

Comparison of Symmetry Breaking Methods in Constraint Programming

Karen E. Petrie and Barbara M. Smith

Cork Constraint Computation Center

University College Cork

Cork, Ireland

k.petrie@4c.ucc.ie, b.smith@4c.ucc.ie

Abstract

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

1 Introduction

Constraint Satisfaction Problems (CSPs) are often highly symmetric. Symmetries may be inherent in the problem, as in placing queens on a chess board that may be rotated and reflected. Additionally, the modelling of a real problem as a CSP can introduce extra symmetry: problem entities which are indistinguishable may in the CSP be represented by separate variables, leading to $n!$ symmetries between n variables. Symmetries may be found between variables or values or variable/value combinations. An example of value symmetry can be found in graph colouring problems where the different colours are interchangeable.

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

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

An alternative is to adapt the search algorithm so that constraints are added during search to prevent exploration of assignments. Symmetry Breaking During Search (SBDS), as can be seen in previous chapters, adds such constraints on backtracking. SBDS requires a function for each symmetry in the problem describing its effect on the assignment of a value to a variable. Although SBDS has been successfully used with a few thousand symmetry functions, many problems have too many symmetries to allow a separate function for each. To allow SBDS to be used in such situations Gent *et al.* [Gent *et al.*, 2002] linked SBDS (in ECLⁱPS^e) with GAP (Groups, Algorithms and Programming) [GAP, 2000], a system for computational discrete algebra and in particular computational group theory. GAP-SBDS allows the symmetry group rather than its individual elements, to be described.

GAP-SBDS allows the symmetry to be handled more efficiently than in SBDS; the elements of the group are not explicitly created, as is required in the original SBDS. On the other hand, GAP-SBDS has the overhead of the communication between ECLⁱPS^e and GAP. Furthermore, the symmetry-

breaking constraints posted on backtracking are constructed dynamically from the stabiliser rather than being pre-defined in the symmetry functions as in SBDS. It can be expected that GAP-SBDS will be a better choice than SBDS when the symmetry group is large, if only because it becomes impractical to list explicitly the individual elements of the group. However, for small symmetry groups, SBDS can be faster.

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

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

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

2 Graceful Graphs

A labelling f of the nodes of a graph with q edges is *graceful* if f assigns each node a unique label from $\{0, 1, \dots, q\}$ and when each edge xy is labelled with $|f(x) - f(y)|$, the edge labels are all different. (Hence, the edge labels are a permutation of $1, 2, \dots, q$.) Figure 1 shows an example. The study of graceful graphs is an active area of graph theory. Gallian [Gallian, 2004] surveys graceful graphs, i.e. graphs which have a graceful labelling, and lists the graphs whose status is known. [Gallian, 2004] is the latest version of a dynamic

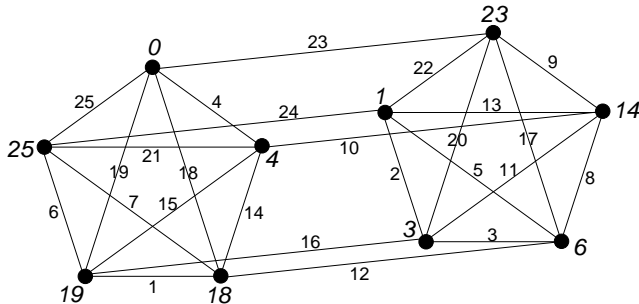


Figure 1: The unique graceful labelling of $K_5 \times P_2$.

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

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

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

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

2.1 Modelling Decisions

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

All Different

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

In accordance with these results we use the *global all_different* on the edge variables and the \neq version on the node variables. They are treated differently because the values assigned to the edge variables form a permutation and hence give more scope for domain pruning than the node variables, which have far more possible values than variables.

