Symmetry Breaking in Graceful Graphs

Karen E. Petrie and Barbara M. Smith

University of Huddersfield, Huddersfield HD1 3DH, U.K.
[k.e.petrie,b.m.smith]@hud.ac.uk

Abstract. Symmetry in a Constraint Satisfaction Problem (CSP) can cause wasted search, which can be avoided by adding constraints to the CSP to exclude symmetric assignments or by modifying the search algorithm so that search never visits assignments symmetric to those already considered. Two such approaches are SBDS (Symmetry Breaking During Search) and SBDD (Symmetry Breaking by Dominance Detection); modifications of these are GAP-SBDS and GAP-SBDD, which work with the symmetry group rather than the individual symmetries. There has been little experience of how these techniques compare in practice. We compare SBDS, GAP-SBDS and GAP-SBDD in finding all *graceful labellings* of graphs with symmetry. For these problems, the benefits of GAP-SBDS over SBDS outweigh the additional overheads, except when the number of symmetries is small. When simple constraints can be found to break all the symmetry, they can give better problem-solving performance than GAP-SBDS; however, if the constraints break only part of the symmetry, GAP-SBDS does less search and is faster. Surprisingly, GAP-SBDD is slower than GAP-SBDS for these problems, and we show that this is due to a feature of the CSP model. Eliminating symmetry has allowed us to find all graceful labellings, or prove that there are none, for several graphs whose gracefulness was not previously known.

1 Introduction

Symmetry in a Constraint Satisfaction Problem (CSP) occurs when different (partial or complete) solutions to the CSP correspond to essentially the same solution to the problem that the CSP models. For instance, in 3-colouring the nodes of a graph, it is only necessary to partition the nodes into three sets of different colours. If a CSP model allocates a specific colour to each node, it will have sets of equivalent solutions in which the three colours are permuted.

More formally, a symmetry of a CSP P whose set of constraints is C is a bijective function f such that for any partial or full assignment A to the variables in P, $f(A)$ satisfies the constraints C iff A does.

Symmetry in CSPs can cause wasted search, because the search for solutions may repeatedly visit partial assignments symmetric to ones already considered. If a partial assignment does not lead to a solution, neither will any symmetrically equivalent assignment. If we are searching for all solutions, then for any solution found, we will also find all its symmetrically equivalent solutions; in terms of the original problem modelled by the CSP, these are not distinct solutions.

A common way to reduce or eliminate symmetry is to add constraints to the CSP, to exclude some or all symmetric equivalents. Ideally, the new constraints should be satisfied by only one assignment in any symmetry equivalence class. It is often hard to find simple symmetry constraints; moreover, they can interact badly with the search strategy. It may be that amongst the solutions in a symmetry equivalence class that do not satisfy the new constraints is one that would be found earlier, given the search strategy being used, than any of the solutions that can still be found.

An alternative is to adapt the search algorithm so that constraints are added during search to prevent exploration of symmetric assignments. Symmetry Breaking During Search [1, 10] adds such constraints on backtracking to a choice point: having explored the subtree resulting from the choice (typically the assignment of a value to a variable, say $var = val$), the search will explore the subtree resulting from the opposite choice ($var \neq val$). If the set of assignments already made along the path from the root to this point is A, then for every symmetry g, SBDS adds the constraint $g(A) \Rightarrow g(var \neq val)$.

SBDS has been implemented in ILOG Solver, and an SBDS library is now available in the ECL^{i}PS^e constraint programming system. SBDS requires a function for each symmetry in the problem describing its effect on the assignment of a value to a variable. Although SBDS has been successfully used with a few thousand symmetry functions, many problems have too many symmetries to allow a separate function for each.

To allow SBDS to be used in such situations, Gent *et al.* [8] linked SBDS (in ECL^{i}PS^e) with GAP (Groups, Algorithms and Programming) [6], a system for computational discrete algebra and in particular computational group theory. GAP-SBDS allows the symmetry group, rather than its individual elements, to be described. GAP is used, for instance, to find the *stabiliser* of a partial assignment, i.e. the subgroup which leaves it unchanged.

