Automatic Generation of Constraints for Partial Symmetry Breaking

Christopher Jefferson[1] and Karen E. Petrie[2]

[1] Computer Science, University of St Andrews, UK
caj@cs.st-andrews.ac.uk
[2] School of Computing, University of Dundee, UK
karenpetrie@computing.dundee.ac.uk

Abstract. Constraint Satisfaction Problems (CSPs) are often highly symmetric. Symmetries can give rise to redundant search, since subtrees may be explored which are symmetric to subtrees already explored. To avoid this redundant search, constraint programmers have designed methods, which try to exclude all but one in each equivalence class of solutions. One problem with many of the symmetry breaking methods that eliminate all the symmetry is that they can have a large running overhead. To counter this flaw many CP practitioners have looked for methods that only eliminate a subset of the symmetries, so called partial symmetry breaking methods, but do so in an efficient manner. Partial symmetry breaking methods often work only when the problem is of a certain type. In this paper, we introduce a new method of finding a small set of constraints which provide very efficient partial symmetry breaking. This method works with all problem classes and modelling techniques.

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.

Symmetries can give rise to redundant search, since subtrees may be explored which are symmetric to subtrees already explored. To avoid this redundant search, constraint programmers have designed methods which try to exclude symmetrically equivalent solutions.

In recent years CSP practitioners have created ways to automatically detect symmetry [1,2]. In this paper we consider how a constraint solving system such as Minion [3] should automatically exclude this detected symmetry. In particular, we wish to use a method that will increase the efficiency of search by effectively excluding part of the search space, without incurring an overhead. This will provide an efficient method for dealing with the occurrence of symmetry, without requiring user input.

J. Lee (Ed.): CP 2011, LNCS 6876, pp. 729–743, 2011.
© Springer-Verlag Berlin Heidelberg 2011

The symmetry breaking method considered here is the addition of constraints to the CSP, which exclude some or all symmetric equivalents. The focus of this paper is to find a small subset of symmetry breaking constraints, the placing of which has negligible effect on the solver; yet that efficiently excludes a large proportion of the symmetrically equivalent search space.

In the next section of the paper we discuss the previous work in the area of symmetry exclusion and in particular partial symmetry breaking. We then explain how our method choosing the set of symmetry breaking constraints to be placed. The final section of this paper provide detailed benchmarking for possible sets of symmetry breaking constraints, across multiple problems.

2 Background

Methods to eliminate symmetry in CSP fall into two broad categories: dynamic and static. Dynamic symmetry breaking methods eliminate symmetry during search. Static symmetry breaking methods eliminate symmetry before search commences. Both of these methods can add a large efficiency overhead to solving a CSP. To counter this CP practitioners have tried partial methods which eliminate only a subset of the symmetry. Partial methods have been experimented with in both the dynamic and static contexts. We will look at each of these in turn.

Symmetry Breaking During Search [4,5] is a dynamic symmetry elimination method which adds constraints on backtracking. McDonald and Smith [6] considered using a subset of the full symmetry functions which would be required for complete symmetry breaking, with SBDS to provide partial symmetry breaking. They proved that there is indeed a cross over point where by using an increased number of symmetry functions would not create the efficiency of solving the problem. The problem with dynamic symmetry breaking methods in general is that the framework to use them must be available in your solver, many modern solvers such as Gecode and Minion [3,7] do not have this framework.

Static symmetry breaking methods do not require solver support. The most common static symmetry breaking method is to add constraints to the CSP. Crawford et al. [8] give a systematic method for generating symmetry breaking constraints. Definition 1 explains Crawford ordering constraints.

Definition 1. *For a variable symmetry group of size s the Crawford ordering method produces a set of $s-1$ lex constraints that provide complete symmetry breaking. We first decide on a canonical order for the variables in the CSP, then post constraints such that this ordering is less than or equal to the permutation of the ordering by each of the symmetries. Consider the following 2×3 matrix with the symmetries that the rows and columns can be swapped independently.*

$$\begin{matrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \end{matrix}$$

If we choose a row-wise canonical variable ordering, in this case $x_{11}x_{12}x_{13}x_{21}x_{22}x_{23}$*, then we can generate* 11 *lex constraints to break all the symmetries. For example the constraint generated for the permutation which swaps the rows of the matrix is:*

$$x_{11}x_{12}x_{13}x_{21}x_{22}x_{23} \leq_{\text{lex}} x_{21}x_{22}x_{23}x_{11}x_{12}x_{13}$$

The flaw with this complete symmetry breaking method is that if a problem has many symmetries than a large number of symmetry breaking constraints will be needed to eliminate all the symmetry. To counter this problem CP practitioners have tried placing just a small subset of these constraints. The question then arises as to which subset should be placed? This is the question we tackle in this paper.

Puget and Smith [9,10] have previously looked at this question and shown that a small subset of constraints can provide complete symmetry breaking when there is an alldifferent constraint across all the problem variables.