Version of <i>all_diff</i> constraint	bt	sec.
$\neq all_diff$ on nodes & $\neq all_diff$ on edges	3188	91.94
Global <i>all_diff</i> on nodes & $\neq all_diff$ on edges	3184	91.56
$\neq all_diff$ on nodes & Global <i>all_diff</i> on edges	147	12.53
Global <i>all_diff</i> on nodes & Global <i>all_diff</i> on edges	147	12.41

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

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

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

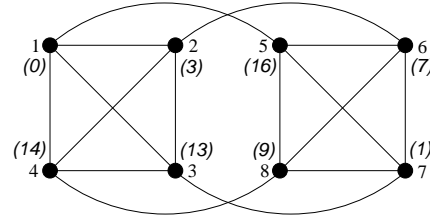


Figure 2: The graph $K_4 \times P_2$.

3 Search Decisions

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

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

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

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

The symmetries of the graph are, first, intra-clique permutations: any permutation of the 4-cliques which acts on both in the same way. For instance, we can transpose nodes 1 and 2 and simultaneously 5 and 6. Second, inter-clique permutations: the labels of the first clique (nodes 1, 2, 3, 4) can be

interchanged with the labels of the corresponding nodes (5, 6, 7, 8) in the second. These can also be combined with each other and with the complement symmetry. Hence, the size of the symmetry group is $4! \times 2 \times 2$, or 96.

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

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

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

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

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

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

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

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

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

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

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

For each problem, strategy C is faster for SBDS than for GAP-SBDS, showing that for a sufficiently small number of symmetries (120 for $K_5 \times P_2$) it is faster to have the individual symmetries expressed explicitly than as a group, and so avoid

the overheads of interacting with GAP. However, it does not depend solely on the size of the symmetry group: for $K_4 \times P_2$, strategy A is faster for GAP-SBDS than for SBDS, although there are only 96 symmetries.

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

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

5 Symmetry-Breaking Constraints

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

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

These constraints can be compared with those added during search by SBDS. When a symmetry function is had in SBDS for every graph symmetry (strategy B) the effect will be similar to adding a constraint to the model for every symmetry, and will result in many duplicated constraints added on

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

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

This symmetry transforms the general solution $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8)$ to $(q - x_5, q - x_8, q - x_7, q - x_6, q - x_1, q - x_4, q - x_2, q - x_3)$. Following [Crawford *et al.*, 1996], constraints could be added to ensure that the first solution is lexicographically less than the second. ECL²PS^e allows this to be stated as a single constraint, which is equivalent to (and in other constraint programming systems would have to be expressed as) the set of constraints: $x_1 \leq q - x_5$; if $x_1 = q - x_5$ then $x_2 \leq q - x_8$; if $x_1 = q - x_5$ and $x_2 = q - x_8$ then $x_3 \leq q - x_7$; if $x_1 = q - x_5$ and $x_2 = q - x_8$ and $x_3 = q - x_7$ then $x_4 \leq q - x_6$. This is a cumbersome way of excluding just one symmetrically equivalent solution. Unless we can combine and simplify the constraints arising from different symmetries, which would require time and effort even if possible, we need one such set of constraints for every one of the 48 remaining symmetries.

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

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

constraints in graceful graphs.

6 GAP-SBDS v. Constraints

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

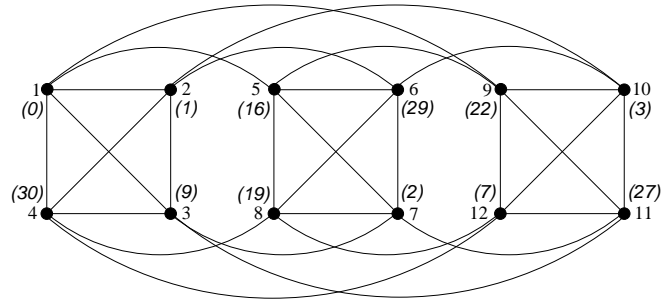


Figure 3: The graph $K_4 \times K_3$.

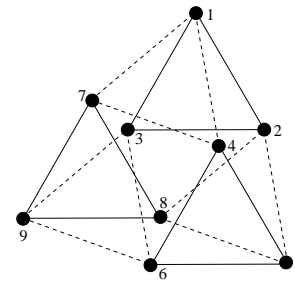


Figure 4: The graph $K_3 \times K_3$.