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

Symmetry Breaking via Dominance Detection (SBDD) [3] performs a check at every node in the search tree to see if it is dominated by a symmetric equivalent of a subtree already explored, and if so prunes this branch. Harvey [11] explains that SBDS and SBDD are closely related; the difference is where in the search tree, and how, symmetry breaking is enforced. In SBDD, the dominance detection function is based on the problem symmetry and is hand-coded for each problem. Gent *et al.* [9] have recently developed GAP-SBDD, a generic version of SBDD that uses the symmetry group of each problem rather than an individual dominance detection function and links SBDD (in ECL^{i}PS^e) with GAP. At each node in the search tree ECL^{i}PS^e communicates with GAP, which performs the dominance check. Sometimes when the node is not dominated, GAP can identify variable/value pairs which can be deleted from domains; this information is returned to ECL^{i}PS^e for use in the search.

In [8], limited experiments with GAP-SBDS are reported; it is much faster than the original SBDS on the Alien Tiles problem [7], where the symmetry group has 1152 elements. Experiments with two Balanced Incomplete Block Design problems (BIBDs) showed that it can successfully handle symmetry groups with millions of elements, which SBDS clearly cannot do. However, symmetry constraints gave better results than GAP-SBDS on the two problems considered, except for variable orderings incompatible with the constraints. Fahle *et al.* [3] showed that SBDD could find a set of non-isomorphic solutions to the golfers' problem when SBDS was unable to, although SBDD used a simpler model. Gent *et al.* compared GAP-SBDD with GAP-SBDS applied to BIBDs and showed that GAP-SBDD could solve much larger problems, and was faster than GAP-SBDS on the smaller problems which both could solve.

However, more experience of these techniques is required in order to identify which is appropriate for a given situation. For instance, we need to know whether GAP-SBDD will be faster than GAP-SBDS in general. If simple constraints can be found that will break some of the symmetry in a CSP, we need to know whether we should use them, or use one of the techniques that break symmetry during search. In this paper, we investigate symmetry breaking in a class of graph labelling problems. We compare SBDS with GAP-SBDS; GAP-SBDS with constraints to break the symmetry; and GAP-SBDS with GAP-SBDD.

As well as providing further experience of these techniques in practice, constraint programming has proved to be a valuable technique for investigating these problems. Eliminating symmetry has allowed us to find many new results. We present some of these results here; more can be found at http://scom.hud.ac.uk/scombms/Graceful.

2 Graceful Graphs

A labelling f of the nodes of a graph with q edges is *graceful* if f assigns each node a unique label from $\{0, 1, ..., q\}$ and when each edge xy is labelled with $|f(x) - f(y)|$, the edge labels are all different. Figure 1 shows an example. The study of graceful graphs is an active area of graph theory. Gallian [5] surveys graceful graphs, i.e. graphs which have a graceful labelling, and lists the graphs whose status is known. [5] is the

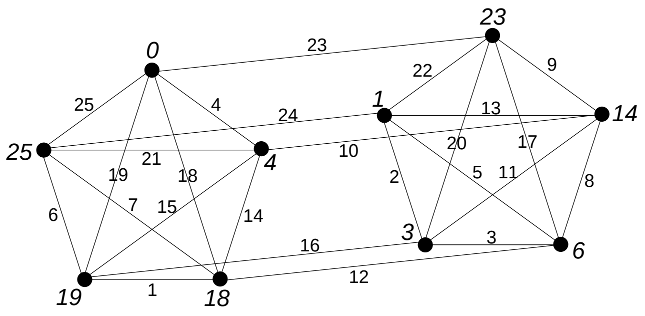

Fig. 1. The unique graceful labelling of $K_5 \times P_2$.

latest version of a dynamic survey which first appeared in 1997 and has been regularly updated.

Finding a graceful labelling of a given graph, or proving that one does not exist, can be expressed as a CSP. Specific cases have previously been considered; for instance, the all-interval series problem (problem 007 in CSPLib, at http://www.csplib.org) is equivalent to finding a graceful labelling of a path, and Lustig & Puget [12] found a graceful labelling of a graph discussed in section 3.

A CSP model has a variable for each node, $x_1, x_2, ..., x_n$, each with domain $\{0, 1, ..., q\}$ and a variable for each edge, $d_1, d_2, ..., d_q$, each with domain $\{1, 2, ..., q\}$. The constraints of the problem are: if edge k joins nodes i and j then $d_k = |x_i - x_j|$; $x_1, x_2, ..., x_n$ are all different; and $d_1, d_2, ..., d_q$ are all different.

