

Combining SBDS and SBDD

Karen E. Petrie

School of Computing and Engineering
University of Huddersfield
Huddersfield, HD1 3DH, UK
k.e.petrie@hud.ac.uk

Abstract

We introduce two hybrid methods of GAP-SBDS and GAP-SBDD; *SBDS+D* and *SBDD+S*. The implementation of these hybrids comes from a comparison of GAP-SBDS and GAP-SBDD, which concludes that combining the methods may produce a more efficient symmetry breaking method than either of the two methods alone. SBDS+D allows the user to change from SBDS to SBDD, at any depth in the search tree, SBDD+S allows the opposite change from SBDD to SBDS. Experimentally we show that SBDD+S is an incomplete symmetry breaking method, which is less efficient than either GAP-SBDS or GAP-SBDD alone. Before presenting SBDS+D, which is a complete symmetry breaking method; that can outperform both GAP-SBDS and GAP-SBDD on a range of problems with different types of symmetry.

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 no-good, then so too is $f(\alpha)$. [13]

Symmetries can give rise to redundant search, while searching for solutions a partial assignment may be considered which is symmetric to one previously examined. If a partial assignment does not lead to a solution, neither will any symmetric assignment, and if it does lead to a solution, the new

solution is symmetrically equivalent to one already found. 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 [1; 10], and symmetry breaking via dominance detection [3; 5]. More recently, computational group theoretic versions of these methods have been devised, namely GAP-SBDS [8] and GAP-SBDD [9].

Symmetry breaking during search (SBDS), was developed by Gent and Smith [10], having been introduced by Backofen and Will [1]. 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. 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 ECLⁱPS^e constraint programming system [2]. 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.* [8] have linked SBDS in ECLⁱPS^e with GAP (Groups, Algorithms and Programming) [7], a system for computational algebra and in particular *computational group theory* (CGT). 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 variable, 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 ECLⁱPS^e.

Symmetry Breaking via Dominance Detection (SBDD) [3; 5] 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. In SBDD, the dominance detection function is based on the problem symmetry and is hard-coded for each problem. This means in practice SBDD can be difficult to implement, as the design of the dominance detection function may be complicated; the user has to ensure that all the symmetry of the problem is incorporated within the function to enforce full symmetry breaking.

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 ECLⁱPS^e) with GAP. At each node in the search tree, ECLⁱPS^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. ECLⁱPS^e removes these values from the corresponding variables domains before search continues.

In this paper we compare GAP-SBDS and GAP-SBDD using ‘Graceful Graphs’ as our motivating example. This study leads us to an analysis of exactly how the methods differ. We use this analysis to create hybrid methods of GAP-SBDS and GAP-SBDD. Before comparing our new SBDS+D and SBDD+S methods to both GAP-SBDS and GAP-SBDD, producing favourable results in the case of SBDS+D.

2 Comparison of SBDS and SBDD

Limited past work has been conducted comparing GAP-SBDS and GAP-SBDD. Harvey [11] studied the algorithms theoretically to draw the conclusion that SBDS and SBDD are closely related; the difference is where in the search tree

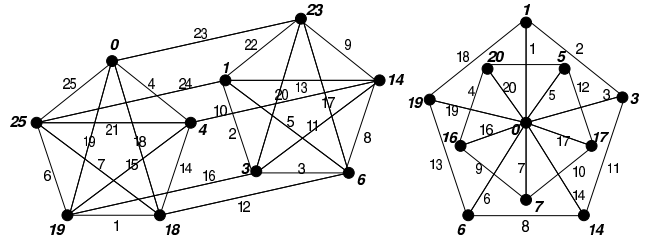


Figure 1: Graceful Labellings of $K_5 \times P_2$ and the Double Wheel DW_5

and how, symmetry breaking is enforced. Gent *et. al.* [9] compared GAP-SBDS and GAP-SBDD applied to instances of the balanced incomplete block design (BIBD) problem and showed that GAP-SBDD could solve much larger problems, and was faster than GAP-SBDS on the smaller problems which both could solve. They surmised this was due to the communication overhead between GAP and ECLⁱPS^e, the overhead in GAP-SBDD in only returning a true or false answer, is less than that of GAP-SBDS where fairly complicated constraints have to be placed. Petrie and Smith [15] investigated symmetry breaking in the *Graceful Graphs* problem, where they found GAP-SBDS to outperform GAP-SBDD on all instances. In this section we perform further analysis on this problem.