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

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

Finding all graceful labellings of this graph takes too long using ECL²PS^e, and we restricted the search to solutions in

which the edge with value 30 occurs in a 4-clique rather than a 3-clique. When using the symmetry constraints, this means that $x_9 = 0$, since $x_{12} = 30$. GAP-SBDS found the 17 non-isomorphic solutions in 10.6 hours runtime, but only 29 solutions had been found using the symmetry constraints after allowing 50% more time. (Since the complement symmetries are not being broken, there are 34 possible solutions.)

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

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

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

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

The row and column symmetries of the corresponding matrix are broken as for $K_4 \times K_3$, and then the results of constraining the maximum element to be in each of the four corners in turn are compared. (Instead a corner element could be forced to be the *minimum* element in the matrix, and change the ordering constraints on the row and column containing the corner appropriately. However, in ECL²PS^e this gives exactly

the same number of backtracks for this problem.)

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

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

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

7 Double-wheel Graphs

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

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

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

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

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

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

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

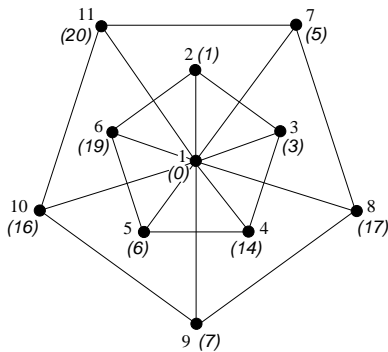


Figure 5: The double wheel DW_5 , with a graceful labelling.

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

Table 5 compares symmetry-breaking constraints and GAP-SBDS for these graphs. Constraints are the better choice, when the variables are assigned in the usual order

$x_1, x_2, x_3, x_4, \dots$. However, the difference in speed is not as great as might be expected, given that in DW_5 , say, GAP-SBDS is handling a group of 400 elements, whereas only 12 constraints are required for the same task. Clearly, GAP-SBDS is dealing with a large group very efficiently.

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

8 Finding One Solution

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

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

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

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

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

9 Conclusions

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

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

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

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

References

[Beutner and Harboth, 2002] D. Beutner and H. Harboth. Graceful labelings of nearly complete graphs. *Results in*

Mathematics, 41:34–39, 2002.

- [Crawford *et al.*, 1996] J. Crawford, M. L. Ginsberg, E. Luks, and A. Roy. Symmetry-breaking predicates for search problems. In Luigia Carlucci Aiello, Jon Doyle, and Stuart Shapiro, editors, *KR'96: Principles of Knowledge Representation and Reasoning*, pages 148–159. Morgan Kaufmann, San Francisco, California, 1996.
- [Flener *et al.*, 2002] P. Flener, A.M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Breaking row and column symmetries in matrix models. In P. Van Hentenryck, editor, *Proc. of CP'02*, LNCS 2470, pages 462–476. Springer, 2002.
- [Gallian, 2004] J. A. Gallian. A Dynamic Survey of Graph Labeling. *The Electronic Journal of Combinatorics (DS6)*, 2004. (<http://www.combinatorics.org/Surveys>).
- [GAP, 2000] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.2*, 2000. (<http://www.gap-system.org>).
- [Gent *et al.*, 2000] I.P. Gent, S.A. Linton, and B.M. Smith. Symmetry breaking in the alien tiles puzzle. Technical Report APES-22-2000, APES Research Group, October 2000. Available from <http://www.dcs.st-and.ac.uk/~apes/apesreports.html>.
- [Gent *et al.*, 2002] I. P. Gent, W. Harvey, and T. Kelsey. Groups and constraints: Symmetry breaking during search. In P. Van Hentenryck, editor, *Proc. of CP'02*, LNCS 2470, pages 415–430. Springer, 2002.
- [Henz, 2001] M. Henz. Scheduling a major college basketball conference—revisited. *Operations Research*, 49(1):163–168, 2001.
- [I.J.Lustig and Puget, 2001] I.J.Lustig and J.-F. Puget. Program Does Not Equal Program: Constraint Programming and Its Relationship to Mathematical Programming. In *INTERFACES*, volume 31(6), pages 29–53, 2001.
- [Puget, 2004] J.-F. Puget. Breaking symmetries in all different problems. In *Proceedings SymCon-04: Symmetry and Constraint Satisfaction Problems*, pages 71–78, 2004.