The constraints presented by Puget and Smith can also be simply used to create a partial symmetry breaking method on problems without alldifferent constraints, as demonstrated in Example 1. We will consider these constraints later in this paper.

Example 1. The algorithm of Puget in [9] would generate for the example in Definition 1 the set of constraints:

$$x_{11} < x_{12}, \quad x_{11} < x_{21}, \quad x_{11} < x_{22}, \quad x_{11} < x_{23}, \quad x_{12} < x_{13}$$

These constraints are generating by truncating each Crawford ordering constraint at the first index where the variables are different. These can be transformed into a partial set of symmetry breaking constraints for any CSP by weakening them to:

$$x_{11} \leq x_{12}, \quad x_{11} \leq x_{21}, \quad x_{11} \leq x_{22}, \quad x_{11} \leq x_{23}, \quad x_{12} \leq x_{13}$$

It has also been proposed in SAT and CP that using just the group generators provided by Nauty [11], after automatic symmetry detection, provide an efficient set of symmetry breaking constraints [12,13,14]. Our method often improves on the performance given by the group generators produced by Nauty. While there has been work into automatically finding the symmetries of CP problems [1,2], so far the only general symmetry breaking method is to use the generators returned by the symmetry detection method into Crawford ordering constraints.

A discussion of partial symmetry breaking would not be complete without mention of Double Lex [15,16]. Double Lex is a method of placing lexicographic constraints to provide partial symmetry breaking on matrix models of CSPs. Double Lex is a very widely used symmetry breaking method. However, it has the limitation of only working when the problem is modelled as a matrix.

3 Overview of Group Theory Required for Our Method

In this section we explain the group theory required to understand how we choose a subset of constraints. Group theory is the mathematical study of symmetry.

Stabiliser chains provide an algorithmic method of constructing a small generating set [17,18] for any group and provide the inspiration for our algorithm. The stabiliser chain relies on the concept of the point wise stabiliser. We start by giving the definition of a stabiliser.

Definition 2. *Let G be a permutation group acting on the set of points Ω. Let $\beta \in \Omega$ be any point. The* stabiliser *of β is the subgroup of G defined by: $Stab_G(\beta) = \{g \in G \mid \beta^g = \beta\}$, which is the set of elements in G which fixes or* stabilises *the point β. The stabiliser of any point in a group G is a subgroup of G. The stabiliser of a set of points, denoted $Stab_G(i, j, \ldots)$, is the elements of G which move none of the points.*

The definition of the stabiliser chain follows.

Definition 3. *Stabiliser chains are built in an recursive fashion. Given a permutation group G and a point i, the first level of the stabiliser chain is built from an element of G which represents each of the places i can be mapped to. The next level of the stabiliser chain is built from applying this same algorithm to $Stab_G(i)$, again choosing representative elements for all the places some point $j \neq i$ can be mapped to. The stabiliser chain is finished when the stabiliser generated contains only the identity element.*

Stabiliser chains, in general, collapse quickly to the subgroup containing only the identity since the order of each new stabiliser must divide the order of the stabilisers above it. The following example shows the construction of a stabiliser chain.

Example 2. Consider the group consisting of all 24 permutations of $\{1, 2, 3, 4\}$. We compute a chain of stabilisers of each point, starting arbitrarily with 1 (denoted $Stab_{S_4}(1)$). 1 can be mapped to 2 by $[2, 1, 3, 4]$, 3 by $[3, 1, 2, 4]$ and 4 by $[4, 1, 2, 3]$. These group elements form the first level of the stabiliser chain.

The second level is generated by looking at the orbit and stabiliser of 2 in $Stab_{S_4}(1)$. In the stabiliser of 1, 2 can be mapped to both 3 and 4 by the group elements $[1, 3, 2, 4]$ and $[1, 4, 2, 3]$. We now stabilise both 1 and 2, leaving only the group elements $[1, 2, 3, 4]$ and $[1, 2, 4, 3]$. Here 3 can be mapped to 4 by the second group element, and once 1, 2 and 3 are all stabilised the only element left is the identity and the algorithm finishes.

4 Methods of Creating Subset of Symmetry Breaking Constraints

In this section we will investigate methods of generating both partial and complete sets of symmetry breaking constraints. While in general complete symmetry breaking is NP-complete, for certain groups it is polynomial and by studying these groups we hope to derive general principles. We begin by proving a simple result about Crawford ordering constraints. This result will be used in the following sections.