2.1 Graceful Graphs

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 $\{0, 1, \dots, q\}$ and each edge xy is labelled with $|f(x) - f(y)|$ the edges are all different [6]. (Hence, the edge labels are a permutation of $1, 2, \dots, q$.) Figure 1 shows two examples. Lustig and Puget [12] 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, \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:

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

ECLⁱPS^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. More information on the modelling of this problem and the symmetry group can be found in [15].

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 $K_5 \times P_2$

		BT	ECL ² PS ^e time	GAP time	Total time
GAP-SBDD	$K_3 \times P_2$	13	0.23	0.50	0.73
	$K_4 \times P_2$	173	7.18	2.72	9.90
	$K_5 \times P_2$	4402	337.69	88.20	426.89
GAP-SBDS	$K_3 \times P_2$	9	0.20	0.33	0.53
	$K_4 \times P_2$	165	7.15	1.35	8.50
	$K_5 \times P_2$	4390	352.10	36.61	388.71

Table 1: Comparison of GAP-SBDS and GAP-SBDD showing backtracks (bt) and the time (in seconds) for finding all graceful labellings of $K_3 \times P_2$, $K_4 \times P_2$, $K_5 \times P_2$.

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. Third, complement symmetry: we can replace every vertex label x_i by its complement $n - x_i$. These two graph symmetries and the complement symmetry can be combined with each other. 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. All experiments throughout this paper are run on a 1.6GHz Pentium 4 processor with 512MB of memory, using ECL²PS^e version 5.7 and GAP version 4.2. Analysing the results in table 1 we can see that GAP-SBDD is slower than GAP-SBDS for all instances.¹ Experiments have shown these results to be general for all graphs, these results are documented in [14].

2.2 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.

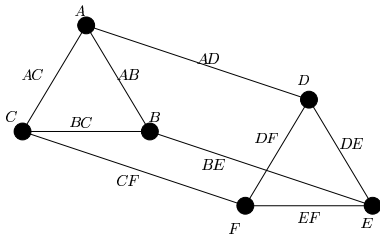


Figure 2: Graph $K_3 \times P_2$ showing node variable and edge variable naming

¹These results differ from those presented in [14] as GAP-SBDD has since been updated for efficiency and now provides more domain removals

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. 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 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. This is important when studying GAP-SBDS, as every time the $var \neq val$ branch is followed, symmetry breaking constraints can be placed.

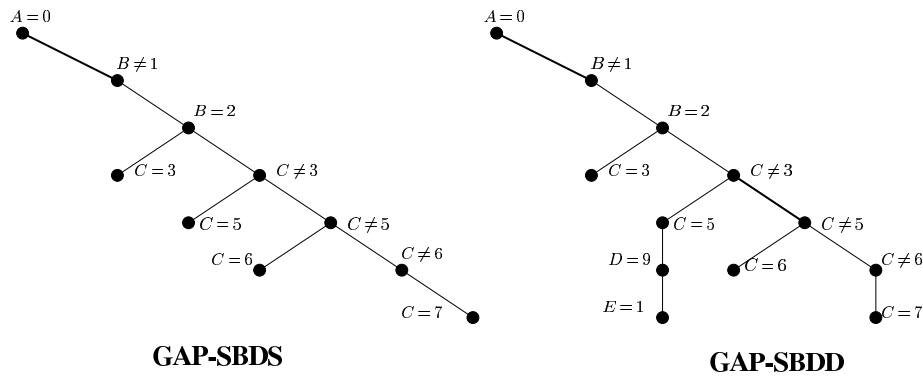
Looking more closely at the branch of the search tree where the first difference occurs (this can be found in figure 3) shows that GAP-SBDS enables earlier pruning than GAP-SBDD. This pruning happens after setting $C = 5$. GAP-SBDS immediately reverses from this decision to follow the $C \neq 5$ branch, 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. 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 these 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, the only variable/value pairs returned for domain pruning are, 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 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. Further details of these experiments and analysis can be found in [14].