Other modelling decisions were made after experiments to find a model which would give good results in general. The allDifferent constraint on the node variables is treated as a set of binary $\neq$ constraints, whereas for the edge variables we use the highest level of propagation provided for the allDifferent constraint in ECL^{i}PS^e. They are treated differently because the values assigned to the edge variables form a permutation and hence give more scope for domain pruning than the node variables, which have far more possible values than variables.

We assign values to the node variables only and use lexicographic variable ordering. In [12], a smallest domain ordering was found to be far superior; we have found it usually better, but not dramatically so. On some graphs, when using symmetry constraints, smallest domain ordering is seriously misled and gives poor results. Using a static variable ordering also simplifies the comparison of symmetry-breaking strategies; differences in performance are due solely to the symmetry breaking.

To find all graceful labellings of all but the simplest graphs, it is essential to eliminate all or most of the symmetry, both to identify a set of non-isomorphic solutions and to avoid wasted search. There are two kinds of symmetry in the CSP: first, there may be symmetry in the graph. For instance, if the graph is a clique, any permutation of the node labels in a graceful labelling is also graceful. If the graph is a path, P_n, the node labels can be reversed to give an equivalent labelling. The second type of symmetry is that we can replace the value of every node variable x_i by its complement $q - x_i$. We can also combine each graph symmetry with the complement symmetry. For instance, the graceful labelling (0, 3, 1, 2) of P_4 has three symmetric equivalents: reversal (2, 1, 3, 0); complement (3, 0, 2, 1); reversal & complement (1, 2, 0, 3).

3 $K_m \times P_2$ Graphs: SBDS *v.* GAP-SBDS

The graph shown in Figure 2, with the node numbering used in the CSP and one of its graceful labellings, is the cross-product of the clique K_4 and the path P_2: it consists of two copies of K_4, with corresponding vertices in the two cliques also forming the vertices of a path P_2. Lustig & Puget [12] found a graceful labelling of this graph; it was not previously known to be graceful. However, they looked for only one graceful labelling of the graph and did not break any of the symmetry.

The symmetries of the graph are, first, intra-clique permutations: any permutation of the 4-cliques which acts on both in the same way. For instance, we can transpose nodes

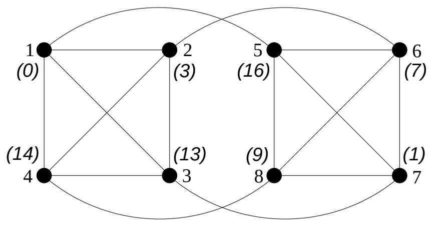

Fig. 2. The graph $K_4 \times P_2$.

1 and 2 and simultaneously 5 and 6. Second, inter-clique permutations: the labels of the first clique (nodes 1, 2, 3, 4) can be interchanged with the labels of the corresponding nodes (5, 6, 7, 8) in the second. These can also be combined with each other and with the complement symmetry. Hence, the size of the symmetry group is 4! × 2 × 2, or 96.

GAP-SBDS requires the symmetry group of the problem as input. In GAP-SBDS, symmetries act over *points*, which correspond to variable-value pairs. To simplify inputting any group (such as the subgroup of graph symmetries) which affects only the CSP variables, we have extended GAP-SBDS by writing code (in the GAP language) to convert a set of generators for such a group to act over points. For $K_4 \times P_2$, possible permutations of the nodes which will generate the graph symmetries are, in cyclic form, (1 2)(5 6), (2 3)(6 7) and (3 4)(7 8) for the permutations within the cliques and (1 5)(2 6)(3 7)(4 8) to interchange the two cliques. These will generate all 48 elements of the graph symmetry group. (Note that the set of generators need not be minimal; as pointed out in [8], the user can add representative permutations of each type of symmetry in the problem to the set of generators, until GAP produces the full group.) We have written further GAP code to transform the graph symmetry generators to form new generators over the points for the full symmetry group, including the complement symmetries, which affect the values of the node variables. This could be generalised to other cases where some symmetries affect the variables and some the values.