Definition 4. *Given a permutation p on the ordered set $S = \{x_1, \ldots, x_n\}$ other than the identity permutation, the first moved point of p is the smallest i such that $p(x_i) \neq x_i$.*

Lemma 1. *Consider any permutation p on the ordered set $V = \{x_1, \ldots, x_n\}$ which is not the identity permutation, then the Crawford ordering constraint generated by p is logically equivalent to $x_i \leq p(x_i) \wedge (x_i = p(x_i) \rightarrow C)$, where x_i is the first moved point of p and C is some constraint on the elements of V.*

Proof. The Crawford ordering constraint generated from p is the constraint $[x_1, \ldots, x_n] \leq_{lex} [p(x_1), \ldots, p(x_n)]$. Up until the first moved point of p the variables in the two arrays are identical, so have no effect. At position i, the first moved point, the theorem follows from the definition of lexicographic ordering constraints, with C equal to the constraint $[x_{i+1}, \ldots, x_n] \leq_{lex} [p(x_{i+1}), \ldots, p(n)]$. □

While Lemma 1 follows fairly directly from the definitions of the Crawford ordering and lexicographic ordering constraints, it is useful when analysing subsets of Crawford ordering constraints. In the next section we will show how important the first moved point is in generating good sets of partial symmetry breaking constraints.

4.1 Analysing the Complete Symmetry Group

The complete, or symmetric, group is the group which contains all permutations on some set. There are many problems which have the complete group of variable symmetries. There are already known methods of achieving complete symmetry breaking for the complete group. In this section we will produce a dichotomy which tells us all subsets of permutations which lead to sets of Crawford ordering constraints which break all symmetries.

Theorem 1. *Given a subset S of the complete symmetry group G on the set $\{x_1, \ldots, x_n\}$, the Crawford ordering constraints generated by S with the ordering $x_1, \ldots, x_n$ will be complete symmetry breaking constraints for G if and only if:*

$$\forall i \in \{1, \ldots, n-1\}.\ \exists p \in S.\ (p(x_i) = x_{i+1} \wedge \forall j \in \{1, \ldots, i-1\}.\ p(x_j) = x_j)$$

Proof. We will perform this proof in two parts. Firstly by Lemma 1 a permutation where $p(x_i) = x_{i+1} \wedge \forall j \in \{1, \ldots, i-1\}.\ p(x_j) = x_j$ implies $x_i \leq x_{i+1}$. Therefore these constraints together imply that any assignment to the x_i must be non-decreasing, which leads to complete symmetry breaking.

Now consider any permutation q where for some fixed c:

$$q(x_c) = x_{c+1} \wedge (\forall j \in \{1, \ldots, c-1\}.\ q(x_j) = x_j)$$

does not hold. Then by Lemma 1, the Crawford ordering constraint generated by this permutation is equivalent to $x_r \leq x_s \wedge (x_r = x_s \rightarrow C)$, where either $r \neq c$ or $s \neq c + 1$.

Consider the assignment where $x_i = i$ for all i, and the assignment where $x_i = i$ for all i, except x_c is assigned $c+1$ and x_{c+1} is assigned c. Both of these assignments are accepted by the Crawford ordering constraint generated from q, as both satisfy the constraint $x_r < x_s$ for all $r < s$, except in the case $r = c, s = c+1$.

This implies that if there is any c where no permutation in S satisfies $q(x_c) = x_{c+1} \wedge \forall j \in \{1, \ldots, c-1\}.\ q(x_j) = x_j$, then the Crawford ordering constraints generated from S cannot break all symmetry. □

Theorem 1 describes all sets of permutations which lead to a complete set of symmetry breaking constraints for the symmetric group. We can deduce some interesting properties from the requirements of Theorem 1.

Theorem 1 requires permutations which fix the first i variables of the ordering used for the Crawford ordering. Such permutations are generated by stabiliser-chain based algorithms, as these begin by looking for permutations which fix as many points as possible, in the order given to the algorithm. Such algorithms are unlikely to arise when chose at random. In particular, any set of permutations whose Crawford ordering constraints break all symmetry must include the permutation which fixes all but the last two variables.

While in general we cannot find a polynomial-sized set of permutations which break all symmetry, Theorem 1 suggests we should investigate sets which contain permutations which fix many initial points, even when these form a very small part of the full group.

4.2 Generating Stabiliser Chains

We introduced the concept of a stabiliser chain in Section 3. Nauty [11] and other graph-theoretic systems generate a stabiliser chain for a group. Permutations in a stabiliser chain for a group, form a set of generators for that group. Stabiliser chains also have many other useful properties. In this section we shall analyse the Crawford ordering constraints the permutations from stabiliser chains generate.

Algorithm 1 shows a very basic outline of an algorithm for finding stabiliser chains (in . Example 2 shows this algorithm in practice with no optimisations. Algorithms similar to this basic form are used by Nauty, GAP and other group-theoretic systems. Most of the complication of the algorithm occurs in line 5, where the majority of the work is done.

Algorithm 1. Generate Stabiliser Chain: sc(G)

Require: A group G defined over the points $[x_1, \ldots, x_n]$
1: Initialize *Gens*: Stabiliser chain for G
2: **for all** i in $[n, \ldots, 1]$ **do**
3:    **for all** j in $[n, \ldots, i+1]$ **do**
4:       **if** Optimisation Check Fails **then**
5:          **if** $\exists g \in G.\ (\forall k \in \{1, \ldots, i-1\}.\ g(x_k) = x_k) \wedge g(x_i) = x_j$ **then**
6:             Record g as the permutation mapping x_i to x_j
7: **return** *Gens*

The interesting part for this paper is the optimisation function used to skip parts of search when some permutations have been found. These algorithms ensure that we find a complete set of generators for the group, while skipping parts of the search space.

