Approaches to Conditional Symmetry Breaking*

Ian P. Gent[1], **Iain McDonald**[1], **Ian Miguel**[2], **Barbara M. Smith**[3]
[1]School of Computer Science, University of St Andrews, St Andrews, Fife, KY16 9SS, UK
[2]Department of Computer Science, University of York, YO10 5DD, UK
[3]Cork Constraint Computation Centre, Department of Computer Science,
University College Cork, Cork, Ireland
{ipg,iain}@dcs.st-and.ac.uk, ianm@cs.york.ac.uk, b.smith@4c.ucc.ie

Abstract

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

1 Introduction

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

One important aspect of modelling is dealing with *symmetry*. Symmetry is a solution-preserving transformation; symmetry in a model can result in a great deal of wasted effort when the model is solved via systematic search. Hence, it is necessary to ensure that symmetry is broken effectively. The majority of research in symmetry in constraint models considers only the symmetry present in a model before search begins. However, as we will discuss, it is commonly the case that symmetry can form during search. We refer to this as *conditional* symmetry, since its formation depends on the choices made during search. In order to avoid redundant search, it is important to break this symmetry in addition to unconditional symmetry breaking.

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

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

2 Background

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

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

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

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

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

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

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

3 Conditional Symmetry-breaking Constraints

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

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

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

3.1 Graceful Graphs

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

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

The constraints of the problem are:

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

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

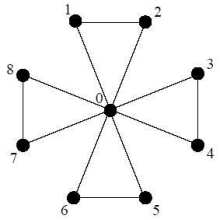

Figure 1: The windmill graph $C_3^{(4)}$

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

The CSP has three instance-independent symmetries:

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

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

Relabelling a Triangle

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

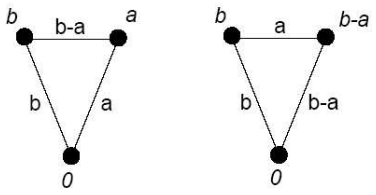

Figure 2: Relabelling a triangle in a graceful labelling with centre node labelled 0.

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

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

Labellings with Centre Node Labelled 1

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

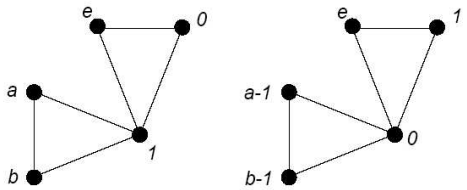

Figure 3: Transforming a labelling with centre node labelled 1.

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

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

Symmetry-Breaking Constraints

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

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

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

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

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

Results

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

3.2 Steel Mill Slab Design

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

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

A Matrix Model

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

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

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

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

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

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

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

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

Symmetry and Conditional Symmetry

In this initial model, $slab_M$ has instance-independent column symmetry: a (non-)solution can be transformed into a (non-)solution by permuting the values assigned to each element of $slab_M$ and permuting the corresponding rows of $order_M$. This symmetry can be broken simply by ordering the elements of $slab_M$ as follows:

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

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

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

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

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

$$\begin{array}{c} \\ slab_1 \\ slab_2 \\ \ldots \end{array} \begin{array}{c} a \\ \left(\begin{array}{ccc} 1 & 0 & \ldots \\ 0 & 2 & \ldots \\ \ldots & \ldots & \ldots \end{array}\right) \end{array} \qquad \begin{array}{c} \\ slab_1 \\ slab_2 \\ \ldots \end{array} \begin{array}{c} a \\ \left(\begin{array}{ccc} 0 & 2 & \ldots \\ 1 & 0 & \ldots \\ \ldots & \ldots & \ldots \end{array}\right) \end{array}$$

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

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

Formation of Compound Order Symmetry

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

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

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

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

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

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

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

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

$$subsum_M[position-1] < compoundSize \times instanceNo$$
$$subsum_M[position] \geq compoundSize \times instanceNo$$