In SBDS we require a function for every symmetry other than the identity: i.e. 95 functions for $K_4 \times P_2$. We used GAP to output the required functions, using the same generators as for GAP-SBDS. As described in [7], GAP was used in a similar fashion to produce the symmetry functions for the Alien Tiles problem.

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

A: We can eliminate all the symmetry (i.e. the full symmetry group of 96 elements) using SBDS or GAP-SBDS.

B: Alternatively, we might choose not to eliminate the complement symmetries; at worst this will double the number of solutions. We are left with just the graph symmetry group, i.e. 48 symmetries.

C: Once we ignore the complement symmetry, the inter-clique symmetry can be eliminated by adding constraints to the CSP, provided that they do not interfere with permuting the node labels within the cliques. A permissible constraint is that the small-

	$K_3 \times P_2$				$K_4 \times P_2$				$K_5 \times P_2$			
	SBDS		GAP-SBDS		SBDS		GAP-SBDS		SBDS		GAP-SBDS	
Strategy	BT	sec.	BT	sec.	BT	sec.	BT	sec.	BT	sec.	BT	sec.
A	6	0.25	9	0.54	147	12.9	165	8.3	4172	1356	4390	382
B	16	0.24	16	0.59	369	13.1	369	14.3	9889	929	9889	793
C	16	0.21	16	0.58	369	11.0	369	14.1	9889	659	9889	783
D	16	0.2	-	-	369	10.6	-	-	9889	629	-	-

Table 1. Comparison of different levels of symmetry breaking using SBDS or GAP-SBDS for finding all graceful labellings of $K_n \times P_2$. BT is the number of backtracks.The running time is on a 1.6GHz Pentium 4, running ECL^{i}PS^e.

est node label in the first clique is less than the smallest in the last. With this constraint, SBDS or GAP-SBDS need only eliminate the 24 remaining symmetries.

D: the 24 remaining symmetries consist of all permutations of the subsets {1,5}, {2,6}, {3,7} and {4,8} of the variables. This is a generalisation of symmetry due to variables being indistinguishable, and in this special case, the symmetry can be eliminated by just the transpositions of the variables, or sets of variables, being permuted. Here, each transposition acts on two of the variable subsets, e.g. one swaps the labels of nodes 1 and 2, and of nodes 5 and 6. In SBDS, we provide a function for each of the six transpositions. This cannot be done in GAP-SBDS, however, because the subset of transpositions is not closed under composition and so is not a subgroup of the full symmetry group. As in strategy C, the inter-clique symmetry is broken by a constraint.

These different ways of dealing with symmetry in $K_4 \times P_2$ using SBDS and GAP-SBDS are compared in Table 1. Results are also given for $K_3 \times P_2$, with 24 symmetries, and $K_5 \times P_2$, with 480. There are 4 non-isomorphic graceful labellings of $K_3 \times P_2$ and 15 of $K_4 \times P_2$. $K_5 \times P_2$ has a unique graceful labelling (shown in Figure 1). Strategies B, C and D only break the graph symmetries and so find twice as many solutions. $K_3 \times P_2$ and $K_4 \times P_2$ were already known to be graceful, but the number of non-isomorphic graceful labellings, and the results for $K_5 \times P_2$, are new.

For each graph, when all but the complement symmetries are eliminated, i.e. for strategies B, C and D, SBDS and GAP-SBDS incur the same search effort, although the runtime differs considerably. In strategy A, SBDS does less search than GAP-SBDS; this seems to be due to the lazy evaluation of $g(A)$ in GAP-SBDS, to delay imposing constraints. The best strategy for SBDS is D, where the number of symmetry functions is smallest. Increasing the number of symmetry functions severely affects the running time of SBDS, to the extent that strategy A is much the slowest strategy for SBDS on the largest problem, even though the number of backtracks is more than halved. The running time of GAP-SBDS, on the other hand, is much less affected by the number of symmetries, and its best strategy is to break all the symmetry and hence benefit from the reduction in search.