In this paper we will consider two possible optimisation conditions, given in Definition 5. The **Nauty** optimisation condition is one of the techniques used by Nauty, Saucy and GAP. The **reduced** optimisation condition is the one we will be most interested in here. As the **reduced** optimisation condition is logically weaker than the **Nauty** condition, we do not have to prove that it is valid to use it to reduce search when searching for stabiliser chains. It should be noted that the set of constraints produced by the **Nauty** condition is always a subset of the subset of constraints obtained from the **Reduced** condition.

Definition 5.
*The **Nauty** optimisation condition for line 4 of Algorithm 1 is:*

$$\exists k.\ i \leq k < j \text{ and a permutation mapping } x_k \text{ to } x_j \text{ was already found}$$

*The **Reduced** optimisation condition is:*

$$\exists k.\ i < k < j \text{ and a permutation mapping } x_k \text{ to } x_j \text{ was already found}$$

The reduced optimisation condition will produce a larger set of generators. These generators have the property that the Crawford ordering constraints generated from these permutations subsume the binary $\leq$ constraints generated by Puget's algorithm, as given in Example 1.

Theorem 2. *Given a set of generators S for a group G on an ordered list $[x_1, \ldots, x_n]$ generated by Algorithm 1 with the **reduced** optimisation condition, the Crawford ordering constraints generated from S under the ordering $[x_1, \ldots, x_n]$ will imply $x_i \leq x_j$ for every pair of variables if any of the full set of symmetry breaking constraints for G imply that constraint.*

Proof. If the Crawford ordering constraint for some $g \in G$ implies $x_i \leq x_j$, then by Lemma 1, the permutation g must have smallest moved point x_i, which is moved to x_j. We shall prove that all such inequalities are implied by the Crawford ordering constraint generated from S by contradiction.

If some inequalities are not implied by the Crawford ordering constraints generated from S, consider some permutation g where the inequality $x_i \leq x_j$ generated by g has the smallest possible value for $j - i$.

Now, as $g \in G$, then the algorithm would find g when looking for a permutation which mapped x_i to x_j while fixing all the x_z for $z < i$, unless this part of search was skipped over by the **reduced** optimisation condition. However in this case there must exist some k with $i < k < j$ such that a permutation mapping x_k to x_j was already found. In this situation there are two cases to consider:

1. S contains a permutation mapping x_i to x_k which fixes all $x_z, z < i$. In this case the Crawford ordering constraint generated by this permutation implies $x_i \leq x_k$ and the permutation mapping x_k to x_j which caused the **reduced** optimisation condition to trigger implies the constraint $x_k \leq x_j$. These constraints together imply $x_i \leq x_j$.
2. S does not contain a permutation mapping x_i to x_k which fixes all $x_z, z < i$. Such permutations certainly exist in G, by applying the permutation which maps x_i to x_j, and the inverse of the permutation which maps x_k to x_j. Therefore permutations of this kind must have been skipped by the **reduced** optimisation condition. Then either there exists some permutation in G which implies $x_i \leq x_k$ in which case we are done, or there does not, in which case as $k - i < j - i$ our assumption that x_i and x_j were the pair where the inequality $x_i \leq x_j$ was not generated and $j - i$ was minimised is false. □

As we know that the binary inequalities break all symmetry in the presence of all different constraints, intuitively we might expect them to do well when breaking symmetries in general. Later in our experimental section we shall test this hypothesis.

5 Experimental Results

In this section we will compare a number of methods of generating both partial and complete sets of symmetry breaking constraints, to compare their effectiveness. In each case we automatically detect the symmetry of the problem using a variant of the algorithm given in [19] and Nauty. We consider 5 different methods:

Nauty: Crawford ordering constraints created from the set of permutations generated by Nauty.
ArityOne: $\leq$ constraints derived from Puget's algorithm, as in Example 1.
All: Crawford ordering constraints created from all symmetries.
Reduced: Crawford ordering constraints generated from our new algorithm, described as the reduced optimisation condition in Definition 5
Basic Stabiliser: Crawford ordering constraints generated by running Algorithm 1 with no optimisation condition.

In order to fit our tables into a compressed space, we use a short-hand to label our results tables:

C: The number of constraints added.
T1: The time taken to find one solution.
N1: The nodes taken to find one solution.
TA: The time taken to find all solutions.
NA: The nodes taken to find all solutions.
S: The total number of solutions.

Due to space limitations, in some tables we use scientific notation to represent large numbers. For example, 1.3E8 is equal to 1.3×10^8.

