Proc. SymCon'03

Why SBDD can be worse than SBDS

Karen E. Petrie

School of Computing & Mathematics
University of Huddersfield
Huddersfield HD1 3DH, U.K.
k.e.petrie@hud.ac.uk

Abstract. A detailed comparison of GAP-SBDS and GAP-SBDD on the Graceful Graphs problem. GAP-SBDD performs substantially slower then GAP-SBDS on this problem class. This paper presents full analysis as to why this is the case.

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.

Definition of Symmetry *Given a CSP L, with a set of constraints C, a symmetry of L is a bijective function f which maps a representation of a search state α to another search state, so that the following holds:*

1. *If α satisfies the constraints C, then so does $f(\alpha)$.*
2. *Similarly, if α is a nogood, then so too is $f(\alpha)$. [12]*

Symmetries can give rise to redundant search since subtrees may be explored which are symmetric to subtrees already explored. To avoid this redundant search constraint programmers try to exclude all but one in each equivalence class of solutions. Many methods have been developed for this purpose. Two of these methods for symmetry exclusion which operate during search are, symmetry breaking during search (SBDS) [2, 9], and symmetry breaking via dominance detection (SBDD) [3, 4]. More recently, computational group theoretic versions of these methods have been devised, namely GAP-SBDS[7] and GAP-SBDD[8]. The development of these methods has led to the need for investigations into when each method should be applied. Petrie and Smith [13] carried out a comparison of GAP-SBDS and GAP-SBDD and found unexpected results. This paper presents a more detailed examination of how the two systems behave during search, on the problem studied, to clarify their behaviour.

2 Introduction to SBDD and SBDS

Symmetry breaking during search (SBDS), was developed by Gent and Smith [9], having been introduced by Backofen and Will [2]. The search tree is built from decision points, where a decision point has two possible choices; either assign a value to a variable, or do not assign that value to that variable. When a decision point is first reached during search a value is assigned to a variable; if at a later stage in search the decision point is revisited then a constraint is imposed that the variable should not have the previously assigned value. SBDS operates by taking a list of symmetry functions (provided by the user) and placing related constraints when backtracking to a decision point and taking the second branch.

A feature of SBDS is that it only breaks symmetries which are not already broken in the current partial assignment: this avoids placing unnecessary constraints. A symmetry is broken when the symmetric equivalent of the current partial assignment is not consistent with that assignment.

> **Definition of Broken Symmetry** *Given a partial assignment A, where $g(A)$ is the symmetric equivalent of A, then the symmetry g is broken if $g(A)$ is inconsistent with A.*

The following expression explains how SBDS works:

$$A \;\&\; g(A) \;\&\; var \neq val \;\Rightarrow g(var \neq val)$$

where A is the partial assignment made so far during search, $g(A)$ is the symmetric equivalent of A and $var \neq val$ is the symmetrical equivalent to this failed assignment. If A is the current partial assignment and we have established that $var \neq val$, we need to ensure that we are dealing with an unbroken symmetry, so we check that $g(A)$ still holds. Then to ensure that the symmetrically equivalent subtree to the current subtree will not be explored, the constraint $g(var \neq val)$ is placed. An SBDS library is now available in the ECL^{i}PS^e constraint programming system [1]. As previously mentioned, SBDS requires a function for each symmetry in the problem describing its effect on the assignment of a value to a variable. If these symmetry functions are correct and complete, all the symmetry will be broken; as a result of this only non-isomorphic solutions will be produced. 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 situations where there are too many symmetries to allow a function to be created for each, Gent *et. al.* [7] have linked SBDS in ECL^{i}PS^e with GAP (Groups, Algorithms and Programming) [6], a system for computational algebra and in particular *computational group theory*. Group theory is the mathematical study of symmetry. GAP-SBDS allows the symmetry group rather than its individual elements to be described. GAP is used when a value is assigned to a value at a decision point to find the *stabiliser* of the current partial assignment, i.e. the subgroup which leaves it unchanged. Then if the decision point is revisited on backtracking, the constraints are dynamically

calculated from the stabiliser and placed accordingly. GAP-SBDS allows the symmetry to be handled more efficiently than in SBDS; the elements of the group are not explicitly created which is akin to what the symmetry functions represent in SBDS. However, there is an overhead in communication necessitated between GAP and ECL^{i}PS^e.