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

We can extend these symmetry-breaking strategies to the graph $K_4 \times P_3$. This has a third 4-clique whose nodes are joined pairwise to the second. We have found the 704 non-isomorphic graceful labellings of this graph. As with $K_5 \times P_2$, $K_4 \times P_3$ was not previously known to be graceful. We compared GAP-SBDS, breaking all the symmetry, with SBDS using strategy D, i.e. the best strategy for each method. The symmetries are exactly as in $K_4 \times P_2$, but instead of swapping the first and second clique, we swap the first and third. GAP-SBDS takes 386,068 backtracks and 21150 sec.; SBDS takes 844,629 backtracks and 32700 sec.

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

4 Symmetry-Breaking Constraints

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

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

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

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

This symmetry transforms the general solution $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8)$ to $(q - x_5, q - x_8, q - x_7, q - x_6, q - x_1, q - x_4, q - x_2, q - x_3)$. Following [2], we could add constraints to ensure that the first solution is lexicographically less than the second. ECL^{i}PS^e allows this to be stated as a single constraint, which is equivalent to (and in other constraint programming systems would have to be expressed as) the set of constraints: $x_1 \leq q - x_5$; if $x_1 = q - x_5$ then $x_2 \leq q - x_8$; if $x_1 = q - x_5$ and $x_2 = q - x_8$ then $x_3 \leq q - x_7$; if $x_1 = q - x_5$ and $x_2 = q - x_8$ and $x_3 = q - x_7$ then $x_4 \leq q - x_6$. This is a cumbersome way of excluding just one symmetrically equivalent solution. Unless we can combine and simplify the constraints arising from different symmetries, which would require time and effort even if possible, we need one such set of constraints for every one of the 48 remaining symmetries.

To break these 48 symmetries in SBDS, we provide a symmetry function for each. As explained in [10], this effectively automates the conditional constraints and again gives independence of the variable order. GAP-SBDS, of course, does the same job, but using the group and not the individual elements.

Hence, for these graphs, it does not seem practicable to derive constraints to break the complement symmetries using the approach in [2]. In the next sections we consider further the role of symmetry constraints in graceful graphs.

5 GAP-SBDS *v.* Constraints

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

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

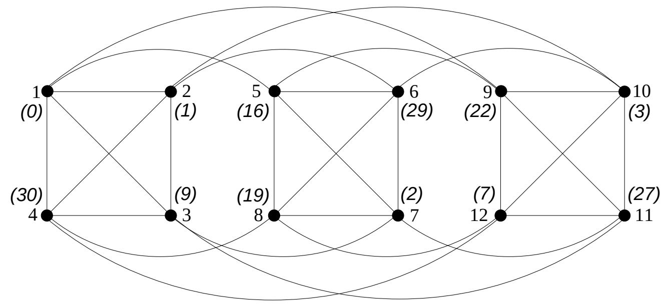

Fig. 3. The graph $K_4 \times K_3$.

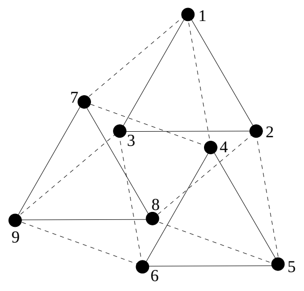

Fig. 4. The graph $K_3 \times K_3$.

ordered. In this case, $x_{12} = 30$ (the number of edges) and since there must be an edge joining the nodes labelled 0 and 30, either $x_4 = 0$ or $x_9 = 0$.

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

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

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

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

Table 2. Comparison of GAP-SBDS and symmetry constraints in proving that $K_3 \times K_3$ is not graceful.

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

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

Table 2 shows that with symmetry-breaking constraints, the least search is done when the maximum element of the matrix is constrained to be in the bottom-left corner, and that the bottom-right corner (as used for $K_4 \times K_3$) is the worst choice. It seems to us that this behaviour would be hard to predict, and that *a priori* any of the four choices seem reasonable. When GAP-SBDS breaks the same symmetry as the constraints, its performance is comparable: it does better than the worst choice of constraints, but worse than the best. When it breaks more symmetry than the constraints, i.e. when it is given the full symmetry group of 144 elements, or the subgroup of 72 elements, it does less search and is faster.

In the $K_4 \times K_3$ problem, similarly, GAP-SBDS breaks all the symmetry and the constraints only half. The experience with the $K_3 \times K_3$ graph suggests that this, rather than a conflict with the variable ordering, is the reason for the poor performance of the symmetry constraints relative to GAP-SBDS.