5.1 Unconstrained Cycles

Firstly, we consider symmetry breaking in two closely related problems, the symmetric cycle and the non-symmetric cycle.

The symmetric cycle is given to Nauty as a graph on the n variables of the problem, with an edge between x_i and x_{i+1} for all i, as well as an edge between the first and last variables x_1 and x_n. This problem has $2n$ symmetries, as the variables can be rotated and also flipped, by mapping x_i to x_{n-i+1} for all i. We also consider the symmetric problem without the flip, by giving Nauty directed rather than undirected edges. There are no constraints in this problem other than the symmetry breaking constraints produced, meaning that it gives us a testbed for our method without having to worry about how the symmetry breaking constraints are interacting with the problem constraints.

The results are given in Table 1. In this problem we see the efficiency of **Nauty** in terms of the size of the generating sets it creates, with only 2 generators for the symmetric cycle and 1 generator for the non-symmetric cycle. However, these very small sets of constraints do not make good sets of symmetry breaking constraints. Our **reduced** algorithm produces 10 constraints which provide smaller times than both **Nauty** and **ArityOne** consistently.

In this problem, as in all the others, we found that the **Basic Stabiliser** method generated exactly the same sized search as the **Reduced** method, while taking longer and producing more constraints. Therefore we omit it.

Table 1. Solving the unconstrained cyclic problem. All node and solution counts given in millions. Problems specified as: "variable count . domain size". N = Not Symmetric cycle.

	ArityOne				Nauty				All				Reduced			
	C	TA	NA	S	C	TA	NA	S	C	TA	NA	S	C	TA	NA	S
10.4	9	0.6	2.6E6	1.3E6	2	1.1	6.9E6	3.4E6	19	0.3	1.1E6	0.4E6	10	0.4	1.4E6	0.6E6
11.4	10	2.7	13E6	6.5E6	2	5.4	34E6	17E6	21	1.3	4.9E6	2.2E6	10	1.8	8.0E6	3.7E6
12.4	11	13	64E6	32E6	2	25	164E6	82E6	23	6.2	22E6	10E6	11	7.4	31E6	14E6
13.4	12	62	31E7	15E7	2	127	82E7	41E7	25	31	10E7	4.6E7	12	38	17E7	8.1E7
11.5	10	16	83E6	41E6	2	39	252E6	125E6	21	9.0	36E6	16E6	10	11	54E6	25E6
11.5N	10	26	142E6	71E6	1	54	362E6	181E6	10	15	73E6	32E6	10	15	73E6	32E6

5.2 Graceful Graphs

Secondly we consider symmetry breaking in the Graceful Graphs problem [20]. 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 (and form a permutation of $\{1, 2, ..., q\}$). This problem has both variable and value symmetry, in this paper we will consider only the variable symmetry.

The CP model we use has variables for each node of the graph and each edge. Assigning just the node variables is sufficient to break all symmetry, and these

Table 2. Solving instances of the Graceful Graphs problem. The last 4 methods all share a node and solution count.

Graph	Nauty				Complete Methods									
					ArityOne		All		Basic Stab		Reduced			
	C	TA	NA	S	C	TA	C	TA	C	TA	C	TA	NA	S
DW4	5	7.3	1.6E6	196	7	3.2	127	3.63	12	3.5	7	3.3	767,613	88
DW5	5	2700	6.9E8	9112	9	856	199	971	15	864	9	856	2.0E8	2432
K4xP2	3	2.7	739,461	572	7	0.2	47	0.2	10	0.2	7	0.2	62,473	30
K5xP2	20	194	4.7E7	20	9	18.2	239	22.16	15	18.1	15	18.1	5.0E6	2

variables must be all different. Therefore we know that the **ArityOne** constraints are sufficient to break all symmetry. In fact, we find that the **ArityOne**, **All**, **Basic Stabiliser** and **Reduced** problems all generate an identically sized search tree. Therefore in Table 2, we give the solution and node count for each of these problems only once.

We use double-wheel and KnxP2 instances of the Graceful Graphs problem as described in [20]. In Table 2 we see that **Reduced** and **ArityOne** are comparable. This is somewhat surprising as **ArityOne** provides complete symmetry breaking with less constraints then **Reduced**. The **Nauty** constraints perform quite poorly on this problem.

5.3 BIBD

The balanced incomplete block design problem is a commonly used problem to study symmetry breaking in constraint programming. A *balanced incomplete block design* (BIBD) [21] is a $v \times b$ Boolean matrix, with the columns summing to k, the rows summing to r, and exactly λ positions where two rows both have a 1, for any pair of rows.

Given a (non-) solution to a BIBD, it is possible to freely permute all of the rows of the matrix to get other (non-) solutions, and it is also possible to freely reorder the rows. Thus this problem has *row and column symmetry.* On this problem, it was not possible to generate the Crawford ordering constraints for the whole symmetry group, and the results for the ArityOne constraints timed out.

