equivalent constraints, e.g. $x_6, x_5, .., x_0$. However, starting from a variable order that placed x_0 and x_4 first, we would get the constraints $x_0 < x_1; x_0 < x_2; x_0 < x_3; x_0 < x_4; x_0 < x_5$ and $x_4 < x_5$. These constraints are different from the first set; there is no element of the symmetry group which will transform one into the other. Abstracting from the specific naming of the nodes, the first set of constraints says that an arbitrary node of the six is constrained to have the smallest label, and an arbitrary ordering is imposed on the labels of the other two nodes in the *same* 3-clique. The other set of constraints similarly selects a specific node to have the smallest label, but imposes an arbitrary ordering on the two nodes in the *other* clique which are not connected to the first node.

The next section discusses deriving the distinct sets of symmetry-breaking constraints by extending the method given in [Puget, 2004].

4 Deriving Symmetry-Breaking Constraints

Finding symmetry-breaking constraints using the method described in [Crawford *et al.*, 1996] requires, in theory, writing down the effect of every element of the symmetry group. (In practice, they are often derived more intuitively.) The method given in [Puget, 2004] is specialised for problems where there is an allDifferent constraint on the variables and in that case gives the same results as [Crawford *et al.*, 1996]. It deals with the original symmetry group and subgroups of that, rather than the individual elements of the group. Puget's method can be used to derive the constraints automatically; however, even if the constraints are derived by hand, it is much less time-consuming than that in the earlier paper.

Recall that the symmetry group in a graceful graph problem acts only on the variables of the CSP (ignoring the complement symmetry). The first step in Puget's method is to partition the variables into orbits: the orbit of a point acted on by a group is the set of points that it can be transformed into by the group elements. In finding graceful labellings of both K_n and $K_3 \times P_2$, any node can be transformed into any other node by the graph symmetry, so the initial partition has just one set, containing all the variables.

Next, we choose a (non-singleton) orbit, if there is more than one, and pick an arbitrary variable in that orbit, say x_0. We impose the constraints that x_0 should be less than any other variable in its orbit.

Next we find the stabiliser of x_0; this is the subgroup of the original symmetry group that leaves x_0 fixed. Intuitively, it is easy to do this in the graceful graphs instances; we can for instance imagine fixing node 0 in Figure 1 and see what transformations of the graph are still possible. Clearly, with node 0 immovable, we can still swap x_1 and x_2 and simultaneously x_4 and x_5. The stabiliser therefore has just two elements, including the identity transformation that does nothing.

We again partition the variables into their orbits in the stabiliser of x_0. Since x_0 is by definition fixed by its stabiliser, its orbit just contains itself, and some other variables may also be fixed by the stabiliser of x_0 (in the case of $K_3 \times P_2$, x_3 is also fixed by the stabiliser of x_0.) The other orbits in this example are $\{x_1, x_2\}$ and $\{x_4, x_5\}$. Again, we choose an orbit and pick an element in the orbit, say x_1, and impose the constraint $x_1 < x_2$.

We repeat these steps, finding the stabiliser of x_0 and x_1 (or the stabiliser of x_1 within the stabiliser of x_0, which amounts to the same thing). In the example of $K_3 \times P_2$, if we fix nodes 0 and 1, the whole graph is fixed, so the new stabiliser contains just the identity, and this is the signal to terminate the process.

If we select x_0 from the first orbit, and then choose the orbit $\{x_1, x_2\}$ and x_1 within that, we get the first set of symmetry-breaking constraints given earlier for $K_3 \times P_2$, i.e. $x_0 < x_1; x_0 < x_2; x_0 < x_3; x_0 < x_4; x_0 < x_5$ and $x_1 < x_2$. However, if we chose the orbit $\{x_4, x_5\}$ in the second stage, and x_4 within that, the last constraint would be replaced by $x_4 < x_5$, i.e. the second set of constraints given earlier.

We can therefore see that:

- the choice of one variable rather than another within an orbit will give different sets of symmetry-breaking constraints, but the different choices in this case are symmetrically equivalent, since the group (either the original group or the current stabiliser) acts equally on all elements within an orbit;
- the choice of one orbit rather than another will give different sets of symmetry-breaking constraints which are symmetrically distinct, since the group has a different effect on each orbit.