As mentioned in section 1, Gent *et al.* [8] compared GAP-SBDS and symmetry constraints in solving two BIBDs. As with the graphs considered in this section, BIBDs can be represented as matrices of variables with row and column symmetry. It is not usually possible to find simple constraints which are guaranteed to break all the row and column symmetry in a matrix; the graceful graph problems are an exception because of the requirement that all the values must be different. In the BIBDs, constraints were imposed to order the rows and columns lexicographically. Such constraints are not usually able to break all the symmetry, and from our results here we should therefore expect that GAP-SBDS would be faster. However, the constraints did break all the symmetry in the two BIBD instances considered. The problems were solved faster using the constraints than with GAP-SBDS, but it is not clear whether this is because the constraints did break all the symmetry in these instances, or whether the size of the symmetry group (millions of elements) also played a part. Further experiments are needed to see whether GAP-SBDS is in general faster than symmetry constraints when the latter do not break all the symmetry.

6 Double-wheel Graphs

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

We selected this class because we can break *all* the symmetry using simple constraints; and hence it allows us to compare GAP-SBDS and symmetry constraints in a problem class where we expect the constraints to do well. We believe that the gracefulness of these graphs has not previously been investigated: we have shown that DW_3 is not graceful, whereas DW_4 and DW_5 are, with 44 and 1216 non-isomorphic labellings respectively.

The two cycles, C_m, each have rotation and reflection symmetries. These can be eliminated by adding, for the cycle consisting of nodes 2, 3, 4, ..., $m+1$, the m constraints $x_2 < x_3$, $x_2 < x_4$, ..., $x_2 < x_{m+1}$ and $x_3 < x_{m+1}$. We add similar constraints for the second cycle. The labels of the two cycles can also be interchanged: we eliminate that symmetry by a constraint that the smallest node label in the first cycle is less than the smallest in the second. With the other constraints, this simplifies to $x_2 < x_{m+2}$.

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

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

Table 3 compares symmetry-breaking constraints and GAP-SBDS for these graphs. Constraints are the better choice, when the variables are assigned in the usual order $x_1, x_2, x_3, x_4, ...$. However, the difference in speed is not as great as might be expected, given that in DW_5, say, GAP-SBDS is handling a group of 400 elements, whereas only

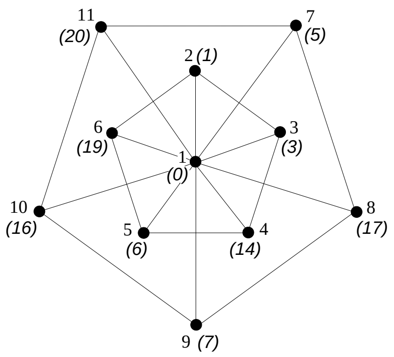

Fig. 5. The double wheel DW_5, with a graceful labelling.

		GAP-SBDS		Constraints	
Graph	Symmetries	BT	sec.	BT	sec.
DW_3	144	48	1.95	21	0.34
DW_4	256	1053	36.1	911	13.3
DW_5	400	33622	1609	26115	539

Graph	Ordering	BT	sec.	BT	sec.
DW_4	1,2,6,7,8,9,3,4,5	963	29.3	570	11.0
DW_4	1,2,3,4,5,6,7,8,9	1053	36.1	911	13.3
DW_4	2,3,4,5,6,7,8,9,1	1328	54.1	1190	49.6
DW_4	9,8,7,6,5,4,3,2,1	1328	53.0	1311	84.5

Table 3. Comparison of GAP-SBDS and symmetry breaking constraints for finding all graceful labellings of double-wheel graphs.

12 constraints are required for the same task. Clearly, GAP-SBDS is dealing with a large group very efficiently.

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

7 GAP-SBDD *v.* GAP-SBDS

As discussed earlier, SBDD and SBDS are related methods which break symmetry during search. This allows their GAP versions to use a similar interface and means that our GAP code which extends GAP-SBDS to make it easier to specify the symmetry group can also be used by GAP-SBDD. We have used the graceful graphs problems of Table 1 to compare GAP-SBDS and GAP-SBDD: the results are shown in Table 4.