Our results for the BIBD are given in Table 3. In this particular problem, we note that all methods generate the same sized search space. Section 5.4 shows that this is not a characteristic of all problems involving row and column symmetry, even on domain size 2. As the **reduced** method generates more constraints, it is slightly slower on this problem.

It is interesting to note that the Crawford ordering constraints generated by Nauty for this, and other, problems with a two dimensional matrix with symmetries of the rows and columns is exactly the Double Lex set of constraints from [15].

Table 3. Solving BIBD models with various symmetry breaking methods

				Nauty			Basic Stabiliser			Reduced		
Problem	N1	NA	S	C	T1	TA	C	T1	TA	C	T1	TA
11,11,5,5,2	66	110	1	21	0.03	0.04	210	0.23	0.23	120	0.18	0.17
13,13,4,1,1	105	815	8	24	0.06	0.06	300	0.57	0.55	168	0.38	0.40
16,16,6,6,2	323	78,842	252	30	0.12	0.82	465	1.73	2.48	255	1.03	1.91
7,35,15,3,5	341	600,598	64,601	40	0.04	3.21	820	0.90	4.87	244	0.40	3.97
8,28,14,4,6	2955	1.8E7	2.0E6	34	0.05	123	595	0.73	137	223	0.42	132
7,49,21,3,7	778	3.2E7	2.2E6	54	0.06	194	1485	2.15	274	342	0.80	215
7,56,24,3,8	1107	1.7E8	1.0E7	61	0.08	1079	1891	3.02	1659	391	1.04	1231

5.4 Plain Row and Column Symmetry

To further investigate row and column symmetries, we tested finding all solutions to a problem with no constraints, placing only symmetry breaking constraints for row and column symmetries.

Table 4 summarises the results for this problem. Here we can see that the **Reduced** method produces both a smaller search and faster time than either the **Nauty** or **ArityOne** method. While generating **All** symmetry breaking constraints produces a smaller search, the time taken to solver the problem is much longer. As in the BIBD problem, the constraints generated by the **Nauty** method is the set of constraints commonly referred to as Double Lex. So on this problem the **Reduced** method outperforms Double Lex.

Table 4. Solving the unconstrained problem with row and column symmetries. Time limit one hour.

	ArityOne				All				Nauty				Reduced			
	C	TA	NA	S	C	TA	NA	S	C	TA	NA	S	C	TA	NA	S
2x5 D8	9	5.5	31E6	15E6	239	6.2	10E6	5E6	5	3.8	22E6	11E6	9	2.3	12E6	6.1E6
2x5 D10	9	40.6	23E7	11E7	239	51	9.3E7	4.5E7	5	30	18E7	9.4E7	9	19.7	10E7	5.2E7
3x3 D10	8	18	10E7	5.1E7	35	13	5.6E7	2.7E7	4	15	10E7	5.0E7	8	12	6.8E7	3.4E7
3x5 D4	14	9.0	47E6	23E6	719	6.3	3.5E6	1.7E6	6	1.4	8.4E6	4.2E6	14	1.3	6.1E6	3.0E6
4x4 D4	15	41	214E6	107E6	575	22	16E6	7.8E6	6	13	84E6	42E6	15	10	49E6	24E6
5x5 D2	24	0.8	3.2E7	1.6E7	14,399	1.2	11,269	5,624	8	0.05	162,567	81,284	24	0.03	38,459	19,230
5x5 D3	24	-	-	-	14,399	-	-	-	8	455	2.7E9	1.3E9	24	106	4.5E8	2.2E8

5.5 Randomly Generated Sets

So far we have compared various variants of stabiliser chains, and with the exception of the **Basic Stabiliser** method, we have found methods which generate more constraints produce smaller search trees while taking longer per node. In our experiments, the **Reduced** method does consistently well. In this section we will investigate two hypothesises by the use of randomly generated sets of permutations.

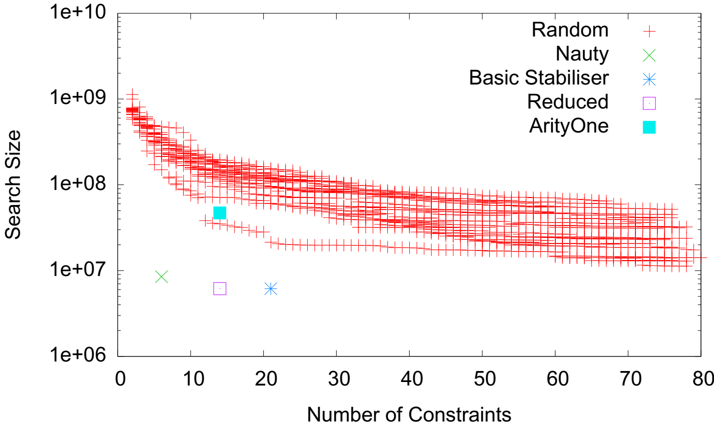

Fig. 1. Plot of solving a 3 by 5 matrix of variables with domain size 4

Firstly, we test the hypothesis that the sets of permutations we are using are better than a randomly generated set of the same size. In particular, we are not simply gaining by increasing the size of the set of permutations we use. Secondly, we test if in general a set of permutations being a generating set improves their effectiveness at symmetry breaking.

