Partial Symmetry Breaking

Iain McDonald[1] and Barbara Smith[2]

[1] University of St Andrews
Fife, Scotland
iain@dcs.st-and.ac.uk

[2] University of Huddersfield
West Yorkshire, England
b.m.smith@hud.ac.uk

Abstract. In this paper we define *partial symmetry breaking*, a concept that has been used in many previous papers without being the main topic of any research. This paper is the first systematic study of partial symmetry breaking in constraint programming. We show experimentally that performing symmetry breaking with only a subset of all symmetries can result in greatly reduced run-times. We also look at the consequences of using partial symmetry breaking in terms of variable and value ordering heuristics. Finally, different methods of selecting symmetries are considered before presenting a general algorithm for selecting subsets of symmetries.

1 Introduction and Motivation

We are now at a point in constraint programming research where there are many methods of both recognizing symmetries and breaking symmetries in CSPs. Backofen and Will [1] described a general method of applying symmetry breaking dynamically during search. Fahle, Schamberger and Sellman [2] recently presented a method of dynamic symmetry breaking which could deal with large groups of symmetries and guarantee unique solutions with large reductions in run-time.

A symmetry breaking **method** for CSPs, can be broken into two parts, the symmetry breaking **technique** and the symmetry **representation**. The technique is how we apply the symmetry breaking. Previous methods of breaking symmetry have involved introducing new constraints to the CSP that break symmetry [3], using heuristics that break symmetry by assigning the most symmetrical variables first [4], forbidding searching subtrees that are symmetrically equivalent to search already done [5, 1] or verifying at each node in search that it is not symmetrically equivalent to search already done [2]. The symmetry representation is concerned with how the descriptions of symmetries are implemented and how we use this implementation to apply the symmetry breaking technique. This was first shown in [6] where Focacci and Milano presented a method of better utilizing the symmetries of the problem independent of the symmetry breaking technique. If we are to come up with better symmetry breaking methods we must realize that the overhead for performing symmetry breaking exists in the representations of the symmetries.

Symmetry breaking can be improved by reducing the number of symmetries we need to consider. This affects the *representation* of the symmetries but not the *technique*.

In Section 2 we introduce the definitions and notation used throughout the paper. Section 3 looks more closely at the symmetry representation and some past improvements. This section also defines partial symmetry breaking and how to perform it with respect to the symmetry representation. We present empirical data of partial symmetry breaking experiments in Section 4 and discuss the implications of partial symmetry breaking and symmetry subset selection in Section 5. Finally we present our conclusions and discuss future work in Section 6.

2 Definitions and Notation

We define a CSP L, to be a finite set of variables X where each variable X_i has a finite domain $D(X_i)$. A solution to L is an assignment of all variables: $\{\forall i \exists j \mid X_i = D(X_i)_j\}$ such that a finite set of constraints C are satisfied. A CSP is a *symmetric CSP* if there are symmetries acting on it. A symmetry is defined as follows.

Definition 1. *Given a CSP L, with a set of constraints C, a symmetry of L is a bijective function $f : A \rightarrow A$ where A is some representation of a state in search*[1] *e.g. a list of assigned variables, a set of current domains etc., such that the following holds:*

1. *Given A, a partial or full assignment of L, if A satisfies the constraints C, then so does $f(A)$.*
2. *Similarly, if A is a nogood, then so too is $f(A)$.*

The set of all symmetries of a CSP form, or can be used to form, a **group**. The way in which partial symmetry breaking is performed depends on the symmetry representation. Previous encodings have used group theory techniques to represent a large number of symmetries by listing a small subset of all of them [7, 8, 9]. If it is possible to recreate the entire group of symmetries by reapplying the symmetries in this small subset, we call the subset a *generator set*. Many of the experiments and findings in this paper are based on representations that encode all the symmetries of a CSP and not just the generator set. The content of this paper is still relevant in the general case and modifications are suggested so that partial symmetry breaking can be used for all symmetry breaking techniques.

We now define two classes of symmetric CSPs.

[1] Symmetries can act on CSPs regardless of the notation used. Thus for generality, the method of representing an assignment and applying a symmetry is left to the reader.