Symmetry Breaking via Dominance Detection (SBDD) [3, 4] performs a check at every node in the search tree to see if it is dominated by a symmetrically equivalent subtree already explored, and if so prunes this branch. Harvey [10] 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 hard-coded for each problem. This means in practise SBDD can be difficult to implement, as the design of the dominance detection function may be complicated.

Gent *et. al.* [8] 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 the details of that node to GAP, and GAP returns false if dominance has been detected and that branch can be pruned, or true otherwise. Occasionally full dominance is not detected but there are variable/value pairs which are easily detected as being eligible for domain deletion; at which point GAP returns true followed by a list of variable/value pairs for which this is the case. This information is utilised by ECL^{i}PS^e to control search. This communication overhead between GAP and ECL^{i}PS^e, required in returning a true or false answer, is less than that of GAP-SBDS where fairly complicated constraints have to be passed. Gent *et. al.* [8] compared GAP-SBDD with GAP-SBDS applied to instances of the balanced incomplete block design (BIBD) problem [1] and showed that GAP-SBDD could solve much larger problems, and was faster than GAP-SBDS on the smaller problem which both could solve.

3 Test Problem: Graceful Graphs

Petrie and Smith [13] investigated symmetry breaking in the *Graceful Graphs* problem. A labelling f of the vertices of a graph with q edges is *graceful* if f assigns each vertex with a unique label from $\{0, 1, ..., q\}$ and each edge xy is labelled with $|f(x) - f(y)|$ the edges are all different [5]. (Hence, the edge labels are a permutation of $1, 2, ..., q$.) Figure 1 shows two examples. Gallian [5] gives a survey of graceful graphs, i.e. graphs with a graceful labelling, and lists graphs whose status is known. Lustig and Puget [11] give a constraint model for finding a graceful labelling of the graph $K_4 \times P_2$.

A basic 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:

[1] BIBD is problem 28 of csp-lib `http://www.csplib.org`

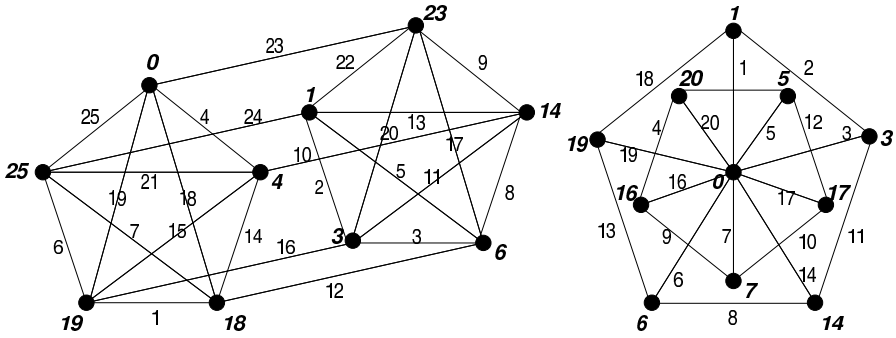

Fig. 1. Graceful Labellings of $K_5 \times P_2$ and the Double Wheel DW_5

1. If edge k joins nodes i and j then $d_k = |x_i - x_j|$.
2. $x_1, x_2, ..., x_n$ are all different.
3. $d_1, d_2, ..., d_q$ are all different.

ECL^{i}PS^e provides two different levels of propagation for the *alldifferent* constraint. It can either be treated as a set of binary $\neq$ constraints or as a *global alldifferent* which has higher propagation. We use the *global alldifferent* on the edge variables and the binary $\neq$ 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 more possible values than variables. The node variables are used as the search variables as they are the simplest set to consider symmetry breaking over.

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 $x_1, x_2, ..., x_n$ can be reversed to give an equivalent labelling $x_n, ..., x_2, x_1$. We might call this the graph symmetry. 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 graph symmetry with the complement symmetry.

4 Tests Undertaken: $K_m \times P_2$

The graph $K_5 \times P_2$, shown in figure 1, consists of two copies of K_5, with corresponding vertices in the two cliques forming the vertices of path P_2. The symmetries of $K_5 \times P_2$ are first any permutation of the 5-cliques which act on both in the same way. Second, inter-clique symmetry: all the node labels in the first clique can be interchanged with the labels of the adjacent nodes in the second. These can be combined with each other and with the complement symmetry. Hence, the size of the symmetry group is $5! \times 2 \times 2$. In general $K_m \times P_2$ graphs

have a symmetry group of size $m! \times 2 \times 2$. In this study we will concentrate on symmetry breaking in 3 such graphs, namely $K_3 \times P_2$, $K_4 \times P_2$ and $K_5 \times P_2$. The results of finding all graceful labellings of these graphs using either GAP-SBDS or GAP-SBDD can be found in table 1.

