Dynamic Symmetry Breaking Constraints

George Katsirelos[1] and **Toby Walsh**[2]

Abstract. We present a general method for dynamically posting symmetry breaking constraints during search. The basic idea is very simple. Given any set of symmetry breaking constraints, if during search a *symmetry* of one of these constraints is entailed and this is consistent with previously posted symmetry breaking constraints, then we post this constraint. We illustrate the method with two examples where a polynomial number of symmetry breaking constraints break an exponential number of symmetries. Like existing static methods for symmetry breaking, this symmetry breaking method benefits from fast and effective constraint propagation. In addition, like existing dynamic methods for symmetry breaking, this symmetry breaking methods does not conflict with the branching heuristic. Initial experimental results appear promising.

1 INTRODUCTION

Many search problems contain symmetries. For example, in scheduling problems, we can have identical orders or machines. As a second example, in workforce rostering problems, we can have equivalently skilled personnel. As a third example, in bin packing problems, we can have equal sized bins. Unless we take care, such symmetries will increase the size of the search space. In some cases, symmetries increase the size of the search space dramatically. For example, if we have n identical machines, then every schedule can be permuted into one of $n!$ symmetric schedules. If this symmetry is not factored out of search, we will waste a lot of time visiting symmetric search states.

There are a number of different methods commonly used to deal with symmetry. For example, we can *statically* add constraints before search which eliminate some or all of the symmetric solutions, or we can modify the search method so that it *dynamically* avoids symmetric solutions. Static symmetry breaking methods are simple to implement, work with any type of symmetry and tend to be highly effective. A small number of constraints can often quickly eliminate many symmetries. However, static methods have one disadvantage compared to dynamic methods: we fix in advance which solutions in each symmetry class are permitted, and branching heuristics may conflict with this choice.

In this paper, we propose a general method for posting static symmetry breaking constraints dynamically and incrementally during search. The posted symmetry breaking constraints are chosen to be consistent with the initial choices of the branching heuristic. This hybrid approach inherits good properties of both static and dynamic methods for symmetry breaking: we profit from fast propagation of the static symmetry breaking constraints, yet do not conflict with the branching heuristic.

[1] NICTA, Sydney, Australia, email: george.katsirelos@nicta.com.au.
[2] NICTA and UNSW, Sydney, Australia, email: tw@cse.unsw.edu.au.

2 BACKGROUND

A constraint satisfaction problem consists of a set of variables, each with a domain of values, and a set of constraints specifying allowed combinations of values for given subsets of variables. A solution is an assignment of values to variables satisfying the constraints. A common method to find a solution is backtracking search. Constraint solvers typically prune their search space by enforcing a local consistency property like domain consistency. A constraint is *domain consistent* iff for each variable, every value in its domain can be extended to an assignment that satisfies the constraint. We make a constraint domain consistent by pruning values for variables which cannot be in any solution. During the search for a solution, a constraint can become entailed. A constraint is *entailed* when any assignment of values left in the domain of the variables is a solution. For instance, $X_1 < X_9$ is entailed if and only if the largest value in the domain of X_1 is smaller than the smallest value in the domain of X_9.

Constraint satisfaction problems can contain symmetry. We will consider two special types of symmetry. A *variable symmetry* is a permutation of the variables that preserves solutions. Formally, a variable symmetry is a bijective mapping, σ of the indices of variables such that if $X_1 = d_1, \ldots, X_n = d_n$ is a solution then $X_{\sigma(1)} = d_1, \ldots, X_{\sigma(n)} = d_n$ is also. A *value symmetry*, on the other hand, is a permutation of the values that preserves solutions. Formally, a value symmetry is a bijective mapping, θ of the values such that if $X_1 = d_1, \ldots, X_n = d_n$ is a solution then $X_1 = \theta(d_1), \ldots, X_n = \theta(d_n)$ is also. More generally, symmetries can act simultaneously on variables and values. Our methods work with such general types of symmetry.