3 Combining SBDS and SBDD

The fact that GAP-SBDD returns limited information to ECL²PS^e allows GAP-SBDD to solve much larger problems than GAP-SBDS as found by Gent *et. al.* in their BIBD experiments [9]. However our experiments and analysis have shown the disadvantage of this reduced communication. We



Key:
 — decision made due to propagation, deep-backtrack commences above
 - - - a deep-backtrack was made at this point

Figure 3: The search tree branch where GAP-SBDS and GAP-SBDD differ

have found that the constraints which GAP-SBDS places during search are adding information to the problem, and for some problems this can cause an increase in propagation. These observations led us to implement hybrid algorithms SBDS+D and SBDD+S, which combine the advantages of both methods.

When using either GAP-SBDS or GAP-SBDD, the methods are called at each node on the search tree, so the simplest way to combine the methods is to have different methods called at different nodes. This avoids duplicating any of the symmetry breaking effort. In SBDS+D we perform SBDS to a given depth in the search tree before switching to SBDD, in SBDS+D SBDD proceeds SBDS. This allows us to use SBDS at either the root or leaf nodes long enough to aid propagation before or after switching to/from the more efficient SBDD. In order to implement this change, two further parameters *method* and *level* are added to the symmetry breaking module. The *method* (either SBDD or SBDS) is started from the depth indicated by *level* in the search tree, with the other method operating until this level i.e. *method* = SBDD and *level* = 1 would perform SBDD on the entire search tree, *method* = SBDD and *level* = 2 would perform SBDS+D switching to SBDD from level 2. Figure 4 shows how this switch works at some other depths in the search tree.

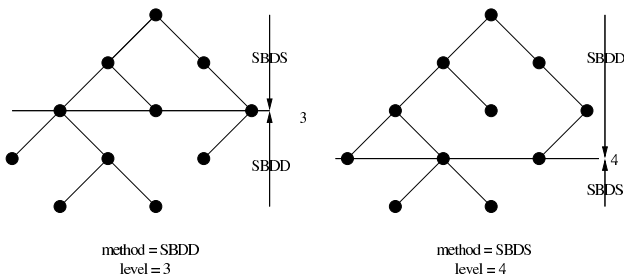


Figure 4: Various arrangements of *method* and *level* using algorithm SBDS+D and an SBDD+S respectively

Both the SBDS+D and the SBDD+S methods called at each node in search, operates the correct symmetry breaking method at the current depth. GAP-SBDS and GAP-SBDD operate on different properties of the symmetry group, we update the group information for both properties at each node. This allows the methods to be interchanged at a later point in search without the overhead of calculating symmetry data on any nodes visited previously on the search tree, but does provide an overhead over GAP-SBDS or GAP-SBDD alone. This interchanging can be done on backtracking, if we backtrack back past *level* as well as while forward searching. Throughout this paper we refer to the components of SBDS+D and SBDD+S as SBDS or SBDS as although they both utilise group theory, they no longer completely follow the GAP-SBDS and GAP-SBDD algorithms. The algorithm for this method is outlined in full.

A point to note (from the algorithm) is that in SBDS asserting $var = val$ does not necessitate any symmetry breaking as it is all performed on the $var \neq val$ branch, whereas in SBDD both the $var = val$ and $var \neq val$ cases are checked for domination. This means that SBDD works actively to break symmetry at all nodes, whereas SBDS just operates actively on a small subset.