	GAP-SBDD				GAP-SBDS			
	BT	ECL^{i}PS^e time	GAP time	Total time	BT	ECL^{i}PS^e time	GAP time	Total time
$K_3 \times P_2$	22	0.30	0.39	0.69	9	0.20	0.34	0.54
$K_4 \times P_2$	496	16.65	3.97	20.62	165	7.00	1.32	8.32
$K_5 \times P_2$	17997	1106	204	1311	4390	344.73	37.69	382.42

Table 1. Comparison of GAP-SBDS and GAP-SBDD showing backtracks (bt) and the time (in seconds) for finding all graceful labellings of $K_3 \times P_2$, $K_4 \times P_2$, $K_5 \times P_2$. On a 1.6 GHz Pentium 4 processor with 512MB of memory.

From table 1 we can see that GAP-SBDD is substantially slower than GAP-SBDS for all instances.[2] This contradicts the results reported by Gent *et. al.* in [8]. We performed further analysis to investigate why the results differ. It is also worth noting from the above results that both the GAP times and the ECL^{i}PS^e times are longer, for each graph, in GAP-SBDD than in GAP-SBDS. This shows that it is not one or other of these processes which is causing the delay, but rather a combined effect.

5 Analysis

We have analysed the difference between GAP-SBDS and GAP-SBDD for the three graphs $K_3 \times P_2$, $K_4 \times P_2$ and $K_5 \times P_2$. The reason for the different times is consistent, but for reasons of simplicity only the results for $K_3 \times P_2$ are presented here. A diagram of $K_3 \times P_2$ can be seen in figure 2, for ease of reference the node variables are named in capital letters, and the edge variables with a letter pairing corresponding to their attached nodes.

We began analysing where GAP-SBDS and GAP-SBDD differ, by finding where the first difference in the search tree occurs. In finding all graceful labellings of $K_3 \times P_2$ this actually happens quite late in the search, after the first two solutions (from a possible 4) have been found. A full diagram of the search tree until this point can be found in the appendix. Of note is the fact that the backtrack count in table 1 refers to *deep-backtracks*. A deep-backtrack is when the search has moved passed a point it later has to revisit. A *shallow-backtrack* is where the $var = val$ branch has been tried, and due to propagation of that choice

[2] The results presented do seem to continue to larger problem instances. A solution was found for $K_6 \times P_2$ within 12 hours using GAP-SBDS, but no result was returned in a similar amount of time by GAP-SBDD.

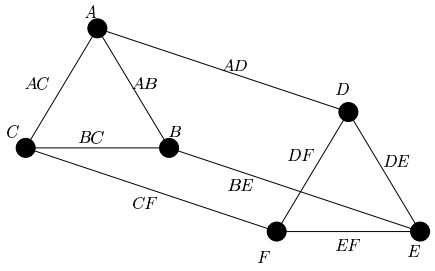

Fig. 2. Graph $K_3 \times P_2$ showing node variable and edge variable naming

reversed in favour of the $var \neq val$ branch. The number of deep-backtracks is the standard backtrack count in most constraint programming environments, so eases the comparison of methods across environments, but perhaps in this case it does not show the most accurate picture of a search tree. In the fragment of the search tree shown there are only 2 deep-backtracks counted, but the $var \neq val$ branch is actually followed 18 times. This is important when studying GAP-SBDS, as every time the $var \neq val$ branch is followed, symmetry breaking constraints can be placed.

Looking more closely at the branch of the search tree where the first difference occurs (this can be found in figure 3) shows that GAP-SBDS enables earlier pruning than GAP-SBDD. This pruning happens after setting $C = 5$. GAP-SBDS immediately reverses from this decision to follow the $C \neq 5$ branch, whereas GAP-SBDD carries on from here to set $E = 1$, and ends up performing a deep-backtrack back to this point later on in search.

To understand why this difference occurs we have to look into the variable propagation. In the first instance both GAP-SBDS and GAP-SBDD perform propagation over the current partial assignment, in conjunction with the problem constraints, to get to the stage shown in figure 4. Past this point GAP-SBDS, propagates any symmetry breaking constraints previously employed on this branch. The two that are vital in this case are a combination of the graph symmetry and the complement symmetry for $B \neq 1$: namely $E \neq 8$ and $F \neq 8$. These extra constraints are placed on the node variables, but they provide extra information for propagation to occur on the edge variables as well. As we have already discussed, the values assigned to the variables form a permutation giving the *alldifferent* constraint more scope for pruning. This added propagation causes $C = 5$ to fail and the alternative path to be followed.