3 AN EXAMPLE

The basic idea is as follows:

> *Given any set of symmetry breaking constraints, if during search a symmetry of one of these constraints is entailed and this is consistent with previously posted symmetry breaking constraints, then we post this constraint so it holds also down all future branches.*

We illustrate this with a simple example involving just 8 symmetries. The *magic squares* problem is to label a n by n square so that the sum of every row, column and diagonal are equal (prob019 in CSPLib). A *normal* magic square contains the integers 1 to n^2. The problem has 8 symmetries corresponding to the rotations and reflections of the square. "Lo Shu", the unique normal magic square up to symmetry for $n = 3$, is an important object in ancient Chinese mathematics:

8	1	6
3	5	7
4	9	2

Consider a model with one variable for each label, an all-different constraint over all variables, and constraints that each row, column or diagonal adds up to $\frac{n^3+n}{2}$. We can rotate the solution given earlier so that the smallest corner label is at top left, and then reflect in the NW-SE diagonal so that the bottom left corner label is smaller than the top right. This eliminates all degrees of freedom.

2	7	6
9	5	1
4	3	8

We can therefore break all symmetry with the following constraints:

$$X[1,1] < X[1,n], \quad X[1,1] < X[n,1], \quad X[1,1] < X[n,n], \\ X[n,1] < X[1,n] \tag{1}$$

Where $X[1,1]$ is the top left corner label, $X[1,n]$ is the top right, $X[n,1]$ is the bottom left and $X[n,n]$ is the bottom right.

Any of the 8 symmetries of these ordering constraints would break all symmetry. For instance, consider the variable symmetry that rotates the magic square 90° clockwise and reflects in the NW-SE diagonal. This maps $X[1,1]$ onto $X[n,1]$, $X[1,n]$ onto $X[n,n]$, $X[n,1]$ onto $X[1,1]$, and $X[n,n]$ onto $X[1,n]$. Applying this variable symmetry to (1) gives:

$$X[n,1] < X[n,n], \quad X[n,1] < X[1,1], \quad X[n,1] < X[1,n], \\ X[1,1] < X[n,n]$$

That is, the smallest corner label is now at bottom left, and the top left is smaller than the bottom right. This set of constraints would also break all symmetry.

We choose which of the 8 symmetries of (1) to use incrementally during search. For example, suppose we begin search by assigning 1 to the bottom left corner:

?	?	?
?	?	?
1	?	?

As the variables take all different values, $X[1,1] > 1$, $X[1,n] > 1$, $X[n,1] = 1$ and $X[n,n] > 1$. Hence, at this point in search, the following ordering constraints are entailed:

$$X[n,1] < X[1,n], \quad X[n,1] < X[1,1], \quad X[n,1] < X[1,n]$$

That is, the smallest corner label is at bottom left. This is a 90° rotation anti-clockwise of the first three symmetry breaking constraints given in (1). It is also a reflection in the NW-SE diagonal followed by a 90° rotation anti-clockwise of (1). At this point, we do not need to choose between these two symmetries. We simply post the three entailed ordering constraints so that they hold on all future branches. Note that the top left and bottom right corners are not yet ordered. The branching heuristic is free to choose which is smaller.

Suppose, the branching heuristics instantiates the bottom row as follows:

?	?	?
?	?	?
1	5	9

Now $X[1,1] < 9$ as variables take all-different values and $X[n,n] = 9$. Hence, at this point, the following ordering constraint is entailed:

$$X[1,1] < X[n,n]$$

That is, the top left corner is smaller than the bottom right. This ordering constraint is consistent with the three ordering constraints already posted; the four ordering constraints can be obtained by reflecting (1) in the NW-SE diagonal and then rotating 90° anti-clockwise. We therefore post this fourth ordering constraint so that it holds on all future branches. All 8 symmetries are now broken in line with the choices of the branching heuristic. Backtracking will find the unique solution with the bottom left corner smallest, and the top left smaller than the bottom right:

4	3	8
9	5	1
2	7	6

In the rest of the paper, we describe two instances of this dynamic method. In each, we post symmetry breaking constraints incrementally during search that are consistent with the choices made by the branching heuristic. Whilst we give specific examples, the method is general and can be applied to *all* types of symmetry breaking constraints. For instance, the method works with specialized symmetry breaking constraints like those used for breaking row or column symmetries [3]. It also works with general purpose symmetry breaking constraints like the "lex-leader" constraints [1].