4 SBDD before SBDS

The SBDD+S algorithm is an incomplete symmetry breaking method, by this we mean that it does not return only the non-isomorphic solutions. Performing SBDD at the top of the tree, performs a check to see if any of the nodes previously visited dominate the current node. At the required level in the search tree SBDS places constraints on backtracking to break the symmetry after the current depth. This combination does not give complete symmetry breaking as there is no mechanism to stop branches symmetric to those explored when performing SBDD, being chosen later in the search tree where SBDS is the chosen method. This method is a partial symmetry breaking method. McDonald and Smith [13] performed experiments using a subset of the total symmetries for each

```

SBDS+D;D+S(A, Var, Val, Method, Level)
assert(Var = Val)
Depth is Current_Depth + 1
update_sbds_info(Var, Val) in GAP
update_sbdd_info(Var, Val) in GAP
if ((Method = SBDS and Level ≥ Depth) or
(Method = SBDD and Level < Depth)) then
if Var = Val is true then
no symmetry breaking
else
retract(Var = Var)
get non_broken_symms from GAP
for g in non_broken_symms do
assert(g(A) ⇒ g(var ≠ val))
end do
end if
else
get is_node_dominated(Bool, Removals) from GAP
if ((Var = Val is true) and (Bool = false)) then
reduce_domains(Removals)
dominance_check_partial_assignment
else if ((Var = Val is false) and (Bool = false)) then
retract(Var = Val)
reduce_domains(Removals)
dominance_check_partial_assignment
else
dominance detected so backtrack
end if
end if
end if

```

The SBDS+D and SBDD+S algorithm

problem with SBDS. They found that each symmetry has an overhead, so there is an optimum number to place for symmetry breaking in order to get the maximum gain in efficiency to solve the problem. In our method we are placing less symmetry breaking constraints than if we were performing pure SBDS, but still place a subset which will aid propagation.

We have undertaken experiments on the graceful graphs problem, to see how our method performs. Plots of experimental data showing number of solutions against level of change from SBDD to SBDS, indicate the amount of symmetry broken at each level. Plots of backtracks against level of change, indicate the search effort associated with each level. This data is found in figure 5 and figure 6 for graceful labellings of $K_3 \times P_2$ and $K_4 \times P_2$ respectively.

The data point at *level* = 1 on the plots shows the results for performing SBDS on the entire tree. Studying the first plot in both figures 5 and figure 6 we can see that $K_3 \times P_2$ has 4 and $K_4 \times P_2$ has 15 non-isomorphic solutions. Performing SBDD from *level* = 2 causes a dramatic rise in the number of solutions, which peaks at *level* = 3 before slowly decreasing. This dramatic rise is due to the fact that if using GAP-SBDS when backtracking past the root node in a search tree none of the symmetry is broken in the current partial assignment so a symmetry constraint is placed for every one of the problem symmetries, which can rule out a vast number of branches from being followed later in search. The slow decrease after the plots peak is because SBDD is conduct-

ing symmetry breaking on more of the tree as the level which SBDS enters at decreases, until a level which finds only the non-isomorphic solutions is reached. At this level, which is lower for $K_4 \times P_2$ than $K_3 \times P_2$ due to the increased number of search variables, the depth in the search where SBDS would be triggered is never reached so full symmetry breaking is conducted by SBDD.

Studying the second graph in both figure 5 and figure 6 we see that the number of backtracks, and hence the amount of search undertaken, rises in proportion to the number of solutions at each level. The total running time also rises in proportion to the number of backtracks. Using SBDD+S never performs better than GAP-SBDD or GAP-SBDS alone. These results have been verified on other graceful graphs and on other problem classes.

5 SBDS before SBDD

The SBDS+D algorithm is a complete symmetry breaking method. SBDS operates at the top of the tree at and near the root node, at this level few decisions have been undertaken which could break symmetry; so if we backtrack past these nodes constraints to break a large percentage of the total problem symmetries can be placed. The propagation of these constraints can reduce branches being explored in the search tree, hence they can cut down the number of failed variable to value assignments. It is worth noting here that SBDD only returns candidate values for domain removal for decisions that if were to be the next decision in the search tree would cause dominance to be detected. Hence early in the search tree there