The process gives the same symmetry-breaking constraints as in [Crawford *et al.*, 1996] if we use a pre-specified order of the variables to guide the choice of orbit and the variable within the orbit. In the $K_3 \times P_2$ example, any variable order that has x_0 first would lead to choosing x_0 from the first orbit. Any variable order that has either x_1 or x_2 before both x_4 and x_5 would choose the orbit $\{x_1, x_2\}$ rather than $\{x_4, x_5\}$; and if the order had x_1 before x_2 it would choose $x_1 < x_2$ rather than $x_2 < x_1$. Hence, many different variable orders would yield exactly the same set of constraints in this example.

5 Alternative Sets of Constraints

We can represent schematically the symmetrically distinct choices, that are given by different orbits, and the constraints that they lead to, as shown in the example below, which shows the derivation of the two distinct sets of symmetry-breaking constraints for $K_3 \times P_2$. (Note that the elements of the original group and the stabilisers are not listed, except when the current stabiliser contains only the identity element, shown as $\{i\}$.)

G_0: symmetry group of $K_3 \times P_2$
orbit: $\{x_0, x_1, x_2, x_3, x_4, x_5\}$
  constraints: $x_0 < x_1; x_0 < x_2; x_0 < x_3; x_0 < x_4;$
  $x_0 < x_5$
    G_1: stabilizer of x_0
    orbit 1: $\{x_1, x_2\}$
      constraints: $x_1 < x_2$
        G_2: stabiliser of $x_0, x_1 = \{i\}$

$$\boxed{\begin{array}{llll} x_0 < x_1; & x_0 < x_2; & x_0 < x_3; & x_0 < x_4; \\ x_0 < x_5; & x_1 < x_2 & & \end{array}}$$

    orbit 2: $\{x_4, x_5\}$
      constraints: $x_4 < x_5$
        G_2: stabiliser of $x_0, x_4 = \{i\}$

$$\boxed{\begin{array}{llll} x_0 < x_1; & x_0 < x_2; & x_0 < x_3; & x_0 < x_4; \\ x_0 < x_5; & x_4 < x_5 & & \end{array}}$$

Returning to the clique, K_n, we get a single set of symmetry breaking constraints, apart from symmetric equivalents. At each stage, the stabiliser of the elements fixed so far is the complete symmetry group, S_k, acting on the remaining k elements, and these elements can all be transformed into each other by the action of the stabiliser, so that there is a single orbit.

G_0: symmetry group of $K_n = S_n$
orbit: $\{x_0, x_1, ..., x_n\}$
  constraints: $x_0 < x_1; x_0 < x_2; .....; x_0 < x_n$
    G_1: stabilizer of $x_0 = S_{n-1}$ acting on
    $x_1, x_2, ..., x_n$
    orbit: $\{x_1, x_2, ..., x_n\}$
      constraints: $x_1 < x_2; x_1 < x_3; ....; x_1 < x_n$
        G_2: stabiliser of $x_0, x_1 = S_{n-2}$ acting on $x_2, x_3, ..., x_n$
        orbit: $\{x_2, ..., x_n\}$
          constraints: $x_2 < x_3; x_2 < x_4; ....; x_2 < x_n$

and so on. As mentioned earlier, the resulting constraints can be simplified into $x_0 < x_1 < x_2 < x_3.... < x_n$.

A more interesting case is $K_3 \times P_3$, shown in Figure 2. In

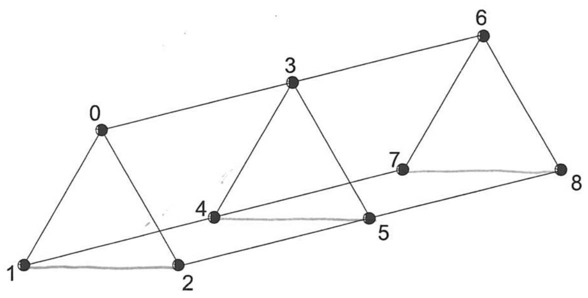

Figure 2: The graph $K_3 \times P_3$

this case, the original symmetry group of the graph partitions the variables into two different orbits, since the nodes in the two outer 3-cliques are interchangeable, as are the nodes in the middle 3-clique, but an outer node cannot be mapped to a middle node. In all, we get 9 distinct sets of symmetry constraints:

G_0: symmetry group of $K_3 \times P_3$
orbit 1: $\{x_0, x_1, x_2, x_6, x_7, x_8\}$
  constraints: $x_0 < x_1; x_0 < x_2; x_0 < x_6; x_0 < x_7;$
  $x_0 < x_8$
    G_1: stabilizer of x_0
    orbit 1: $\{x_1, x_2\}$
      constraints: $x_1 < x_2$
        G_2: stabiliser of $x_0, x_1 = \{i\}$

```
            $\boxed{x_0 < x_1;\ x_0 < x_2;\ x_0 < x_6;\ x_0 < x_7;\ x_0 < x_8;\ x_1 < x_2}$
    orbit 2: $\{x_4, x_5\}$
      constraints: $x_4 < x_5$
        $G_2$: stabiliser of $x_0, x_4 = \{i\}$
            $\boxed{x_0 < x_1;\ x_0 < x_2;\ x_0 < x_6;\ x_0 < x_7;\ x_0 < x_8;\ x_4 < x_5}$
    orbit 3: $\{x_7, x_8\}$
      constraints: $x_7 < x_8$
        $G_2$: stabiliser of $x_0, x_7 = \{i\}$
            $\boxed{x_0 < x_1;\ x_0 < x_2;\ x_0 < x_6;\ x_0 < x_7;\ x_0 < x_8;\ x_7 < x_8}$
orbit 2: $\{x_3, x_4, x_5\}$
  constraints: $x_3 < x_4; x_3 < x_5$
    $G_1$: stabilizer of $x_3$
    orbit 1: $\{x_0, x_6\}$
      constraints: $x_0 < x_6$
        $G_2$: stabiliser of $x_3, x_0$
        orbit 1: $\{x_1, x_2\}$
          constraints: $x_1 < x_2$
            $G_3$: stabiliser of $x_3, x_0, x_1 = \{i\}$
              $\boxed{x_3 < x_4;\ x_3 < x_5;\ x_0 < x_6;\ x_1 < x_2}$
        orbit 2: $\{x_4, x_5\}$
          constraints: $x_4 < x_5$
            $G_3$: stabiliser of $x_3, x_0, x_4 = \{i\}$
              $\boxed{x_3 < x_4;\ x_3 < x_5;\ x_0 < x_6;\ x_4 < x_5}$
        orbit 3: $\{x_7, x_8\}$
          constraints: $x_7 < x_8$
            $G_3$: stabiliser of $x_3, x_0, x_7 = \{i\}$
              $\boxed{x_3 < x_4;\ x_3 < x_5;\ x_0 < x_6;\ x_7 < x_8}$
    orbit 2: $\{x_1, x_2, x_7, x_8\}$
      constraints: $x_1 < x_2; x_1 < x_7; x_1 < x_8$
        $G_2$: stabiliser of $x_3, x_1 = \{i\}$
          $\boxed{x_3 < x_4;\ x_3 < x_5; x_1 < x_2;\ x_1 < x_7;\ x_1 < x_8}$
    orbit 3: $\{x_4, x_5\}$
      constraints: $x_4 < x_5$
        $G_2$: stabiliser of $x_3, x_4$
        orbit 1: $\{x_0, x_6\}$
          constraints: $x_0 < x_6$
            $G_3$: stabiliser of $x_3, x_4, x_0 = \{i\}$
              $\boxed{x_3 < x_4; x_3 < x_5; x_4 < x_5; x_0 < x_6}$
        orbit 2: $\{x_1, x_7\}$
          constraints: $x_1 < x_7$
            $G_3$: stabiliser of $x_3, x_4, x_1 = \{i\}$
              $\boxed{x_3 < x_4; x_3 < x_5; x_4 < x_5; x_1 < x_7}$
        orbit 3: $\{x_2, x_8\}$
          constraints: $x_2 < x_8$
            $G_3$: stabiliser of $x_3, x_4, x_2 = \{i\}$
              $\boxed{x_3 < x_4; x_3 < x_5; x_4 < x_5; x_2 < x_8}$
```

Note that although ten sets of constraints are derived above, the 5th and 8th sets are identical.