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

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

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

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

Breaking Compound Order Symmetry

Consider n symmetrical compound orders. We order these compounds ascending by the column on which they appear, breaking ties by ordering the 'first', 'second', ..., 'nth' compounds in a column, as defined in the previous section, ascending. We denote the *switch* and *position* variables of the ith compound under this ordering as $switch_i$ and $position_i$. The conditional symmetry can be broken straightforwardly as follows:

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

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

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

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

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

Experimental Results

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

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

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

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

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

4 Breaking Conditional Symmetry by Reformulation

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

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

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

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

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

5 A Generic Method of Breaking Conditional Symmetries

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

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

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

5.1 The problem with using SBDS to break conditional symmetry

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

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

5.2 Using SBDD to break conditional symmetries

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

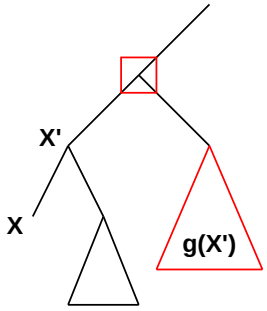

Figure 6: Upon backtracking to the highlighted node from X' SBDS posts a constraint to forbid $g(X')$ in the local subtree.

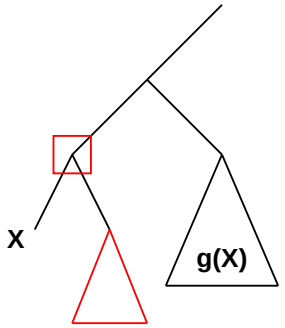

Figure 7: The symmetrical variant $g(X)$ of the nogood X, does not exist in the local subtree. SBDS can cope with such a situation by forbidding $g(X')$ (as shown in Figure 6). The **condition** that makes a bijective mapping h a symmetry, for example, may not hold at the node X', hence the nogood $h(X')$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

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

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

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

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

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

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

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

6 Conclusions and Future Work

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

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

References

[1] J. Crawford, M. L. Ginsberg, E. Luks, A. Roy. Symmetry-breaking Predicates for Search Problems. *Proc. 5th Int. Conf. on Principles of Knowledge Representation & Reasoning*, pp. 148–159, 1996.

[2] T. Fahle, S. Schamberger, M. Sellmann. Symmetry Breaking. *Proc. 7th Int. Conf. on Principles & Practice of Constraint Programming*, 93-107, 2001.

[3] A. M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, T. Walsh. Global Constraints for Lexicographic Orderings. *Proc. 8th Int. Conf. on Principles & Practice of Constraint Programming*, pp. 93–108, 2002.

[4] A. M. Frisch, I. Miguel, T. Walsh. Symmetry and Implied Constraints in the Steel Mill Slab Design Problem. *Proc. CP'01 Workshop on Modelling & Problem Formulation*, pp. 8–15, 2001.

[5] J. A. Gallian. Graph Labeling. *The Electronic Journal of Combinatorics*, Dynamic Surveys (DS6), `www.combinatorics.org/Surveys`,2003.

[6] I. P. Gent, W. Harvey, T. Kelsey. Groups and Constraints: Symmetry Breaking During Search. *Proc. 8th Int. Conf. on Principles & Practice of Constraint Programming*, 415-430, LNCS 2470, 2002.

[7] I. P. Gent, W. Harvey, T. W. Kelsey, S. A. Linton. Generic SBDD Using Computational Group Theory. *Proc. 9th Int. Conf. on Principles & Practice of Constraint Programming*, pp. 333-347, 2003.

[8] I. P. Gent, I. McDonald, B. M. Smith. Conditional Symmetry in the All-Interval Series Problem. *Proc. 3rd Int. Workshop on Symmetry in Constraint Satisfaction Problems*, 2003.

[9] I. P. Gent, B. M. Smith. Symmetry Breaking in Constraint Programming. *Proc. 14th European Conf. on AI*, 599-603, 2000.

[10] S. Prestwich, J. C. Beck. Exploiting Dominance in Three Symmetric Problems. *Proc. 4th Int. Workshop on Symmetry & Constraint Satisfaction Problems*, 2004.