Definition 2. *Given a CSP L where the number of symmetries of L increases polynomially with respect to the sizes of the variables X and their domains $D(X)$, L is said to be* polynomially symmetric*.*

Definition 3. *Given a CSP L where the number of symmetries of L increases exponentially with respect to the sizes of the variables X and their domains $D(X)$, L is said to be* exponentially symmetric*.*

In the polynomially symmetric most perfect squares problem[2] for example, the number of symmetries is $8n^2$ for an $n \times n$ board. Naïve encodings of the exponentially symmetric golfer's problem [10] have $p! \times g! \times w!$ symmetries for p players, g groups and w weeks. Clearly, increasing either the number of players, groups or weeks by even one will greatly increase the number of symmetries.

In group theory there are many groups with common structure and thus they are named e.g. given a set of n objects that can be permuted freely, the group acting on this set is called the symmetric group or S_n which has order $n!$. Most symmetric CSPs that are exponentially symmetric have S_n as part of their group.

3 Partial Symmetry Breaking and Symmetry Representation

There have already been two improvements reported on the representation of symmetries i.e. methods for removing symmetries from consideration from the symmetry representation. The first is found in the symmetry breaking method SET (symmetry excluding trees) developed by Backofen and Will [1] and this removes broken symmetries. Removing broken symmetries from consideration is also in the symmetry breaking method that will be used in the experiments in this paper: SBDS. Therefore, the concept will be explained in terms of the SBDS notation. Symmetry Breaking During Search (SBDS), developed by Gent and Smith [5], works by adding constraints to the current search subtree. After backtracking from a failed assignment $var_i = val_j$, to a point in search with a partial assignment A, we post the constraint:

$$g(A) \ \& \ (var_i \neq val_j) \Rightarrow g(var_i \neq val_j)$$

for every g in the symmetry representation. Symmetries are represented by functions and SBDS removes a function from consideration when it discovers that a pre-condition (i.e. $g(A)$) of the constraint it creates is guaranteed false from the current subtree. For example consider a node k in search, a symmetry function may produce a pre-condition $var_i = val_j$ but if at point k, $var_i \neq val_j$ we can ignore that symmetry function at all children nodes of k.

[2] A variation on the magic squares problem. See http://www.geocities.com/~harveyh/most-perfect.htm for more details

The second improvement is found in [8] where only unique symmetries are considered. McDonald showed how at certain points in search, some sets of symmetries all have the same effect on a partial assignment and so we can discard all but one symmetry from this set e.g. there may be a set of symmetries g and h such that given a partial assignment A, $g(A) = h(A)$. If this is the case we only break symmetry on g or h but not both. These two improvements reduce the number of symmetries to consider without reducing the amount of symmetry breaking possible i.e. they do not introduce non-unique solutions.

We now consider a third optimization. In this paper we show that where there is a large number of symmetries, we can discard some of them and by doing so reduce run-time greatly. If we are trying to solve a problem that is exponentially symmetric we may not be able to fully utilize a given symmetry breaking technique. We cannot apply the dominance check used in SBDD[3] for all symmetries at every node in search for the golfer's problem as it is too expensive. It is still possible to use SBDS if we limit the number of symmetry breaking functions and we can still use SBDD by applying a dominance check under a subgroup of all the symmetries. Describing only a subset of the symmetries does not lose any solutions and may result in redundant search but the overhead of performing symmetry breaking will not be as great. By only describing a subset of symmetries we are performing *partial symmetry breaking* (PSB) i.e. performing some redundant search because the symmetry breaking technique is too costly or even impossible to perform.

An example of PSB can be found in [5] where a restricted method of SBDS is described. It was proved in [1] that if S_n acts on the values of a variable, we can break all symmetry in the values by breaking the symmetry of just the $O(n^2)$ transpositions. In [10] the entire paper used PSB. Finally, in [2] the golfer's problems is solved using SBDD for the most part with only a subgroup of all the possible symmetries[4] since the dominance check on the whole group is too expensive.

As can be seen, previous work has looked at PSB and some small experiments have been completed [2], but this paper is the first to consider systematically what the benefit of PSB is and how its benefit can be maximized.

3.1 Explicit Symmetries and Group Theory

Given a symmetry representation, we perform PSB by only applying the symmetry breaking technique to a subset of the symmetries in the symmetry representation. How the subset of symmetries is generated depends on the symmetry representation. There are two types of symmetry representation:

1. A list of explicit symmetries
2. A generator set of a group

[3] Symmetry breaking via dominance detection [2].

[4] Apart from at the leaf nodes where the entire group is used to verify unique solutions.