4 INTERCHANGEABLE VALUES

We consider an example where dynamically posting symmetry breaking constraints during search is especially simple. A common type of value symmetry is when values are interchangeable. For example, when coloring the vertices in a graph, the colors (values) are interchangeable. Suppose we have n variables, X_1 to X_n taking interchangeable values from 1 to m. Based on [7], we can statically break such symmetry by converting it into variable symmetry and ordering the introduced variables. We begin by channelling into variables, Z_j representing the indices at which values first occur:

$$X_i = j \quad \Rightarrow \quad Z_j \leq i \\ Z_j = i \quad \Rightarrow \quad X_i = j$$

Where $1 \leq i \leq n$, $1 \leq j \leq m$, and $Z_j \in [1, n+m)$. We then break all symmetry by posting the ordering constraints:

$$Z_1 < Z_2 < Z_3 < \ldots < Z_m \tag{2}$$

These ordering constraints enforce "value precedence" [5]. That is, they ensure that the first occurrence of j is before that of k for $j < k$.

Example 1 *Consider the following assignment:*

$$X_1, X_2, \ldots, X_6 \quad = \quad 1, 1, 2, 1, 3, 2$$

In this case, $Z_1 = 1$, $Z_2 = 3$ and $Z_3 = 5$. Thus (2) is satisfied and the assignment obeys value precedence.

Consider, on the other hand, the symmetric assignment in which we interchange 2 and 3:

$$X_1, X_2, \ldots, X_6 \quad = \quad 1, 1, 3, 1, 2, 3$$

In this case, $Z_2 = 5$ and $Z_3 = 3$ so (2) is not satisfied. This assignment therefore does not satisfy value precedence.

As an alternative to posting (2), we can break symmetry by posting *any* symmetry of (2). To be precise, if σ is any permutation of 1 to m, we can break all symmetry by posting:

$$Z_{\sigma(1)} < Z_{\sigma(2)} < Z_{\sigma(3)} < \ldots < Z_{\sigma(m)}$$

That is, the first occurrence of $\sigma(j)$ is before that of $\sigma(k)$ for $j < k$. For instance, we could post the following symmetry of (2):

$$Z_1 < Z_m < Z_2 < Z_{m-1} < \dots$$

This ensures 1 first occurs before m, which itself first occurs before 2, etc. It is simple to post incrementally a symmetry of (2) during search. We post the channelling constraints at the start of search as they are invariant to symmetry. Then, if at any point during search $Z_j < Z_k$ is entailed, we post $Z_j < Z_k$ so it holds on all future branches. To ensure transitivity of the Z_j variables, we maintain domain consistency on the channelling and ordering constraints. We call this $DynamicValOrder$.

Example 2 *Consider a constraint satisfaction problem with 4 interchangeable values. Suppose the branching heuristic first assigns $X_1 = 3$. The channelling constraints ensure $Z_3 = 1$, $Z_1 > 1$, $Z_2 > 1$ and $Z_4 > 1$. Hence $Z_3 < Z_1$, $Z_3 < Z_2$ and $Z_3 < Z_4$ are entailed. We therefore post these symmetry breaking ordering constraints so they hold on all future branches. These ensure that 3 is the first value used in any assignment.*

Suppose the branching heuristic next assigns $X_2 = 3$ and $X_3 = 1$. The channelling constraints ensure $Z_1 = 3$, $Z_2 > 3$ and $Z_4 > 3$. Hence $Z_1 < Z_2$ and $Z_1 < Z_4$ are entailed. We therefore post these ordering constraints so they hold on all future branches. At this point:

$$Z_3 < Z_1 < Z_2, \qquad Z_1 < Z_4$$

These constraints ensure that we only consider assignments in which 3 is used before 1, and 1 before 2 and 4. Note that values 2 and 4 are still interchangeable. The branching heuristic is free to choose which occurs fist.