6 How Many Sets Can There Be?

Puget showed that variable symmetry in a problem with an allDifferent constraint on the affected variables can be broken by a linear number of binary $<$ constraints. The examples above might suggest that the number of distinct sets of symmetry-breaking constraints might be similarly limited. However, it is not; the following example shows that there can be exponentially many sets. $K_m \times P_2$ is the general class of which $K_3 \times P_2$ is a small example. Under the original symmetry group, all the nodes form a single orbit, but at every later stage, there are two possible orbits, one corresponding to each clique, and hence two choices for the next constraints to add. The following shows the first few levels.

```
$G_0$: symmetry group of $K_m \times P_2$
orbit: $\{x_0, x_1, x_2, ......, x_{2m-1}\}$
  constraints: $x_0 < x_1; x_0 < x_2; ...; x_0 < x_{2m-1}$
    $G_1$: stabilizer of $x_0$
    orbit 1: $\{x_1, x_2, ..., x_{m-1}\}$
      constraints: $x_1 < x_2, x_1 < x_3, ..., x_1 < x_{m-1}$
        $G_2$: stabiliser of $x_0, x_1$
        orbit 1: $\{x_2, x_3, ..., x_{m-1}\}$
          constraints: $x_2 < x_3; x_2 < x_4; ..., x_2 < x_{m-1}$
          ......
        orbit 2: $\{x_{m+2}, x_{m+3}, ......, x_{2m-}\}$
          constraints: $x_{m+2} < x_{m+3}; x_{m+2} < x_{m+3};$
                    $...; x_{m+1} < x_{2m-1}$
          ......
    orbit 2: $\{x_{m+1}, x_{m+2}, ......, x_{2m-1}\}$
      constraints: $x_{m+1} < x_{m+2}; x_{m+1} < x_{m+2};$
                $...; x_{m+1} < x_{2m-1}$
        $G_2$: stabiliser of $x_0, x_{m+1}$
        orbit 1: $\{x_2, x_3, ..., x_{m-1}\}$
          constraints: $x_2 < x_3; x_2 < x_4; ..., x_2 < x_{m-1}$
          ......
        orbit 2: $\{x_{m+2}, x_{m+3}, ......, x_{2m-1}\}$
          constraints: $x_{m+2} < x_{m+3}; x_{m+2} < x_{m+3};$
                    $...; x_{m+1} < x_{2m-1}$
          ......
```

So in constructing a set of symmetry-breaking constraints, we can flip backwards and forwards between the two cliques, adding constraints between the variables of either clique.

At each point where there is a choice between two orbits, the first constraint resulting from either choice will never arise if the other orbit is chosen instead. For instance, when there is a choice between the orbits $\{x_1, x_2, ..., x_{m-1}\}$ and $\{x_{m+1}, x_{m+2}, ..., x_{2m-1}\}$, we get a constraint $x_1 < x_2$ in the first case, and $x_{m+1} < x_{m+2}$ in the second. Since x_1 and x_{m+1} have the same stabiliser within the stabiliser of x_0, neither variable appears in any subsequent non-singleton orbit along either branch, hence no further constraints involving these variables can arise. Depending on the later choices, some of the constraints added at this point may later become redundant due to transitivity: for instance, if we choose $\{x_1, x_2, ..., x_{m-1}\}$ at this stage and $\{x_2, x_3, ..., x_{m-1}\}$ at the next, the constraints $x_1 < x_3, ..., x_1 < x_{m-1}$ are subsumed by $x_1 < x_2, x_2 < x_3; x_2 < x_4; ..., x_2 < x_{m-1}$. But the first constraint added at each stage will not become redundant, and as already shown, it would never arise if the alternative choice of orbit were made. Hence every set of symmetry-breaking constraints, resulting from choosing between two orbits at each stage of the algorithm, is different in at least one constraint from any other set resulting from making different choices. There are $m - 2$ levels of binary choice between two orbits (the last choice is between $\{x_{m-2}, x_{m-1}\}$ and $\{x_{2m-2}, x_{2m-1})$ and hence 2^{m-2} different sets of symmetry-breaking constraints can be derived. (As already shown, $K_3 \times P_2$ has 2 distinct sets of symmetry-breaking constraints.)

7 Which Constraints To Choose?

If there is more than one set of possible symmetry-breaking constraints, which should we choose? This is not a question that has hitherto been much considered. If users derive symmetry breaking constraints systematically, they are likely to start from a lexicographic order based on some numbering of the variables which reflects in some way the structure of the problem, and find just one set of constraints. It is now clear that there can be many distinct sets of symmetry breaking constraints, and choosing arbitrarily may not lead to the most efficient way of solving the problem at hand.

A further complication is that the symmetry breaking constraints interact with the search strategy, so that neither can be chosen independently. It can be expected that if the variable ordering used during the search for solutions is incompatible with the symmetry-breaking constraints, in some sense, finding solutions can be delayed rather than made more efficient. The aim in adding symmetry-breaking constraints to the CSP is to forbid all but one solution from each symmetry equivalence class; the variable ordering also induces an order on the solutions in each equivalence class. If the only solution allowed by the constraints appears late in the induced order of symmetrically equivalent solutions, then search effort may be wasted in considering partial solutions that could lead to complete solutions to the original problem but are now excluded by the symmetry-breaking constraints. In this way, adding symmetry-breaking constraints could be counter-productive if used with an incompatible variable ordering.