Generating a subset of symmetries from a list of explicit symmetries is trivial, however, the implicit nature of using generators of groups makes it difficult (but still possible) to select a subset of symmetries. Symmetry breaking methods that use generators of groups must still generate elements of the group in order to perform the symmetry breaking technique. When this process of generating group elements takes place, the number of the elements generated should be limited by some global number. By doing so we are performing PSB.

This paper uses the SBDS symmetry breaking method whose symmetry representation uses a list of explicit symmetries. The current thinking in symmetry breaking is that group theory should be more widely used and we will show later that PSB will be easier to extend to symmetry breaking methods that use group theory.

4 Partial Symmetry Breaking Experiments

It is a straightforward assumption that by breaking more symmetries i.e. by increasing the number of symmetries in the representation, we can reduce the search space further up to a certain point.

However, as the number of symmetries represented increases, so does the overhead. Figure 1 illustrates this point by suggesting that there may be a point where the benefit in reducing search from adding more symmetries is out-weighed by the extra overhead

In order to discover how the cpu-time of solving a symmetric CSP varies with the number of symmetries used in the symmetry representation we have constructed the following experiment. Given a symmetric CSP L with n symmetries acting on it, L is solved k times using no symmetry breaking and the cpu-time is recorded. We then solve L another k times - again recording the cpu-time - using some symmetry breaking technique and k different pseudo-random subsets of symmetries of size 1 as the symmetry representation. This pattern is repeated for k pseudo-random subsets of size 2 up to n.

Given the data from the above experiment we can plot graphs of cpu-time *vs* number of symmetries used. We can then use this graph to estimate how

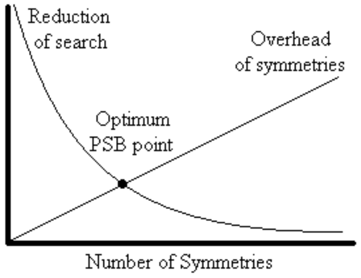


Fig. 1. Finding the optimum point

many symmetries we need to break to minimize cpu-time for SBDS. It should be highlighted though that by doing this we allow duplicate solutions. As was shown in [2] though, using SBDD to break all symmetry on leaf nodes of search is inexpensive and guarantees unique solutions.

4.1 Fractions Puzzle

We consider a very simple problem as an example experiment. Given the following problem:

$$\frac{A}{BC} + \frac{D}{EF} + \frac{G}{HI} = 1$$

Can we find values for each variable such that the equation[5] is satisfied? We can permute the fractions freely, yielding 5 symmetries and the identity e.g. one symmetry is $A \leftrightarrow D$, $B \leftrightarrow E$ and $C \leftrightarrow F$. Since the number of symmetries is so small it is possible to run the experiment with all possible subsets of symmetries. The cpu-times were then averaged for each subset size. Figure 2 contains the graph of the averaged cpu-time with respect to the number of symmetries. As you can see, by adding more symmetries the cpu-time decreases.

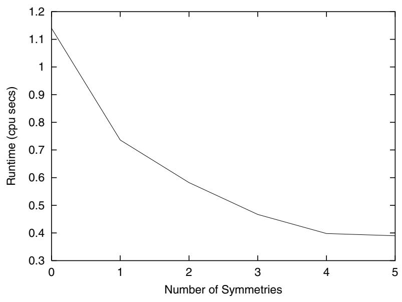


Fig. 2. Fractions Puzzle PSB

4.2 Alien Tiles

SBDS has already been used to solve alien tiles problems with good results [11]. The alien tiles problem can be described with two parameters n and c, the size of the board and the number of colours respectively. An alien tiles board is an $n \times n$ grid of n^2 coloured squares[6]. By clicking on any square on the board, the colour

[5] BC does not mean $B \times C$ but rather $(10 \times B) + C$.