Suppose we now backtrack. The channelling and symmetry breaking constraints leave no other choices for X_1, and just one other choice for X_2, namely $X_2 = 1$. Other assignments are symmetric to previously considered assignments (e.g. $X_1 = 3$, $X_2 = 2$ is symmetric to $X_1 = 3$, $X_2 = 1$, whilst $X_1 = 1$, $X_2 = 1$ is symmetric to $X_1 = 3$, $X_2 = 3$).

We prove that the $DynamicValOrder$ method breaks all symmetry. A symmetry breaking method is sound if it leaves at least one solution in each symmetry class, and complete if it leaves at most one solution.

Theorem 1 *$DynamicValOrder$ is a sound and complete symmetry breaking method for interchangeable values.*

Proof: Soundness follows quickly from the soundness of the underlying static symmetry breaking method. We have a relaxation that can only permit more assignments. Note that by maintaining domain consistency on the symmetry breaking constraints, we can always extend to a total ordering. Completeness also follows quickly from the completeness of the underlying static symmetry breaking method. Suppose we visit a complete assignment. Then we post ordering constraints for all used values. Suppose we now visit a second complete assignment that is in the same symmetry class. This contradicts one of the symmetry breaking constraints fixed by the first complete assignment. Hence, we cannot visit more than one complete assignment in each symmetry class. □

The method easily extends to partial interchangeability where values partition into equivalence classes, and values within each equivalence class are freely interchangeable. If at any point during search, $Z_j < Z_k$ is entailed where j and k are in the *same* equivalence class, then we post $Z_j < Z_k$ so it holds on all future branches.

5 VALUE PRECEDENCE

Our second example is more complex but provides additional propagation. Suppose we again have n variables, X_1 to X_n taking interchangeable values from 1 to m. As in the last section, we shall eliminate value interchangeability by enforcing a symmetry of value precedence. In [13], a global value propagator is proposed for the precedence constraint. Unlike the static method used in the last section, this propagator enforces domain consistency so prunes all possible symmetric values (see Theorem 5 in [14] for an example of symmetric values which are not pruned by the static method). The propagator in [13] uses a simple decomposition that we adapt to post symmetry breaking constraints incrementally.

We introduces $n + 1$ variables, Q_i for $i \in [0, n]$ that record the largest value used up to the index i. We set Q_i by posting:

$$Q_0 = 0, \qquad Q_i = \max(Q_{i-1}, X_i) \tag{3}$$

We then ensure value precedence by posting:

$$Q_{i+1} \quad \leq \quad 1 + Q_i \tag{4}$$

(3) and (4) break all symmetry due to interchangeable values.

Example 3 *Consider again:*

$$X_1, X_2, \dots, X_6 \quad = \quad 1, 1, 2, 1, 3, 2$$

Then, by (3):

$$Q_0, Q_1, \dots, Q_6 \quad = \quad 0, 1, 1, 2, 2, 3, 3$$

This satisfies (4). On the other hand, consider again the symmetric assignment in which we interchange 2 and 3:

$$X_1, X_2, \dots, X_6 \quad = \quad 1, 1, 3, 1, 2, 3$$

Then, by (3):

$$Q_0, Q_1, \dots, Q_6 \quad = \quad 0, 1, 1, 3, 3, 3, 3$$

This does not satisfy (4).