Since the procedure for deriving symmetry-breaking constraints in [Crawford *et al.*, 1996] requires a variable order to be specified (as does the procedure described earlier), it is generally felt that the same order should be used for the search, if a static variable order is to be used, or as a tie-breaker if a dynamic order such a smallest domain is used. In this section, the sets of symmetry-breaking constraints derived earlier for the $K_3 \times P_3$ problem are investigated, and their interaction with a static order. Two questions of interest are: is it possible to decide which symmetry-breaking constraints are best, and is it true that the variable order used to derive the constraints should also be used for the search?

For each of the nine representative sets of symmetry-breaking constraints in this problem, we can find a variable order that would give that set. In the results given below, these variable orders are used as static variable orders during the search for all solutions. The variable orders found are, of course, not unique; here, the lexicographically smallest variable order has been chosen. Choosing a different variable order might give a more efficient search. Nevertheless, the results may give some indications of the answers to these questions.

The sets of constraints as given earlier are:

- A: $x_0 < x_1; x_0 < x_2; x_0 < x_6; x_0 < x_7; x_0 < x_8;$ $x_1 < x_2$
- B: $x_0 < x_1; x_0 < x_2; x_0 < x_6; x_0 < x_7; x_0 < x_8;$ $x_4 < x_5$
- C: $x_0 < x_1; x_0 < x_2; x_0 < x_6; x_0 < x_7; x_0 < x_8;$ $x_7 < x_8$
- D: $x_3 < x_4; x_3 < x_5; x_0 < x_6; x_1 < x_2$
- E: $x_3 < x_4; x_3 < x_5; x_0 < x_6; x_4 < x_5$
- F: $x_3 < x_4; x_3 < x_5; x_0 < x_6; x_7 < x_8$
- G: $x_3 < x_4; x_3 < x_5; x_1 < x_2; x_1 < x_7; x_1 < x_8$
- H: $x_3 < x_4; x_3 < x_5; x_4 < x_5; x_1 < x_7$
- I: $x_3 < x_4; x_3 < x_5; x_4 < x_5; x_2 < x_8$

Variable orders that would lead to the constraints A, for instance, must have x_0 as the first variable, so that we choose the first orbit ($\{x_0, x_1, x_2, x_6, x_7, x_8\}$), at the top level rather than the second ($\{x_3, x_4, x_5\}$), and x_0 rather than x_1, x_2, x_6, x_7 or x_8 from the first orbit. It must then have x_1 before x_2, x_4, x_5, x_7 or x_8, so that we choose the first orbit as the second level, and x_1 rather than x_2. A possible order is x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, but many other orders would give the same constraints, and many more would give equivalent constraints.

Nine possible variable orders, giving the nine sets of symmetry-breaking constraints, have been derived. These have been used as static variable orders in the search for graceful labellings of $K_3 \times P_3$. Tables 1 to 4 show the results (using ILOG Solver 6.0). For space reasons, the variable orders are given as lists of subscripts, so that 0, 1, 2, 3, 4, 5, 6, 7, 8 represents $x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8$. Each variable order is labelled A to I, according to the corresponding set of symmetry breaking constraints. In Tables 1 and 3, both allDifferent constraints have been treated as sets of binary $\neq$ constraints; in Tables 2 and 4, bounds consistency has been maintained on the allDifferent constraints on the edge variables. (For this problem, bounds consistency is almost as effective in pruning variable domains as generalized arc consistency, and is much faster.) The final variable order included in the tables, $x_3, x_4, x_5, x_0, x_1, x_2, x_6, x_7, x_8$, is one that a user might choose for this problem, since it reflects the structure of the graph: the first three variables are the most constrained (before adding symmetry-breaking constraints), and it makes intuitive sense to instantiate all the variables in

	Symmetry-breaking Constraints								
Variable order	A	B	C	D	E	F	G	H	I
0, 1, 2, 3, 4, 5, 6, 7, 8 (A)	3,434	885	2,904	6,202	4,939	4,566	975	836	975
0, 4, 1, 2, 3, 5, 6, 7, 8 (B)	273	273	353	797	377	495	323	873	1,797
0, 7, 1, 2, 3, 4, 5, 6, 8 (C)	2,008	3,999	2,074	7,991	2,827	7,133	169	169	456
3, 0, 1, 2, 4, 5, 6, 7, 8 (D)	41	294	36	294	256	1,296	41	36	1,240
3, 0, 4, 1, 2, 5, 6, 7, 8 (E)	1,006	545	1,809	545	260	1,381	1.006	1,809	1,753
3, 0, 7, 1, 2, 4, 5, 6, 8 (F)	20	1,450	20	1.450	716	2,946	20	20	1,879
3, 1, 0, 2, 4, 5, 6, 7, 8 (G)	42	840	37	141	141	399	314	315	314
3, 4, 1, 0, 2, 5, 6, 7, 8 (H)	3,336	2,087	6,398	34	26	116	34	231	1,328
3, 4, 2, 0, 1, 5, 6, 7, 8 (I)	4,113	2,053	7,105	111	55	138	14	154	1,462
3, 4, 5, 0, 1, 2, 6, 7, 8	5,018	2,989	8,436	1,581	1,493	2,187	726	1,519	3,095