[6] Alien tiles puzzles can be found online at http://www.alientiles.com/

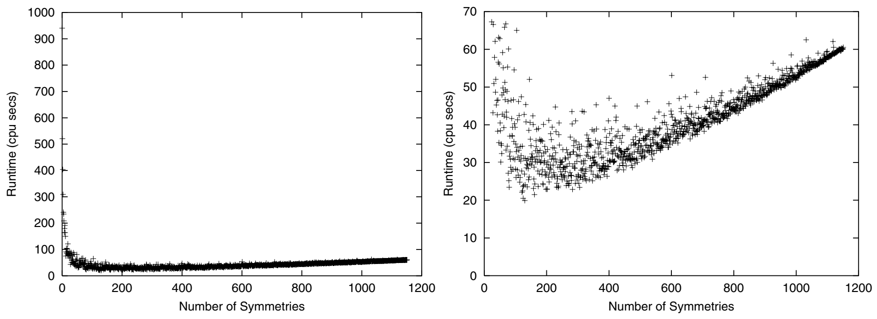


Fig. 3. Random PSB Subsets - Alien Tiles

of the square is changed +1 *mod c*. As well as this, the colour of every square in the same row and column is also altered +1 *mod c*. Given an initial state and a goal state, the problem is to find the required number of clicks on each square which can be anything between 0 and $c - 1$ (since $0 \equiv c$, $1 \equiv c + 1$ etc). A more challenging problem for constraint programming is finding the most complicated goal state (in terms of the number of clicks needed) and then reaching that goal state in as few clicks as possible and verifying optimality.

The problem we consider is a 4×4 board with 3 colours. The smallest number of clicks that can take us to the most complicated goal state is 10. Proving that 10 clicks is optimal needs a complete traversal of the entire search tree. An instance of the alien tiles problem is exponentially symmetric. Given a solution we can freely permute the rows and columns and flip the board around a diagonal. For a board with n^2 variables, the group acting on the board is $S_n \times S_n \times 2$ which for a 4×4 board is a group of size 1152, or 1151 symmetries and the identity. We derive this number by noting that we have 24 (or 4!) row permutations, which can be used in conjunction with the 24 column permutations, which can be used with the diagonal flip ($2n!^2$). The reason we are using this symmetric CSP as the main example of PSB is that it is not a trivially easy problem to solve, but with $n = 4$ we can cope with all 1151 symmetry functions so we can compare PSB against breaking all symmetry.

Figure 3 shows the cpu-time to solve the alien tiles problem described above with different sized random[7] subsets of the 1151 symmetries. Figure 3 also shows a magnified version of the same graph so that we can see the results more clearly. By looking at the graphs we can deduce three things. Firstly, most of the run-time improvement from 940.4 seconds and 18751 backtracks with no symmetry breaking to 60.5 seconds and 135 backtracks with all 1151 symmetry functions comes from adding the first 20 or so symmetries. Secondly and perhaps most importantly, we can see that shortest cpu-time comes from using a random subset of size 130. With this subset the problem was solved in 19.9 seconds and with 216 backtracks. The size of this subset is much smaller than the size of the group

[7] In this experiment the ECL^{i}PS^e random function was used.

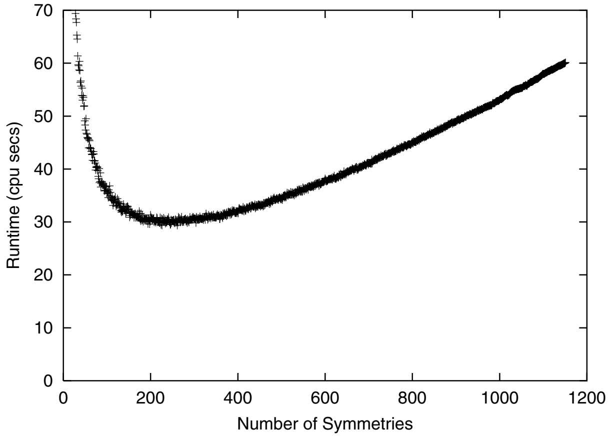


Fig. 4. Average cpu-times - Alien Tiles

acting on the alien tiles CSP. Thirdly, different subsets of a similar size have large differences in cpu-time. This implies that which symmetries we include in our subset is just as important as the size of the subset e.g. another random subset of size 130 yielded a cpu-time of 54.9 seconds (almost as much as breaking all symmetry).

The above experiment was run with 218 different random subsets[8] for each subset size to produce the less scattered curve in Figure 4. It is possible to gain an average factor of 2 improvement over breaking all symmetry and a factor of 32 improvement over no symmetry breaking. In the case of the subset of size 130 mentioned above, we gain a factor of 3 improvement over breaking all symmetry and a factor of 47 improvement over no symmetry breaking. The shape of the curve in Figure 4 is consistent with Figure 1 i.e. the overhead increasing approximately linearly with the number of symmetries, combined with a steep reduction in search as the first few symmetries are added. This reduction tails off as most of the redundant search is pruned, making further symmetries less effective.