To post such symmetry breaking constraints incrementally during search, we take the somewhat counter-intuitive step of introducing *more* symmetry into the problem. We observe that value precedence can use any ordering on the values. For example, it could insist that the first occurrence of 3 is before that of 1, and that of 1 before that of 2. We introduce an ordering on values incrementally during search that is consistent with the branching heuristic. To define this new ordering, we introduce m variables, P_j. The constraints will ensure $P_j = k$ if and only if the value j is in the kth position in the value precedence ordering. To break all symmetry, we post:

$$\begin{array}{llll} Q_0 = 0, & Q_i = \max(Q_{i-1}, P_{X_i}), & Q_{i+1} \leq 1 + Q_i, & \\ Q_1 = 1, & \textsc{AllDiff}(P_1, \dots, P_m), & P_{X_i} \leq 1 + Q_{i-1} & \end{array} \tag{5}$$

Q_i now contains the maximum *position* in the ordering defined by P_j of all the values used up to index i.

These constraints introduces $m!$ variable symmetries into the problem since the total order defined by P_j can correspond to any of the $m!$ permutations of 1 to m. For instance, one total ordering is given by:

$$P_1 = 1, P_2 = 2, P_3 = 3 \dots, P_m = m \tag{6}$$

This will ensure 1 is the first value to occur ($P_1 = 1$), then 2 ($P_2 = 2$), then 3 ($P_3 = 3$), etc. Alternatively, we might have one of the $m!$ symmetries of (6) like:

$$P_1 = 1, P_2 = 3, P_3 = 5, \ldots, P_m = 2$$

This symmetry ensures 1 is the first value to occur ($P_1 = 1$), then m ($P_m = 2$), then 2 ($P_2 = 3$), etc.

We choose which symmetry of (6) to post incrementally during search. To do this, we maintain domain consistency on (5) and keep any prunings on the P_j when backtracking. We call this the $DynamicPrecedence$ method. The method again easily extends to partial interchangeability where values partition into equivalence classes.

Example 4 *Consider a constraint satisfaction problem with 4 interchangeable values. Suppose the branching heuristic first assigns $X_1 = 3$. From (5), we have $Q_1 = 1$ and $P_3 = 1$. As $P_3 = 1$, and P_j take all-different values, $P_1 > 1$, $P_2 > 1$ and $P_4 > 1$. Value precedence thus ensures that 3 is the first value used in any assignment. Suppose the branching heuristic next assigns $X_3 = 1$. From (5), we have $Q_2 \leq 2$, and thus $2 \leq P_1 \leq 3$. That is, the value 1 occurs 2nd or 3rd in the precedence ordering. This is to be expected. If $X_2 = 1$ or 3 then it occurs 2nd, whilst if $X_2 = 2$ or 4 then it occurs 3rd.*

Suppose we backtrack and next try $X_3 = 2$ instead. From (5), we have $2 \leq P_2 \leq 3$. That is, the value 2 also occurs 2nd or 3rd in the precedence ordering. Since we kept all prunings on P_j from the first branch, we still have $2 \leq P_1 \leq 3$. Thus P_1 and P_2 have two values between them. Propagating the all-different constraint then ensures $P_1 \in \{2, 3\}$, $P_2 \in \{2, 3\}$, $P_3 = 1$, $P_4 = 4$. At this point in search, value precedence ensures the value 3 occurs first, then 1 and 2 in either order, and the value 4 is the last of the interchangeable values to occur.

We prove that the $DynamicPrecedence$ method breaks all symmetry.

Theorem 2 *$DynamicPrecedence$ is a sound and complete symmetry breaking method for interchangeable values.*

Proof: Similar to $DynamicValOrder$. Note that by maintaining domain consistency on ALLDIFF($P_1, \ldots, P_m$), we can always construct a solution for the P_j. □

6 EXPERIMENTS

We implemented the symmetry breaking methods described in this paper in Gecode 2.0.1 and evaluated them on two problems: Schur numbers and graph coloring problems. Experiments were run on an 2-way Intel Xeon with 6MB of cache and 4 cores in each processor, running at 2GHz. Our hypothesis was that dynamic symmetry breaking methods would be less sensitive to the branching heuristic compared to static methods.