In contrast to this, GAP-SBDD just returns a boolean to indicate whether the current node is dominated or not, and possibly a list of values to prune from the domains of specific search variables. In the current implementation, variable/value pairs are returned for domain pruning, when the variable is the only one in the current partial assignment not to cause domination to be detected. So in this case $E/8$ and $F/8$ are not returned. This successfully breaks

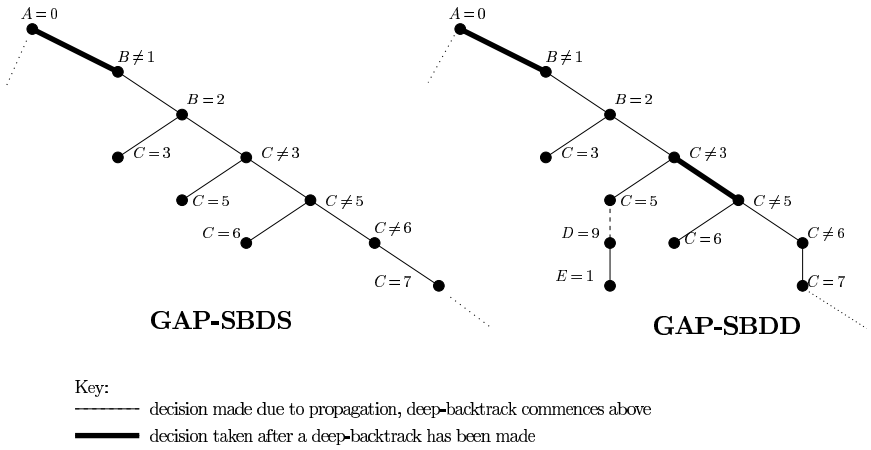

Fig. 3. The search tree branch where GAP-SBDS and GAP-SBDD differ

the symmetry and prunes the search tree, but it does not provide information that can propagate on any non-search variables, in this case the edge variables.

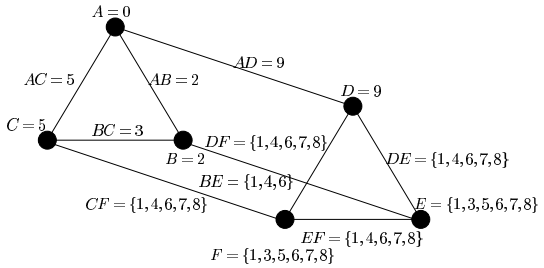

Fig. 4. The domain of the node and the edge variables after propagating $C = 5$

6 Conclusion

The fact that GAP-SBDD returns limited information to ECL$^{\text{i}}$PS$^{\text{e}}$ allow GAP-SBDD to solve much larger problems than GAP-SBDS, as found by Gent *et. al.* in their BIBDs experiments [8]. However, our results have shown the disadvantage of this reduced communication. We have found that the constraints which GAP-SBDS places during search, are adding information to the problem, and for problems this can cause an increase in propagation. In this case the propagation

of the non-search variables with these constraints gives scope for domain pruning of the search variables. It is not unusual for CP modellers to develop CSPs like graceful graphs, where only some of the variables are used for search, but constraint propagation over the full set of search variables is crucial to solving the problem quickly. Hence the fact that GAP-SBDD may performs badly on such a model is an important finding.

In general, we have shown that simple empirical comparisons of symmetry breaking methods are not always enough to give an accurate guide to which method should be used when. When thinking about how a symmetry breaking method may operate on a given problem, it is valuable to consider how the construction of the problem and the symmetry breaking method may interact.

Acknowledgements

The author is a member of the APES group and would like to thank the other members; I should especially like to thank Barbara Smith and Ian Gent for their interest. Tom Kelsey and Steve Linton were very helpful in confirming some of the data of this study. I am also very grateful to Warwick Harvey for his ongoing encouragement and Neil Yorke-Smith for his support. This work was supported by grant GR/R29673.