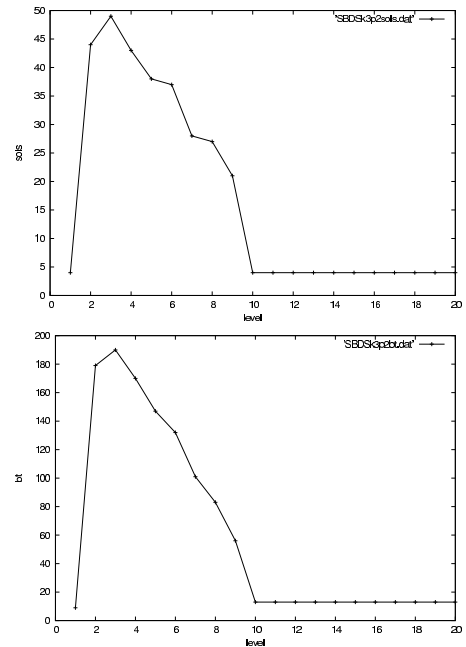


Figure 5: Plots of $K_3 \times P_2$ showing depth of change from SBDD to SBDS against number of solutions and number of backtracks respectively.

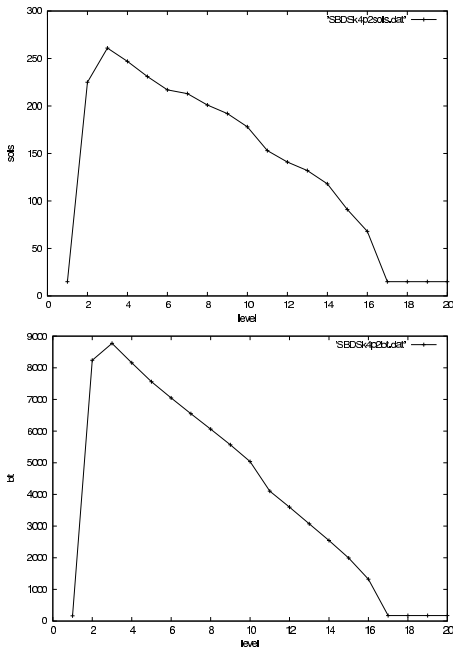


Figure 6: Plots of $K_4 \times P_2$ showing depth of change from SBDD to SBDS against number of solutions and number of backtracks respectively.

are unlikely to be any of these domain removal values to be returned, so the use of SBDD at this point would not aid constraint propagation. Later in the tree we change from SBDS to SBDD, which performs the faster dominance check at each node. In doing so SBDD checks all solutions for dominance by a previous solution, ensuring only the non-isomorphic solutions are returned.

We have performed experiments with SBDS+D on various problems, with various ‘types’ of symmetry groups. The results of these experiments have shown that SBDS+D, can do better than GAP-SBDD or GAP-SBDS alone.

5.1 Graceful Graphs

The graceful graphs problem, as we have shown, is a problem where GAP-SBDS operates more efficiently than GAP-SBDD. This is due to the fact that the search variables are not the most constrained variables, and the symmetry breaking constraints placed by SBDS propagate with these non-search variables. GAP-SBDD performs a more efficient symmetry check at each node than GAP-SBDS. So we experimented with SBDS+D to see whether the extra propagation of SBDS combined with SBDD will produce a better symmetry breaking method for this class of problems. We are also interested to see at which level this will happen. Figure 7 and figure 8 show plots of $K_3 \times P_2$ and $K_4 \times P_2$ respectively showing backtracks and time against level of change from SBDS to SBDD.

The data point at $level = 1$ on the plot represents performing GAP-SBDD on the entire search tree. Studying the first plot in both figure 7 and figure 8, we see the backtrack

count for GAP-SBDD alone is 13 and 173 for $K_3 \times P_2$ & $K_4 \times P_2$ respectively; at $level = 2$ this falls to 9 & 163 which it then remains at for all subsequent levels. Using GAP-SBDS $K_3 \times P_2$ & $K_4 \times P_2$ have 9 and 163 backtracks respectively. By placing SBDS constraints at the root node the search effort of SBDS+D is equal to that of GAP-SBDS. Turning to the time plots; the time for GAP-SBDD alone is higher than that for any level of SBDS+D for both graphs labellings. The time falls dramatically from $level = 1$ to $level = 2$ in $K_3 \times P_2$ and to $level = 3$ in $K_4 \times P_2$, we then see a slight rise as the overhead for placing SBDS constraints, while maintaining group information for SBDD begins to take effect. The time decreases again as SBDS is used across a bigger proportion of the search tree, until SBDD is never triggered. The symmetry breaking is performed purely by SBDS at $level = 10$ for $K_3 \times P_2$ and $level = 17$ for $K_4 \times P_2$. The total time taken by SBDS+D is comparative to that taken by SBDS from $level = 5$ for $K_3 \times P_2$ and $level = 11$ for $K_4 \times P_2$.