In our first experiments, we used graph coloring. Given a graph $G = \langle V, E \rangle$, we want to label each vertex $v \in V$ with a color $c(v)$, such that if $(u, v) \in E$ then $c(u) \neq c(v)$, using the smallest possible number of colors. We model this as an optimization problem with a variable for each vertex. The value of a variable is its assigned color. We post not-equals constraints among variables corresponding to neighboring vertices. All values in this problem are interchangeable. We break symmetry either with a static value precedence constraint [13] or with the $DynamicPrecedence$ method. The $DynamicValOrder$ method proved significantly slower especially on the harder problems. The results for two different value orderings, lexicographic and inverse lexicographic, are shown in the top of Table 1. All methods use the fail-first variable ordering heuristic.

We notice that the static symmetry breaking method is affected significantly by the value ordering. When using an inverse lexicographic value ordering, the static method performs uniformly worse than when using a lexicographic value ordering. The only exceptions to this are very easy instances and the instance `school1`, in which it finds a better solution. On the other hand, the dynamic method is largely unaffected by the value ordering and performs approximately the same with both branching heuristics. It is the best method in some cases, sometimes by a significant factor (e.g. `dsjc125.1gb` and `school1`). In addition, it is never significantly slower that the best performing method. As predicted, the pruning from static symmetry breaking constraints can interfere with the fail first heuristic, guiding search away from easy to find solutions. In contrast, dynamic methods impose no symmetry breaking at the start of search, and thus do not prevent the branching from finding a good coloring quickly.

In our second experiment, we used problems based on Schur numbers. The Schur number $S(k)$ is the largest integer n such that $[1, n]$ can be partitioned into k sets with a, b and c placed in the same partition only if $a + b \neq c$. We turn this into a hyper-graph coloring problem by fixing n and minimizing k. We use a variable X_i for each integer $1 \leq i \leq n$, and assign $X_i = j$ iff i is placed in the j^{th} partition. Each variable's domain is therefore $[1, k]$. We post not-all-equals constraints for each triplet X_a, X_b, X_c where $a + b = c$. Clearly all values are interchangeable, as we can swap two partitions of any solution without violating any constraints. We again break symmetry either with a static value precedence constraint or with the $DynamicPrecedence$ method. Results are shown at the bottom of Table 1. As hypothesized, the performance of the dynamic method is more robust to changes in the branching heuristic than the static method. Irrespective of the branching heuristic, the dynamic method explores an almost identical search tree to the lexicographic heuristic with static symmetry breaking. By comparison, with static symmetry breaking, the inverse lexicographic heuristic is faster on `schur-30` and `schur-40`, but is less successful on `schur-35`.

7 RELATED WORK

Puget proved that symmetric solutions can be eliminated by the addition of static constraints [6]. Crawford *et al.* presented the first general method for constructing static constraints for breaking variable symmetries [1]. Their "lex-leader" method constructs a symmetry breaking constraint for each symmetry which ensures that any solution found is lexicographically less than any of its symmetries. Crawford *et al.* also argued that it is NP-hard to eliminate all symmetric solutions in general. There are two weaknesses to the lex-leader method. First, it requires as many symmetry breaking constraints as symmetries. Second, it may conflict with the branching heuristic. Puget and Walsh independently extended the lex-leader method to value symmetries [9, 12]. The full set of lex-leader constraints can often be simplified. For example, if we have an array of decision variables with row symmetry (that is, the rows can be permuted), the exponential number of lex-leader constraints simplifies to a linear number of lexicographical ordering constraints between rows [11, 3]. As a second example, for problems where variables are symmetric and must take all different values, Puget has shown that the lex-leader constraints simplify to a linear number of ordering constraints [8].