Arbitrary Random Sets. We consider one problem in depth, a 3 by 5 matrix of variables with domain size 4 and the Graceful graph problem of the "double wheel" of size 4. Figures 1 and 2 show how search size varies with the number of random constraints, compared to the specialised algorithms we have considered. These graphs clearly show how much our algorithms out-perform a random set of constraints. This shows that the **Reduced** method of generating constraints,is far better than just picking the same number of constraints at random.

Random sets of Generators. Set of generators are used to express groups compactly, and therefore it makes intuitive sense that they may make good sets of permutations to generate Crawford ordering constraints. We will investigate this intuition in this section.

There are two classes of generators we can consider, minimal and non-minimal. Non-minimal sets of generators are any set of permutations which generate the group in question. A minimal set of generators is the smallest number of generators needed to generate a group.

Random sets of constraints are very likely to be sets of generators for most groups. For example, in the 3 by 5 matrix problem given in Section 5.5, by experimentation we find over 91% of random sets of permutations of size 6 and 98% of random sets of permutations of size 8 are generating sets. This means that Figures 1 and 2 can be also be used as a comparison of random sets of generators, as well as random sets of permutations. Therefore the same results as Section 5.5 applies to generators – random sets of generators do not perform well as symmetry breaking constraints.

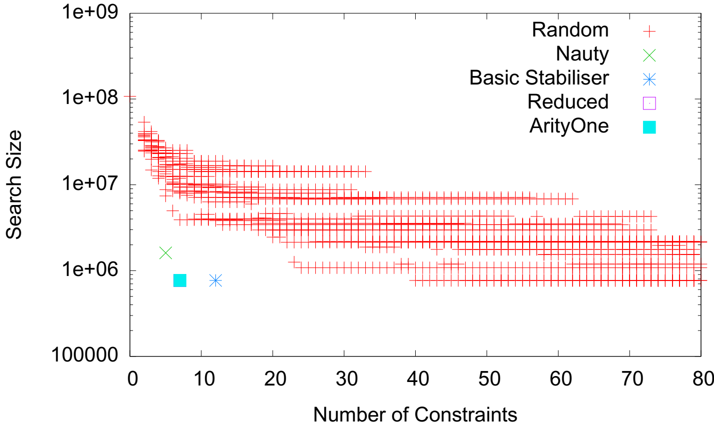

Fig. 2. Plot of solving the DW4 Graceful Graph instance

We shall consider in this section the issue of minimal sets of generators. In particular, we will study if a minimal set of generators produces better Crawford ordering constraints than an arbitrary set of permutations of the same size.

For each problem in Table 5, we created one thousand random sets of minimal generators. Then we created one thousand sets of random permutations of the same size as the minimal generating set. We then found all solutions for each of these pairs and compared how often an arbitrary random set won over a minimal generating set.

Table 5 shows the average and minimum performances, and also compares on each of the thousand runs which of the two methods was fastest. We can see there is no benefit to considering a set of random generators over an arbitrary random set of permutations. This table also shows the sizes these minimal generating sets took. We can see the range of sizes is very small, even when generating a thousand generating sets. Further, these algorithms performed extremely poorly when compared to the **Reduced** algorithm described in this paper. The experiments in Table 5 are a BIBD, a problem with no constraints & row and column symmetry and anna.col, a graph colouring problem from the Stanford Graphbase.

This section, and the previous one, show two important results. Firstly random sets of permutations form poor sets of symmetry breaking constraints, so the

Table 5. Solving problems with 1,000 randomly generated sets of minimal generators

Problem	Sizes	Average Nodes		Minimum		Wins		Nodes
		Gen	Any	Gen	Any	Gen	Any	Reduced
7,7,3,3,1 BIBD	2,3	112264	112056	39031	55837	500	500	23
3x5 D3	2,3,4	8,780,000	8,800,000	2,119,322	2,782,488	508	492	104,129
Anna.col	16	21,485,800	21,489,100	11,471,047	11,789,567	494	506	5,122,183

effort of defining particular subsets is worthwhile. Further, it is not important that the specialised sets of permutations we generate are generators, as in general generators perform no better than an arbitrary set of permutations.

6 Conclusion

The focus of this paper was to find a small reliable set of partial symmetry breaking constraints, which will work to efficiently eliminate symmetry for any problem modelled in any manner. To that end we have introduced the **Reduced** method of generating a set of partial symmetry breaking constraints. We have shown that in practice this method generates slightly more constraints than just using the generators produced by Nauty as the basis for constraints. Although, both methods are based on the stabiliser chain method. However, this slight increase in the number of constraints produced by the **Reduced** method, actually provides more efficient symmetry breaking than the Nauty based constraints. We have also shown that the **Reduced** method produces more constraints than Double Lex does, but again these constraints can provide more efficient symmetry breaking. In our BIBD experiment, where this was not the case, the two methods are comparable. Further, the **Reduced** method also outperforms a randomly chosen set of symmetry breaking constraints.