5.2 N-Queens

The N -queens problem is to place N queens on an $N \times N$ chessboard so that no queens can attack each other along a horizontal, vertical or diagonal line. The symmetry group of this problem comprises of the 7 board symmetries. It is an example of a problem with a small symmetry group which is the same order for any N . This class of problems is more efficiently solved by GAP-SBDS than GAP-SBDD as the communication overhead of GAP-SBDD is less than that of GAP-SBDS, but the group theory calculations are more complex. The results of applying SBDS+D to N-queens are in table 2.

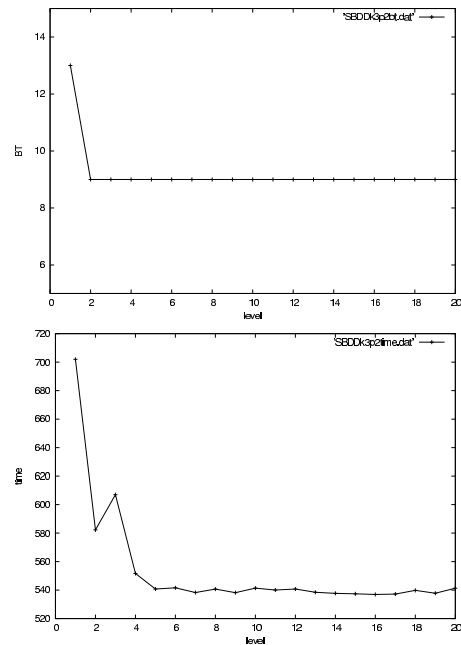


Figure 7: Plots of $K_3 \times P_2$ showing depth of change from SBDS to SBDD against number of backtracks and time (in Ms) respectively.

N	GAP-SBDS		SBDS+D level = 1		SBDS+D level = 2		SBDS+D level = 3		SBDS+D level = 4		SBDS+D level = 5		SBDS+D level = 6	
	BT	time	BT	time	BT	time	BT	time	BT	time	BT	time	BT	time
4	2	312	2	317	2	311	2	305	2	305	2	303	2	311
5	2	310	3	326	2	320	2	313	2	311	2	324	2	309
6	4	338	7	346	6	358	4	338	4	336	4	341	4	337
7	11	360	15	408	13	406	11	383	11	352	11	360	11	361
8	34	436	39	636	37	579	34	574	34	530	34	463	34	481
9	130	875	146	1379	137	1256	130	1130	130	1114	130	983	130	936
10	461	2496	505	4248	477	3674	462	3337	461	3387	461	3156	461	2873

Table 2: Comparison of GAP-SBDS and SBDS+D showing backtracks (bt) and the time (in Ms) for finding all N-Queens solutions.

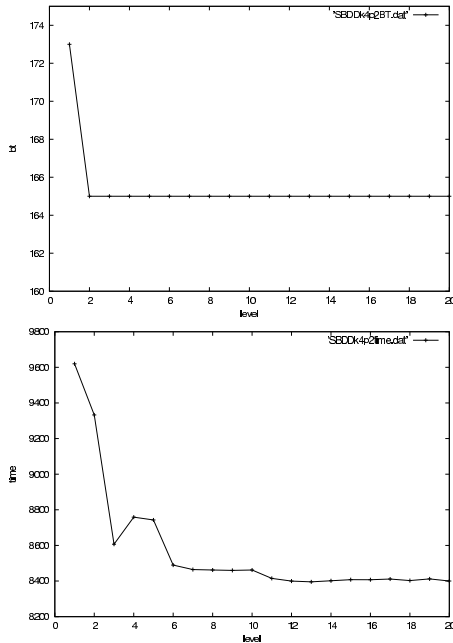


Figure 8: Plots of $K_4 \times P_2$ showing depth of change from SBDS to SBDD against number of backtracks and time (in Ms) respectively.