Problem	Static symmetry breaking						Dynamic symmetry breaking					
	Lex			Inverse Lex			Lex			Inverse Lex		
	k	t (f/p)	b (f/p)	k	t (f/p)	b (f/p)	k	t (f/p)	b (f/p)	k	t (f/p)	b (f/p)
david	**10**	**0.09** / -	135 / -	**10**	0.44 / -	667 / -	**10**	0.42 / -	**0** / -	**10**	0.43 / -	**0** / -
dsjc1251gb	**4**	222.02 / 223.41	533031 / 536151	**4**	328.59 / 329.75	808114 / 810870	**4**	**29.97** / **31.88**	**65776** / **69766**	**4**	33.27 / 35.39	**65776** / **69766**
fullins3	**5**	**0.08** / -	96 / -	**5**	0.28 / -	520 / -	**5**	0.18 / -	**0** / -	**5**	0.17 / -	**0** / -
geom50a	**8**	1.25 / 9.18	15726 / 77246	**8**	**0.06** / 8.55	176 / 61755	**8**	0.08 / **1.32**	**0** / **6721**	**8**	0.07 / **1.32**	**0** / **6721**
miles250	**7**	**0.31** / -	242 / -	**7**	1.29 / -	1151 / -	**7**	1.41 / -	**0** / -	**7**	1.36 / -	**0** / -
myciel4	**4**	**0.01** / **0.02**	**0** / 202	**4**	**0.01** / **0.02**	38 / **162**	**4**	**0.01** / **0.02**	**0** / 188	**4**	**0.01** / **0.02**	**0** / 188
myciel5	**5**	**0.01** / 23.21	**0** / **287203**	**5**	0.05 / **23.12**	177 / 287252	**5**	0.06 / 29.22	**0** / 288622	**5**	0.06 / 29.3	**0** / 288622
r501g	**2**	**0.02** / **0.02**	**7** / **10**	**2**	0.07 / 0.07	199 / 201	**2**	0.07 / 0.07	12 / 15	**2**	0.07 / 0.07	12 / 15
r505gb	**9**	0.29 / 13.53	2196 / 100199	**9**	0.08 / 13.98	349 / 98586	**9**	**0.06** / **0.12**	**6** / **257**	**9**	0.07 / **0.12**	**6** / **257**
school1	39	5.51 / -	590 / -	27	221.33 / -	37886 / -	**21**	**56.41** / -	**0** / -	**21**	62.36 / -	**0** / -
zeroini1	**48**	**0.75** / -	**0** / -	50	8.01 / -	2921 / -	**48**	13.26 / -	**0** / -	**48**	11.46 / -	**0** / -
schur-30	**4**	2.24 / 2.38	20432 / 21091	**4**	**0.69** / **0.83**	**5024** / **5691**	**4**	2.50 / 2.64	20433 / 21095	**4**	2.51 / 2.66	20433 / 21095
schur-35	**4**	**14.58** / **14.77**	**137197** / **137859**	**4**	163.36 / 163.55	1039774 / 1040443	**4**	16.75 / 16.95	137198 / 137863	**4**	17.42 / 17.62	137198 / 137863
schur-40	6	0.05 / -	38 / -	**5**	**0.11** / -	**328** / -	6	0.05 / -	38 / -	6	0.05 / -	38 / -

Table 1. Static versus dynamic symmetry breaking. The table has four sections: static symmetry breaking constraints with lexicographic value ordering, static symmetry breaking constraints with inverse lexicographic value ordering, dynamic symmetry breaking constraints with lexicographic value ordering, and dynamic symmetry breaking constraints with inverse lexicographic value ordering. Each of the sections shows the number of colors **k** in the best solution found within the timeout, the **t**ime and the number of **b**ranches needed to **f**ind the best solution and to **p**rove optimality. "-" indicates that no solution was found (resp. optimality was not proven) within the timeout. The best results for each instance are in bold.

A number of dynamic methods have been proposed to deal with symmetry. For instance, SBDS posts symmetry breaking constraints dynamically during search [4]. SBDS can be seen as instance of the more general method proposed here. A limitation of SBDS is that it adds a symmetry breaking constraint for each unbroken symmetry. As there can be an exponential number of symmetries, this can be prohibitive. One of our main insights is that we can post *other* types of symmetry breaking constraint dynamically during search. A small number of symmetry breaking constraints may be adequate for special symmetries (e.g. those due to interchangeable values) or special classes of problems (e.g. problems where variables are all-different). Another dynamic method for breaking symmetry is SBDD [2]. This checks if a node of the search tree is symmetric to some previously explored node. Finally, Roney-Dougal *et al.* gave a dynamic method to construct a GE-tree, a search tree without value symmetry [10]. A weakness of both these dynamic methods is that they do not propagate their symmetry breaking constraints. It has been shown that propagation between the problem constraints and the static symmetry breaking constraints can reduce search exponentially [14].