Table 1: Number of backtracks to find a graceful labelling of $K_3 \times P_3$, treating the allDifferent constraint on the edge variables as binary $\neq$ constraints.

	Symmetry-breaking Constraints								
Variable order	A	B	C	D	E	F	G	H	I
0, 1, 2, 3, 4, 5, 6, 7, 8 (A)	12	12	12	41	160	27	27	23	27
0, 4, 1, 2, 3, 5, 6, 7, 8 (B)	75	75	54	323	155	215	117	309	550
0, 7, 1, 2, 3, 4, 5, 6, 8 (C)	65	123	68	2,739	17	2,309	86	86	53
3, 0, 1, 2, 4, 5, 6, 7, 8 (D)	16	134	15	134	112	306	16	15	239
3, 0, 4, 1, 2, 5, 6, 7, 8 (E)	200	69	363	69	66	236	200	363	384
3, 0, 7, 1, 2, 4, 5, 6, 8 (F)	9	560	9	560	362	723	9 9	9	576
3, 1, 0, 2, 4, 5, 6, 7, 8 (G)	42	840	37	48	48	152	75	60	75
3, 4, 1, 0, 2, 5, 6, 7, 8 (H)	733	316	1,339	14	11	51	14	100	484
3, 4, 2, 0, 1, 5, 6, 7, 8 (I)	459	338	1,487	49	26	56	5	66	483
3, 4, 5, 0, 1, 2, 6, 7, 8	252	158	226	39	147	162	116	101	589

Table 2: Number of backtracks to find a graceful labelling of $K_3 \times P_3$, maintaining bounds consistency on the allDifferent constraint on the edge variables.

a clique before moving to another clique. (This variable order would give rise to the symmetry-breaking constraints E, if used for that purpose.)

There are several things that can be noticed from the tables. The most obvious is that search effort does vary with both variable order and symmetry-breaking constraints. However, it is difficult to discern any consistent pattern in any of the tables. Significantly, there is no evidence that a set of symmetry-breaking constraints does relatively better with a variable order that could give rise to those constraints than with another variable order. Tables 1 and 2 have been included because one might expect that the link, if any, between a set of symmetry breaking constraints and the corresponding variable order would be stronger than when finding all solutions, but it is still not apparent.

In Tables 3 and 4, bounds consistency affects the relative ranking of the sets of constraints. Constraints D and G give the best results for several variable orders in Table 3, i.e. for finding all solutions with $\neq$ constraints. However, if bounds consistency is maintained, constraints A and B do much better. Note that the first three sets of constraints force node 0 to have the smallest label. In consequence, it must be labelled 0, but this is not easily discovered from the $\neq$ constraints.

Eight of the nine sets of constraints form three overlapping groups. Sets A, B and C are identical except for the final constraint ($x_1 < x_2$; $x_4 < x_5$; $x_7 < x_8$ respectively). For most of the selected variable orders, A is better than B, which is better than C; A is consistently better than C for all orders in both tables.

Sets D, E, F also match except for the same constraints ($x_1 < x_2$; $x_4 < x_5$; $x_7 < x_8$ respectively). In this case, bounds consistency changes the ranking order: in Table 3, D and E are both better than F, and D is usually better than E; however, with bounds consistency, D becomes worse than the other two, and E and F are broadly similar in performance.

Finally, E, H and I are the same in ordering the node labels of the middle clique, and have an additional constraint ($x_0 < x_6$; $x_1 < x_7$; $x_2 < x_8$ respectively).With $\neq$ constraints, E is consistently better than H and I, and H is usually better than I, except for the orders that have x_7 early, although these are not good orders for any set of constraints. With bounds consistency, I is better than E for all the orders; perhaps in this case, the constraint $x_7 < x_8$ allows useful inferences to be made about the third clique, while node labels in the other cliques are being assigned.