Analysing the results for N-Queens we see that SBDS is always faster and more efficient than GAP-SBDD alone (SBDS+D & level = 1). The number of backtracks using SBDS+D across all instances decreases to be the same as that of GAP-SBDS as the level of change increases. The total time for SBDS+D is always less than that of GAP-SBDD and in the smaller cases can also be less than that of GAP-SBDS. Continuing to increase the level further in SBDS+D for $N = 8, 9$ & 10 never decreases the time to bellow that of SBDS. Overall, using SBDS+D decreases the search effort and time in comparison to GAP-SBDD.

5.3 BIBDs

The computational version of the (v, b, r, k, λ) balanced incomplete block design (BIBD) problem is to find a $v \times b$ binary

matrix such that each row has exactly r ones, each column has exactly k ones and the scalar product of each pair of distinct rows is λ . The columns and rows of the matrix can all be permuted, giving the problem a symmetry group of size $v! \times b!$. Gent *et. al.* found GAP-SBDD capable of solving more instances than GAP-SBDS; for the problems both could solve GAP-SBDD was the more efficient method. Our experiments (in table 3) compare GAP-SBDD across the whole problem (SBDS+D with level = 1) against using SBDS just at the root node (SBDS+D with level = 2). SBDS+D with level > 2 does not show any improvement over SBDS+D with level 2 for all instances apart from (8,14,7,4,3), which has a larger search tree than other instances.

Studying table 3 we see that by using SBDS to place symmetry breaking constraints at the root node, we have reduced the number of backtracks in comparison to SBDD by 1, in every instance. Given the small number of backtracks needed to solve these instances, this is a reasonable reduction in search. The solving time is also slightly reduced in every instance apart from (8,14,7,4,3), this instance requires placing SBDS constraints to a deeper level to show an improvement, due to its larger search tree.

We can not solve some of the instances with SBDS+D that SBDD alone can solve, such as the (6,20,10,3,4) and the (7,21,6,2,1) instances, which have $O(10^{21})$ and $O(10^{23})$ problem symmetries respectively. With such large symmetry groups the communication overhead between GAP & ECL²PS^e required to place symmetries even at the root node is too high. In the future we hope to remedy this situation

Parameters					level = 1		level = 2	
v	b	r	k	λ	BT	time	BT	time
7	7	3	3	1	2	0.66	1	0.63
6	10	5	3	2	3	1.90	2	1.80
7	14	6	3	2	8	59.72	7	59.25
9	12	4	3	1	2	3.05	1	2.79
11	11	5	5	2	3	15.79	2	15.47
8	14	7	4	3	10	450.38	9	465.37
13	13	4	4	1	2	23.17	1	21.67

Table 3: Comparison of SBDS+D with level = 1 and level = 2, showing backtracks (bt) and the time (in seconds) for finding all BIBD solutions.

by allowing the SBDS+D user to fix a maximum number of constraints to be placed as well as a level in the search tree to change symmetry breaking method.

The timings shown seem comparable with that of the double-lex symmetry breaking method [4] where constraints are placed to break symmetry before search commences. Double-lex is an incomplete symmetry breaking method, whereas our SBDS+D method guarantees only to return non-isomorphic solutions.

6 Conclusion

We have compared the GAP-SBDS and GAP-SBDD symmetry breaking methods; to conclude that although GAP-SBDD has less of an ECLiPSe/GAP communication overhead than GAP-SBDS, the constraints GAP-SBDS places can cause it to be a more efficient symmetry breaking method. In order to combine the advantages of both methods we implemented hybrid methods SBDS+D and SBDD+S which allows a switch between SBDS and SBDD, or vice versa, at a given depth in the search tree.