4.3 Golfer's Problem

Here we show the existence of similar behaviour for a different problem. This uses Smith's encoding of the golfer's problem taken from [10] with $p!$ symmetries for p players. This well known problem takes three parameters - $golf(p, g, w)$ - the number of players p, the number of groups in each week g where $p \ mod \ g = 0$ and the number of weeks w. The constraints on this problem are that for each week each player plays golf with the other players in their respective group. Once two players have played golf with one another in one group in one week, they cannot play each other again in any other week. The graph shown in

[8] Using ECL^{i}PS^e version 5.3 on a Pentium III 1GHz processor with 512Mb of RAM

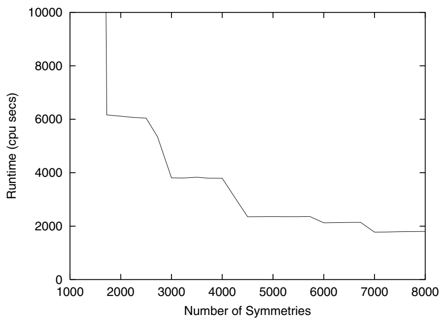


Fig. 5. PSB - Golfer's Problem

Figure 5 shows the results of finding all solutions to $golf(12, 4, 2)$[9]. Using PSB while finding all solutions will introduce duplicate solutions. Smith's model has $12! - 1$ or 479,001,599 symmetries not including the identity. Using GAP [12] it is possible to produce random elements of the group acting on this problem: S_{12}. GAP was then used to output a random subset of 8000 functions representing 8000 random elements of the group. The same experiment described at the start of this section was run with just one random subset, but due to the complexity of this problem the subsets of symmetries incremented in size in steps of 250. It was not possible to solve the problem with 1500 symmetry breaking functions within 1000 minutes of cpu-time.

The graph in Figure 5 is not as clear as that seen in Figure 3. However, whereas the alien tiles problem needed roughly 20 symmetries to do most of the symmetry breaking, the golfer's problem needs roughly 4500. We need to consider at least 1775 symmetries to be able to solve this problem in reasonable time and the more symmetries we add the smaller the improvement in cpu-time. Using SBDS we are limited by the number of functions we can compile. In this respect it is more advantageous to represent symmetries using groups so that larger subsets of symmetries can be used as was discussed in Section 3.1.

5 Symmetry Subset Selection

In the previous section we saw empirical evidence that using PSB can produce significant improvements. This also highlighted the importance of symmetry subset selection i.e. how we choose the subset of symmetries to break. Figure 6 shows the best and worst cpu-time for different sized subsets of symmetries as well as the absolute difference between them based on the 15 random subsets used in the experiment (described in Section 4.2[10]). The minimum cpu-time we can achieve is 12.61 seconds with a subset of 164 symmetries. However choosing a subset of this size can result in a cpu-time as large as 35.27 seconds. We now look at how

[9] Using Ilog Solver version 4.4 on a Pentium II 300MHz processor with 512Mb RAM
[10] Using ECL^{i}PS^e version 5.3 on a dual Pentium III 1GHz processor with 4Gb RAM

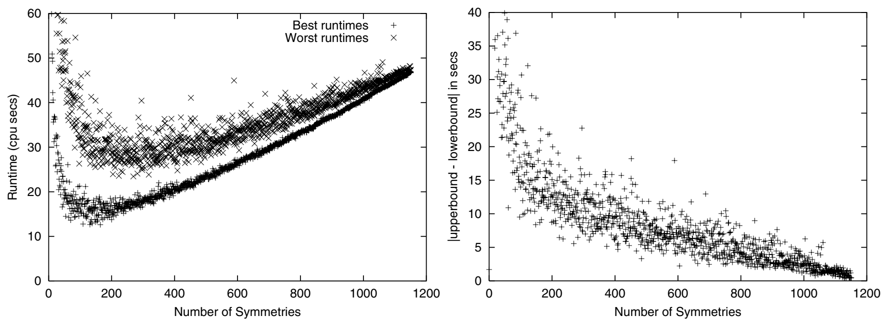


Fig. 6. Best & worst times (left, cut-off at 60secs) and the difference between best & worst times (right, cut-off at 40secs)

the symmetry subset selection effects search and in doing so, hope to find an algorithm to select efficient symmetry subsets.