Overall, the picture is quite confusing. It is evidently not possible to choose a best set of symmetry breaking constraints without taking into account the constraint propagation that will act on these constraints (and the others already in the problem). If the allDifferent constraint is treated as a $\neq$ con-

	Symmetry-breaking Constraints								
Variable order	A	B	C	D	E	F	G	H	I
0, 1, 2, 3, 4, 5, 6, 7, 8 (A)	124,694	140,890	207,994	85,478	113,629	136,843	82,323	138,530	148,077
0, 4, 1, 2, 3, 5, 6, 7, 8 (B)	125,103	139,348	210,131	77,245	103,295	127,583	84,370	139,990	149,995
0, 7, 1, 2, 3, 4, 5, 6, 8 (C)	238,390	354,736	336,589	351,703	225,775	401,824	278,133	407,995	297,546
3, 0, 1, 2, 4, 5, 6, 7, 8 (D)	125,220	142,211	209,668	85,880	114,355	137,805	82,494	139,432	148,769
3, 0, 4, 1, 2, 5, 6, 7, 8 (E)	125,308	141,862	213,354	75,465	102,865	128,765	83,288	136,571	151,150
3, 0, 7, 1, 2, 4, 5, 6, 8 (F)	242,143	366,689	348,953	363,602	260,900	424,596	284,224	420,820	300,730
3, 1, 0, 2, 4, 5, 6, 7, 8 (G)	125,206	142180	209,742	86,076	114,552	137,983	82,664	139,660	148,850
3, 4, 1, 0, 2, 5, 6, 7, 8 (H)	124,264	141,597	213,233	75,552	102,639	128,646	83,189	136,748	150,260
3, 4, 2, 0, 1, 5, 6, 7, 8 (I)	137,990	151,185	222,969	84,106	113,097	135,150	87,162	148,109	153,809
3, 4, 5, 0, 1, 2, 6, 7, 8	141,992	140,712	228,760	70,263	97,132	124,893	82,980	121,352	151,333

Table 3: Number of backtracks to find all graceful labellings of $K_3 \times P_3$, treating the allDifferent constraint on the edge variables as binary $\neq$ constraints.

	Symmetry-breaking Constraints								
Variable order	A	B	C	D	E	F	G	H	I
0, 1, 2, 3, 4, 5, 6, 7, 8 (A)	13,026	14,477	19,947	32,609	32,145	32,262	30,238	47,694	21,582
0, 4, 1, 2, 3, 5, 6, 7, 8 (B)	16,083	14,266	25,739	30,908	29,465	27,398	31,789	47, 458	23,436
0, 7, 1, 2, 3, 4, 5, 6, 8 (C)	41,913	45,394	60,349	78,571	54,644	65,372	61,039	68,409	36,745
3, 0, 1, 2, 4, 5, 6, 7, 8 (D)	12,803	14,045	19,530	33,797	32,465	32,702	30,938	48,923	21,427
3, 0, 4, 1, 2, 5, 6, 7, 8 (E)	14,382	12,953	21,556	33,904	31,222	30,938	33,964	50,975	22,167
3, 0, 7, 1, 2, 4, 5, 6, 8 (F)	38,488	25,199	56,735	83,383	56,735	64,113	62,637	58,455	37,025
3, 1, 0, 2, 4, 5, 6, 7, 8 (G)	12,838	14,235	19,608	33,707	32,547	32,764	31,004	48,959	21,268
3, 4, 1, 0, 2, 5, 6, 7, 8 (H)	15,834	13,482	22,155	33,078	32,217	30,367	33,729	50,372	21,989
3, 4, 2, 0, 1, 5, 6, 7, 8 (I)	12,375	13,799	19,618	37,231	34,895	33,426	35,283	58,496	27,991
3, 4, 5, 0, 1, 2, 6, 7, 8	10,552	14,086	21,501	31,569	28,741	25,943	31,478	36,322	20,880

Table 4: Number of backtracks to find all graceful labellings of $K_3 \times P_3$, maintaining bounds consistency on the allDifferent constraint on the edge variables.

straint, sets D and G do well (for these variable orders), which suggests that heterogeneous constraints, that involve different parts of the problem, are a good choice. On the other hand, if bounds consistency is maintained, constraints A and B do better. We might conclude that constraints that allow strong conclusions to be drawn (e.g. that node 0 must be labelled 0) are a good choice, provided that the level of propagation does allow the conclusion.