Clearly, on these problems GAP-SBDD is performing very poorly in comparison to GAP-SBDS. Detailed examination of the respective search trees has shown that this is because the search variables of the CSP are the node variables, whereas the edge variables cause most constraint propagation. In GAP-SBDD, GAP returns just a boolean to indicate whether the current node is dominated or not, and possibly a list of values to prune from the domains of specified search variables. This successfully breaks the symmetry and prunes the search tree, but it does not provide any information that can propagate to the non-search variables, in this case the edge variables. On the other hand, GAP-SBDS posts constraints on backtracking, in order to break the symmetry, and these can propagate in the same way as any other constraint.

The fact that GAP returns limited information to ECL^{i}PS^e allows GAP-SBDD to solve much larger problems than GAP-SBDS, as found by Gent *et al.* in their BIBDs experiments [9]. However, our results have shown the disadvantage of this reduced communication. It is not unusual for CP modellers to develop CSPs like this, where only some of the variables are used for search, but constraint propagation over the full set of variables is crucial to solving the problem quickly. Hence, the fact that GAP-SBDD performs badly on such a model is an important finding.

	$K_3 \times P_2$				$K_4 \times P_2$				$K_5 \times P_2$			
	GAP-SBDS		GAP-SBDD		GAP-SBDS		GAP-SBDD		GAP-SBDS		GAP-SBDD	
Strategy	BT	sec.	BT	sec.	BT	sec.	BT	sec.	BT	sec.	BT	sec.
A	9	0.54	22	0.69	165	8.3	496	20.6	4390	382	17977	1310
B	16	0.59	50	1.33	369	14.3	1276	53.6	9889	793	51623	3910
C	16	0.58	24	0.63	369	14.1	473	16.5	9889	783	11710	859

Table 4. Comparison of different levels of symmetry breaking using GAP-SBDS or GAP-SBDD for finding all graceful labellings of $K_n \times P_2$.

8 Conclusions

We have carried out an experimental comparison of a number of symmetry-breaking techniques on graceful graph problems. These problems allow us to choose graphs with different symmetry. The symmetry of the graph also combines with the complement symmetry, which doubles the size of the symmetry group.

We have compared three techniques which break symmetry during search, namely SBDS, GAP-SBDS and GAP-SBDD. We extended GAP-SBDS to simplify the input of the symmetry group: this extension can be used for any other problem in which the symmetries affect just the variables (as the graph symmetries do) or in which some symmetries affect the variables and some the values (like the complement symmetries). It is also useful for GAP-SBDD.

In comparing SBDS and GAP-SBDS, our experiments with the $K_m \times P_l$ graceful graph problems have confirmed that increasing numbers of symmetries seriously affect the speed of SBDS. If the symmetry group is small enough (in one case, 120 elements), SBDS is faster than GAP-SBDS. However, since GAP-SBDS is slower on small problems, but still acceptable, whereas SBDS is unusable on large problems, GAP-SBDS is the better general choice, from the point of view of performance.

We have also compared GAP-SBDS with adding constraints to the CSP to eliminate symmetry. For these problems, if simple symmetry-breaking constraints can be devised to eliminate all the symmetry in a problem, they are faster than GAP-SBDS. However, if there is a choice between breaking some of the symmetry using constraints or breaking all of it using GAP-SBDS, then GAP-SBDS does less search and runs faster. This may be generally true when the constraints break at most half of the symmetry, as in the problems considered here. GAP-SBDS has the additional advantage of being independent of the search order; symmetry constraints are based on the assumption of a particular search order and deviating from that order can delay finding solutions.

Finally, we have compared GAP-SBDS and GAP-SBDD. The limited previous experience of comparing these techniques suggested that GAP-SBDD can handle much larger symmetry groups than GAP-SBDS. However, we have shown that GAP-SBDD is much worse than GAP-SBDS at solving the graceful graphs problems. We traced its difficulty to a feature of the CSP model: much of the constraint propagation involves non-search variables. Since this feature occurs quite frequently in the CSP models developed by expert modellers, it is a significant drawback to GAP-SBDD.

For these graceful graphs CSPs, the most successful symmetry-breaking techniques are GAP-SBDS and symmetry constraints. Where all the symmetry, including the complement symmetry, can be broken by simple constraints added to the CSP, this is the best approach to solving these problems. However, for many graphs this will not be possible, and then GAP-SBDS is the better choice.