Using SBDD+S is an incomplete symmetry breaking method; our experiments found this to be worse than GAP-SBDD or GAP-SBDS. In contrast, using SBDS+D is a complete symmetry breaking method. We performed experiments with SBDS+D on three problems, all with different symmetry 'types'. In the graceful graphs problem the symmetry is more efficiently broken by GAP-SBDS than by GAP-SBDD due to the extra propagation of constraints. In this problem SBDS+D breaks the symmetry comparably to GAP-SBDS and more efficiently than GAP-SBDD. The N-queens problem has a symmetry group of order 8 for all instances, a small symmetry group is more efficiently broken by GAP-SBDS than GAP-SBDD; SBDS+D outperforms GAP-SBDD in all instances and GAP-SBDS, for small values of N . The BIBD has a large symmetry group so the symmetry is more efficiently dealt with by GAP-SBDD. The instances we could solve with SBDS+D had smaller search trees than GAP-SBDD and for all but one instance the total time was reduced. Overall, our hybrid SBDS+D method compares favourably with the use of either GAP-SBDS or GAP-SBDD, in nearly all instances undertaken. This is a preliminary study into this new algorithm, more experiments have to be conducted before we can claim under which conditions SBDS+D will outperform GAP-SBDD or GAP-SBDS, but these initial results show that it is a welcome addition to the suite of CGT dynamic symmetry breaking methods.

Acknowledgements

The author is a member of the CP-Pod group and would like to thank the other members; especially Ian Gent, Tom Kelsey, Barbara Smith and Paula Sturdy. I am also very grateful to Warwick Harvey & Steve Linton for their technical assistance, Neil Yorke-Smith, Chris Beck, Meinolf Sellmann & Ian Miguel for their discussions. 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, *Proc. of CP'99*, LNCS 1713, pages 73–87. Springer, 1999.
- [2] A. M. Cheadle, W. Harvey, A. J. Sadler, J. Schimpf, K. Shen, and M. G. Wallace. ECLiPSe: An introduction. Technical Report IC-Parc-03-1, IC-Parc, 2003. www.icparc.ic.ac.uk/eclipse/.
- [3] T. Fahle, S. Schamberger, and M. Sellmann. Symmetry breaking. In T. Walsh, editor, *Proc. of CP'01*, LNCS 2239, pages 93–107. 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. V. Hentenryck, editor, *Proc. of CP'02*, LNCS 2470, pages 462–476. Springer, 2002.
- [5] F. Focacci and M. Milano. Global cut framework for removing symmetries. In T. Walsh, editor, *Proc. of CP'01*, LNCS 2239, pages 77–92. Springer, 2001.
- [6] J. Gallian. A Dynamic Survey of Graceful Labeling. In *The Electronic Journal of Combinatorics*, 2002. (<http://www.combinatorics.org/Surveys>).
- [7] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.2*, 2000. (<http://www.gap-system.org>).
- [8] I. P. Gent, W. Harvey, and T. Kelsey. Groups and constraints: Symmetry breaking during search. In P. V. Hentenryck, editor, *Proc. of CP'02*, LNCS 2470, pages 415–430. Springer, 2002.
- [9] I. P. Gent, W. Harvey, T. Kelsey, and S. Linton. Generic SBDD Using Computational Group Theory. In F. Rossi, editor, *Proc. of CP'03*, LNCS 2833, pages 333–347. Springer, 2003.
- [10] I. P. Gent and B. M. Smith. Symmetry breaking in constraint programming. In *Proc. of ECAI-2002*, pages 599–603. IOS Press, 2000.
- [11] W. Harvey. Symmetry Breaking and the Social Golfer Problem. In *Proc. SymCon-01: Symmetry in Constraints*, pages 9–16, 2001.
- [12] 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.
- [13] I. McDonald and B. M. Smith. Partial symmetry breaking. In *Proc. of CP'02*, LNCS 2470, pages 431–445. Springer, 2002.
- [14] K. Petrie. Why SBDD can be worse than SBDS. In *Proc. SymCon-03: Symmetry in Constraints*, pages 168–176, 2003.
- [15] K. E. Petrie and B. M. Smith. Symmetry breaking in graceful graphs. In *Proc. of CP'03*, LNCS 2833, pages 930–934. Springer, 2003.