The tables also allow some comparison of variable orders for this problem (although note that the variable orders presented here are relatively good ones; many orders that are much worse than any of the orders in the tables can be found). The best results overall, for finding all solutions, are from the final variable order, $x_3, x_4, x_5, x_0, x_1, x_2, x_6, x_7, x_8$. This was also the best performance found, out of many other variable orders not shown in the tables, although a systematic investigation has not been practicable. Again, this variable order gives best results with constraints D if bounds consistency is maintained; otherwise, constraints A are best. This variable order is, however, rather poor for finding just one solution. Similarly, of the variable orders selected for the tables, C and F are clearly much worse than others, at least for finding all solutions. These both assign a value to x_7 early in the search, immediately after x_0 and before x_1. Since x_7 is not directly connected to x_0 and x_1, this order could be expected to lead to wasted search. Orders C and F do not do better with constraints C and F respectively than they do otherwise, so again there is little evidence here that a set of symmetry breaking constraints should be used with a variable order that would give rise to it. However, the link with the variable ordering seems far from straightforward, at least in this problem, and warrants further investigation.

8 Conclusions

The procedure described in [Puget, 2004] for devising symmetry breaking constraints when the symmetry acts only on the variables and there is an allDifferent constraint on the variables, has been extended to find all distinct sets of symmetry-breaking constraints. It is shown, using graceful graphs problems, that there can be exponentially many such sets. This leads to the need to choose between them. The choice is complicated by the fact that the search performance resulting from choosing a set of such constraints is affected by the search strategy. It has been recognised that if there is a conflict between the symmetry-breaking constraints and the variable order, then the search effort may increase rather than decrease as a result of symmetry breaking. A plausible assumption is that, since the choice of symmetry-breaking constraints requires choosing one variable rather than another, the same choices should be reflected in the variable order.

Unfortunately, experiments with finding graceful labellings of $K_3 \times P_3$ do not support this assumption; the results do not show a reduction in search effort from pairing a set of symmetry breaking constraints with a compatible variable order in this way. The best choice of symmetry breaking constraints for this problem depends on the level of constraint propagation that will be maintained on all the constraints during search, and also on the variable order. The results suggest that heterogeneous constraints involving different parts of the problem are best if the allDifferent constraint is treated as $\neq$ constraints, whereas if bounds consistency is maintained on this constraint, a set of constraints that will allow constraint propagation to set the value of one of the variables is better.

In some problems, although not the graceful graphs problems considered here, symmetry-breaking constraints allow the derivation of implied constraints that can further reduce search. Frisch, Jefferson and Miguel [Frisch *et al.*, 2004] suggest that when there is a choice between distinct sets of symmetry-breaking constraints, the choice could be guided by considering the implied constraints and their potential effect on the search.

The experiments reported here use static search variable orders. Investigating the interaction between dynamic variable ordering heuristics such as smallest domain and symmetry-breaking constraints would be more complicated, and probably even more confusing. For graceful graphs problems, smallest domain ordering has sometimes proved to be significantly worse than a static ordering. Nevertheless, there may be problem classes where we want to combine symmetry-breaking constraints with dynamic variable ordering. A possibility is to choose the symmetry-breaking constraints during search: the first variable, say x_0, can be determined before the search starts, its orbit found and the appropriate symmetry-breaking constraints imposed; after making an assignment to x_0, the heuristic chooses the second variable, say x_1, and symmetry breaking constraints are again imposed, and so on [1].

However, even if we confine ourselves to static variable orders, further investigation is needed into the interaction between variable order and symmetry-breaking constraints and how to choose between different sets of symmetry-breaking constraints.

References

[Crawford *et al.*, 1996] J. Crawford, M. Ginsberg, E. Luks, and A. Roy. Symmetry-Breaking Predicates for Search Problems. In *Proceedings KR'96*, pages 149–159, November 1996.

[Frisch *et al.*, 2004] A.M. Frisch, C. Jefferson, and I. Miguel. Symmetry-breaking as a Prelude to Implied Constraints: A Constraint Modelling Pattern. In *Proceedings of ECAI 2004*, pages 171–175, 2004.

[Gallian, 2003] J. A. Gallian. A Dynamic Survey of Graph Labeling. *The Electronic Journal of Combinatorics*, (DS6), 2003.

[Petrie and Smith, 2003] K. E. Petrie and B. M. Smith. Symmetry Breaking in Graceful Graphs. Technical Report APES-56-2003, APES Research Group, January 2003.

[Puget, 2004] J.-F. Puget. Breaking symmetries in all different problems. In *Proceedings of SymCon04, the 4th International Workshop on Symmetry in Constraints*, 2004.

↳ IJCAI'05

[1]This was suggested by one of the reviewers.