We have improved the understanding of small sets of symmetry breaking constraints. We show that **Nauty**, the method generally used in the past because of it's simplicity, is competitive, but not because it is a set of generators. We show strong evidence it is fixing many variables early in the variable ordering, rather than being generator sets, which make stabiliser chain based algorithms so effective at generating good sets of partial symmetry breaking constraints.

In general, we feel that the **Reduced** method is a very reliable way of providing a small set of partial symmetry breaking constraints, which perform well across a range of problems.

Acknowledgments. The authors wish to acknowledge that Dr Petrie is supported by a Royal Society Dorothy Hodgkins Research Fellow and Dr Jefferson by EPSRC grant number EP/H004092/1.

References

1. Puget, J.-F.: Automatic detection of variable and value symmetries. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 475–489. Springer, Heidelberg (2005)
2. Demoen, B., de la Banda, M.G., Mears, C., Wallace, M.: A novel approach for detecting symmetries in csp models. In: Proc. of The Seventh Intl. Workshop on Symmetry and Constraint Satisfaction Problems (2007)
3. Gent, I.P., Jefferson, C., Miguel, I.: Minion: A fast scalable constraint solver. In: Brewka, G., Coradeschi, S., Perini, A., Traverso, P. (eds.) Conf. ECAI 2006, pp. 98–102. IOS Press, Amsterdam (2006)
4. Gent, I.P., Smith, B.M.: Symmetry breaking in constraint programming. In: Proceedings of the 14th European Conference on Artificial Intelligence, ECAI 2000, Berlin, Germany, August 20-25, pp. 599–603. IOS Press, Amsterdam (2000)

5. Bjäreland, M., Jonsson, P.: Exploiting bipartiteness to identify yet another tractable subclass of CSP. In: Jaffar, J. (ed.) CP 1999. LNCS, vol. 1713, pp. 118–128. Springer, Heidelberg (1999)
6. McDonald, I., Smith, B.: Partial symmetry breaking. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, pp. 431–445. Springer, Heidelberg (2002)
7. Gecode Team: Gecode: Generic constraint development environment (2006), http://www.gecode.org
8. Crawford, J., Ginsberg, M., Luks, E., Roy, A.: Symmetry-breaking predicates for search problems. In: Proc. of the Intl. Conference Principles of Knowledge Representation and Reasoning, pp. 148–159 (1996)
9. Puget, J.F.: Breaking symmetries in all different problems. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence, pp. 272–277. Morgan Kaufmann Publishers Inc., San Francisco (2005)
10. Smith, B.M.: Sets of symmetry breaking constraints. In: Proc. Symcon, Agaoka (2005)
11. McKay, B.: Practical graph isomorphism. In: Numerical Mathematics and Computing, Proc. 10th Manitoba Conf., Winnipeg/Manitoba 1980, Congr. Numerantium, vol. 30, pp. 45–87 (1981), http://cs.anu.edu.au/people/bdm/nauty
12. Aloul, F.A., Sakallah, K.A., Markov, I.L.: Efficient symmetry breaking for boolean satisfiability. IEEE Transactions on Computers, 271–276 (2003)
13. Aloul, F., Ramani, A., Markov, I., Sakallah, K.: Solving difficult sat instances in the presence of symmetries. In: Proceedings of the Design Automation Conference, pp. 731–736 (2002)
14. Katsirelos, G., Narodytska, N., Walsh, T.: Breaking generator symmetry. In: The Ninth International Workshop on Symmetry and Constraint Satisfaction Problems (2009)
15. Flener, P., Frisch, A.M., Hnich, B., Kiziltan, Z., Miguel, I., Pearson, J., Walsh, T.: Breaking row and column symmetries in matrix models (2002)
16. Katsirelos, G., Narodytska, N., Walsh, T.: On the complexity and completeness of static constraints for breaking row and column symmetry. In: Cohen, D. (ed.) CP 2010. LNCS, vol. 6308, pp. 305–320. Springer, Heidelberg (2010)
17. Jerrum, M.: A compact presentation for permutation groups. J. Algorithms 7, 71–90 (2002)
18. Seress, A.: Permutation group algorithms. Cambridge tracts in mathematics, vol. (152). Cambridge University Press, Cambridge (2002)
19. Mears, C., De La Banda, M.G., Wallace, M.: On implementing symmetry detection. Constraints 14, 443–477 (2009)
20. Petrie, K.E., Smith, B.M.: Symmetry breaking in graceful graphs. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, pp. 930–934. Springer, Heidelberg (2003)
21. Meseguer, P., Torras, C.: Solving strategies for highly symmetric csps. In: IJCAI (1999)