References

1. The ECLiPSe Constraint Logic Programming System, Mar. 2002. (`http://www-icparc.ic.ac.uk/eclipse/`).
2. R. Backofen and S. Will. Excluding symmetries in constraint-based search. In J. Jaffar, editor, *Principles and Practice of Constraint Programming - CP99*, volume LNCS 1713, pages 73–87. Springer, 1999.
3. T. Fahle, S. Schamberger, and M. Sellmann. Symmetry breaking. In T. Walsh, editor, *Principles and Practice of Constraint Programming*, volume LNCS 2239, pages 93–107. Spinger, 2001.
4. F. Focacci and M. Milano. Global cut framework for removing symmetries. In T. Walsh, editor, *Principles and Practice of Constraint Programming*, volume LNCS 2239, pages 77–92. Spinger, 2001.
5. J. Gallian. A Dynamic Survey of Graceful Labeling. In *The Electronic Journal of Combinatorics*, 2002. (`http://www.combinatorics.org/Surveys`).
6. The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.2*, 2000. (`http://www.gap-system.org`).
7. I. P. Gent, W. Harvey, and T. Kelsey. Groups and constraints: Symmetry breaking during search. In P. V. Hentenryck, editor, *Proceedings of Principles and Practice of Constraint Programing- CP02*, volume LNCS 2470, pages 415–430. Spinger, 2002.
8. 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`).
9. I. P. Gent and B. M. Smith. Symmetry breaking in constraint programming. In *Proceedings of ECAI-2002*, pages 599–603. IOS press, 2000.

10. W. Harvey. Symmetry Breaking and the Social Golfer Problem. In *Proceedings SymCon-01: Symmetry in Constraints*, pages 9–16, 2001.
11. 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.
12. I. McDonald and B. M. Smith. Partial symmetry breaking. In P. V. Hentenryck, editor, *Proceedings of Principles and Practice of Constraint Programing- CP02*, volume LNCS 2470, pages 431–445. Spinger, 2002.
13. K. Petrie and B. Smith. Symmetry breaking in graceful graphs. Technical Report APES-56a-2003, APES Research Group, University of Huddersfield, June 2003. Available from `(http://www.dcs.st-and.ac.uk/~apes/apesreports.html)` To appear in Proceedings Principles and Practice of Constraint Programming 2003. (Springer, ed Francesca Rossi).

Appendix

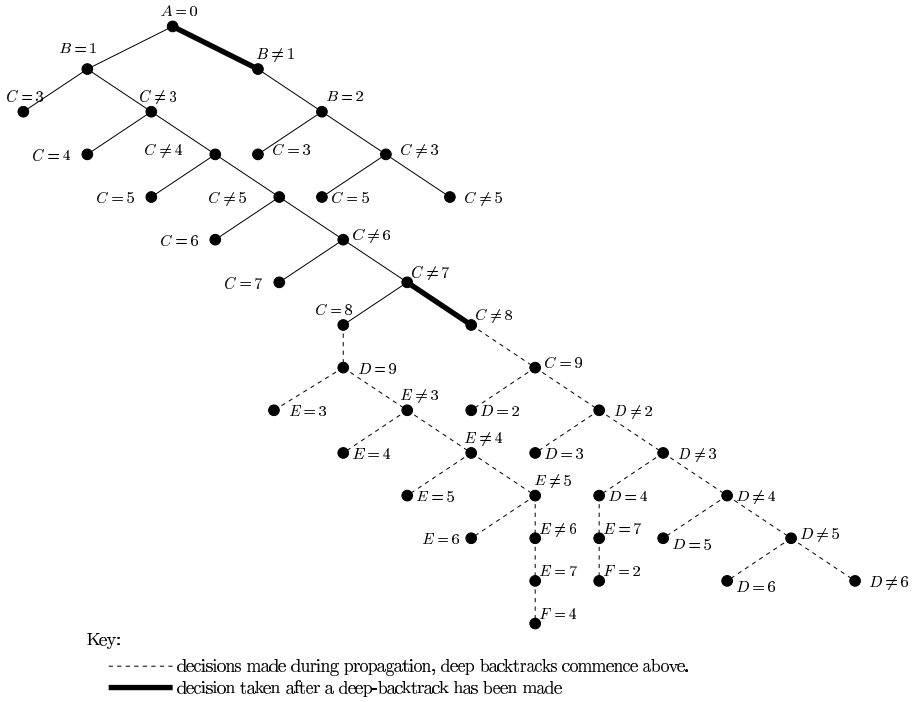

Fig. 5. Search tree for $K_3 \times P_2$ to point where GAP-SBDS and GAP-SBDD differ