With all of these symmetry-breaking methods, more user support is needed before they can be considered easy to use. GAP-SBDS and GAP-SBDD require some basic group theory, while deriving constraints to break symmetry often requires skill and experience. Writing symmetry functions for SBDS can be time-consuming, unless there is some way of automating the process; if based on GAP, this again requires some group theory. If the number of symmetry functions is small enough, however, they can be written by hand and this possibly requires less skill than the other methods. An underlying difficulty is that of identifying the symmetry of the problem: without this, symmetry breaking is clearly impossible. We plan to investigate automated symmetry detection in future; in principle, if symmetry detection can be automated, so too can symmetry breaking, and we plan to link it with GAP-SBDS and GAP-SBDD.

With good symmetry breaking, constraint programming is a valuable tool for finding all graceful labellings of symmetric graphs or proving that they are not graceful. This investigation has produced several new results on graceful graphs, and those for $K_m \times P_n$ graphs are included in the latest version of Gallian's survey [5].

Acknowledgments

The authors are members of the APES group (see http://www.dcs.st-and.ac.uk/˜apes) and would like to thank the other members; we should especially like to thank Ian Gent for his encouragement and interest. Tom Kelsey and Steve Linton have been very helpful in discussing the implementation of GAP-SBDD. We are also very grateful to Warwick Harvey for his help, including continuing advice and technical assistance with ECL^{i}PS^e. This work was supported by EPSRC grant GR/R29673.

References

1. R. Backofen and S. Will. Excluding Symmetries in Constraint-Based Search. In J. Jaffar, editor, *Principles and Practice of Constraint Programming - CP'99*, LNCS 1713, pages 73–87. Springer, 1999.
2. J. Crawford, M. Ginsberg, E. Luks, and A. Roy. Symmetry-Breaking Predicates for Search Problems. In *Proceedings KR'96*, pages 149–159, Nov. 1996.
3. T. Fahle, S. Schamberger, and M. Sellmann. Symmetry Breaking. In T. Walsh, editor, *Principles and Practice of Constraint Programming - CP 2001*, volume LNCS 2239, pages 225–239. Springer, 2001.
4. P. Flener, A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Breaking row and column symmetries in matrix models. In P. van Hentenryck, editor, *Principles and Practice of Constraint Programming - CP 2002*, LNCS 2470, pages 462–476. Springer, 2002.
5. J. A. Gallian. A Dynamic Survey of Graph Labeling. *The Electronic Journal of Combinatorics*, (DS6), 2002. http://www.combinatorics.org/Surveys.
6. The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.3*, 2002. http://www.gap-system.org.
7. I. Gent, S. Linton, and B. Smith. Symmetry Breaking in the Alien Tiles Puzzle. Report APES-22-2000, Oct. 2000. Available from http://www.dcs.st-and.ac.uk/˜apes/apesreports.html.
8. I. P. Gent, W. Harvey, and T. Kelsey. Groups and Constraints: Symmetry Breaking during Search. In P. van Hentenryck, editor, *Principles and Practice of Constraint Programming - CP 2002*, LNCS 2470, pages 415–430. Springer, 2002.
9. I. P. Gent, W. Harvey, T. Kelsey, and S. Linton. Generic SBDD with GAP and ECLiPSe. Technical Report APES-57-2003, APES Research Group, January 2003. Available from http://www.dcs.st-and.ac.uk/˜apes/apesreports.html.
10. I. P. Gent and B. M. Smith. Symmetry Breaking During Search in Constraint Programming. In W. Horn, editor, *Proceedings ECAI'2000*, pages 599–603, 2000.
11. W. Harvey. Symmetry Breaking and the Social Golfer Problem. In P. Flener and J. Pearson, editors, *Proceedings of SymCon'01: Symmetry in Constraints*, pages 9–16, 2001. Available from http://www.it.uu.se/research/group/astra/SymCon01/.
12. I. J. Lustig and J.-F. Puget. Program Does Not Equal Program: Constraint Programming and Its Relationship to Mathematical Programming. *INTERFACES*, 31(6):29–53, 2001.