8 CONCLUSIONS

We have presented a general method for dynamically and incrementally posting symmetry breaking constraints during search. The basic idea is very simple. Given any set of symmetry breaking constraints, if during search a symmetry of one of these constraints is entailed and this is consistent with previously posted symmetry breaking constraints, then we post this symmetry breaking constraint so it holds on all future branches. We illustrated the method with two examples where a polynomial number of symmetry breaking constraints can break an exponential number of symmetries. Both examples eliminate all symmetry due to interchangeable values. The first is simpler whilst the second is more complex but provides more propagation. This hybrid approach inherits good properties of both dynamic and static symmetry breaking methods: we have fast and efficient propagation of the posted symmetry breaking constraints, yet we do not conflict with the branching heuristic. Initial experimental results appear promising. In future work, we intend to develop such hybrid methods for other types of symmetry.

REFERENCES

[1] J. Crawford, G. Luks, M. Ginsberg, and A. Roy, 'Symmetry breaking predicates for search problems', in *Proceedings of the 5th International Conference on Knowledge Representation and Reasoning, (KR '96)*, pp. 148–159, (1996).

[2] T. Fahle, S. Schamberger, and M. Sellmann, 'Symmetry breaking', in *Proceedings of 7th International Conference on Principles and Practice of Constraint Programming (CP2001)*, ed., T. Walsh, pp. 93–107. (2001).

[3] P. Flener, A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh, 'Breaking row and column symmetry in matrix models', in *8th International Conference on Principles and Practices of Constraint Programming (CP-2002)*. (2002).

[4] I.P. Gent and B.M. Smith, 'Symmetry breaking in constraint programming', in *Proceedings of ECAI-2000*, ed., W. Horn, pp. 599–603. IOS Press, (2000).

[5] Y.C. Law and J.H.M. Lee, 'Global constraints for integer and set value precedence', in *Proceedings of 10th International Conference on Principles and Practice of Constraint Programming (CP2004)*, pp. 362–376. (2004).

[6] J.-F. Puget, 'On the satisfiability of symmetrical constrained satisfaction problems', in *Proceedings of ISMIS'93*, eds., J. Komorowski and Z.W. Ras, LNAI 689, pp. 350–361. (1993).

[7] J-F. Puget, 'Breaking all value symmetries in surjection problems', in *Proceedings of 11th International Conference on Principles and Practice of Constraint Programming (CP2005)*, ed., P. van Beek. (2005).

[8] J-F. Puget, 'Breaking symmetries in all different problems.', in *Proceedings of 19th IJCAI*, pp. 272–277. (2005).

[9] J-F. Puget, 'An efficient way of breaking value symmetries', in *Proceedings of the 21st National Conference on AI*. AAAI, (2006).

[10] C. Roney-Dougal, I. Gent, T. Kelsey, and S. Linton, 'Tractable symmetry breaking using restricted search trees', in *Proceedings of ECAI-2004*. IOS Press, (2004).

[11] I. Shlyakhter, 'Generating effective symmetry-breaking predicates for search problems', in *Proceedings of LICS workshop on Theory and Applications of Satisfiability Testing (SAT 2001)*, (2001).

[12] T. Walsh, 'General symmetry breaking constraints', in *12th International Conference on Principles and Practices of Constraint Programming (CP-2006)*. (2006).

[13] T. Walsh, 'Symmetry breaking using value precedence', in *Proceedings of the 17th ECAI*. IOS Press, (2006).

[14] T. Walsh, 'Breaking value symmetry', in *13th International Conference on Principles and Practices of Constraint Programming (CP-2007)*. (2007).