When solving a symmetric CSP using symmetry breaking techniques there are two types of failure resulting in backtracking. We either fail where we discover a new unique nogood or we fail where we find a nogood symmetrically equivalent to a previously unique nogood.

Definition 4. *Given a complete traversal of the search tree of a CSP L, a list of nogoods found K and a group of symmetries G, acting on L, consider a node in search k which is a nogood. If while traversing the search tree we reach node k and $\nexists g \in G$ s.t. $g(k) \in K$ then we call k a* unique nogood *else if $\exists g \in G$ s.t. $g(k) \in K$ where $g \neq e$ (the identity element) then we call k a* symmetric nogood. *Unique nogoods result in unique fails and symmetric nogoods result in symmetric fails.*

It is straightforward to see that exponentially symmetric problems can have significantly more symmetric fails than unique fails. By performing symmetry breaking we can eliminate symmetric fails, however if we use PSB some symmetric nogoods persist. Different symmetries can be used to prune different parts of the search tree. The variable and value ordering heuristics and the propagation level dictate how the search space is traversed, therefore the symmetric nogoods pruned are dependent not only on how many symmetries we break but also on the heuristics we use.

In Figure 7 we present experimental evidence of this by performing the same experiment as in Section 4.2 with the same subsets of symmetries but with different variable ordering heuristics[11]. The subsets used in Section 4.2 were randomly chosen for each different size but in this section we have a standard subset that has a random symmetry added to it at the start of each run. The first heuristic (on the left in Figure 7) instantiates the alien tiles squares along the rows from top left to bottom right. The second (on the right) instantiates the

[11] Using ECL^{i}PS^e version 5.3 on a dual Pentium III processor 1GHz with 4Gb RAM

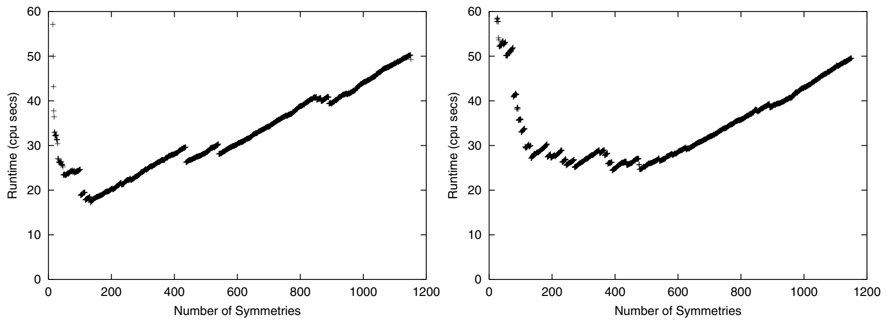


Fig. 7. Identical subsets of symmetries with different variable ordering heuristics (cut-off 60secs). The heuristic used on the left is better up to 376 symmetries after which the heuristic used on the right takes less time

squares along the rows from bottom right to top left. The resulting cpu-times are generally significantly different. On the other hand, changing the value ordering heuristic in solving alien tiles problems makes no difference to the number of symmetric fails we find, since the symmetries in this problem act on just the variables and not the values.

If we want to use PSB with a given symmetry breaking method we need to be aware of the variable and value ordering heuristics when we select a subset of symmetries. We can exploit this fact by choosing heuristics that work well with respect to a subset of symmetries.

5.1 Cats and Dogs

We now look at a problem where it is possible to select a good subset of symmetries with respect to the variable ordering heuristic. The Cats and Dogs problem requires 3 cats and 5 dogs to be placed on a 5×5 chessboard in such a way that no dog can attack any of the cats in a queen's move.

A possible model has a variable for each cat (c_1, c_2, c_3) and a variable for each dog $(d_1, d_2, d_3, d_4, d_5)$. The domain of each variable is the set of squares on the board. There are 15 binary constraints, each between a cat variable and a dog variable, to ensure that no dog and cat are on the same row, column or diagonal. We also have an *all-different* constraint (or one for the cats and one for the dogs, since the constraints already prevent placing a dog and a cat on the same square). The symmetries of the problem are composed of the symmetries of the chessboard, together with the fact that the cat variables are indistinguishable and so are the dog variables. Using this simple model we have $8 \times 3! \times 5!$ or 5760 symmetries. There is only one solution to the problem, ignoring symmetric equivalents. With no symmetry-breaking, there are in fact 5760 solutions: it takes 7046 fails and 4 seconds to find these[12].

[12] Using Ilog Solver version 4.4 on a Celeron 600MHz laptop

The model is solved by assigning the variables in the fixed order $c_1, c_2, c_3, d_1, ..$ $, d_5$. In this case, the variable ordering means that the cat symmetries and the dog symmetries have a different effect on search: the cat symmetries are more significant, because the cat variables are assigned first. Ignoring the symmetries of the chessboard, we could clearly eliminate the symmetry due to indistinguishable variables by adding ordering constraints $c_1 < c_2 < c_3$ and $d_1 < ... < d_5$. This suggests that in SBDS we could eliminate these symmetries by describing all transpositions of the c_is and all transpositions of the d_js *independently.* This can be done with 13 SBDS functions (3 for the cat transpositions and 10 for the dog transpositions), and then 8 solutions are found in 1,074 fails with 0.9 seconds. Adding functions for the 7 board symmetries, other than the identity, (20 SBDS functions) reduces the number of solutions to 3 with 513 fails.

Since there is a unique solution, we have not yet eliminated all the symmetry. We continue by combining a board symmetry with a transposition of the cat variables: this gives an additional 21 SBDS functions, 41 in total. Now only one solution is found in 0.3 seconds, and the number of fails is 210. Hence it has been possible in this case to find a small subset of the symmetries (less than 1% of the total) which eliminate all equivalent solutions to the problem.

5.2 Algorithm for Symmetry Selection

The structure in the Cats and Dogs problem means it is possible to construct a variable ordering heuristic that breaks all symmetry with respect to a few hand picked symmetries. For other problems that do not have as much structure as the Cats and Dogs problem, a general purpose algorithm is needed.

If a subset of symmetries is to be used, the symmetries that prune most nogoods should be included in this subset. The symmetries that rule out nodes near the root should prune the most search. Based on this, the following algorithm (Algorithm 5.1) was implemented using GAP's group theoretic capabilities. A symmetry function can be good with respect to variable and value ordering heuristics **and** a failed partial assignment. Therefore, we look at how near to the root the prune is made with respect to every symmetry and every partial assignment. The algorithm orders the symmetries so that those that prune symmetric nogoods near the root are before those that only prune nogoods near the bottom of the tree. To perform PSB with n symmetries, we should use the first n symmetries in the sorted list returned by this algorithm.

This complete algorithm needs to know ahead of time every partial assignment that will be considered e.g. from the variable and value ordering heuristics to be used, and every symmetry of the group. Every symmetry acting on the problem is then applied to these partial assignments. The computation involved in using this algorithm makes it almost unusable for all but the smallest groups and smallest CSPs. Thankfully we can take some random subset of symmetries of size k and find the best n symmetries of this subset (where $k >> n$). We can also use just a subset of all partial assignments (smaller partial assignments are more preferable).

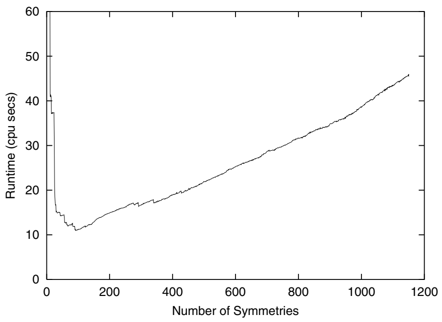
Algorithm 5.1: SymmetrySubsetSelection(*Group*, *PartialAssignments*)

```
for each $g \in Group$
do
    for each $pa \in PartialAssignments$
    do
        $element.partial\_assignment \leftarrow g(pa)$
        $element.symmetry\_used \leftarrow g$
        $element.latest \leftarrow$ latest point in search in $g(pa)$[a]
        $SymmetricPartialAssignments.add(element)$
for each $i \in SymmetricPartialAssignments$
do
    $best \leftarrow i$
    for each $j \in SymmetricPartialAssignments$
    do
        if $j.latest = best.latest$
         and $|j.partial\_assignment| < |best.partial\_assignment|$
          then $best \leftarrow j$
        if $j.latest < best.latest$
          then $best \leftarrow j$
    $Symmetries.add(best.symmetry\_used)$
    $SymmetricPartialAssignments.remove(best)$
$Symmetries.remove\_duplicate\_symmetries()$
return ($Symmetries$)
```

[a] Latest $var = val$ assignment to be considered by the heuristics in the partial assignment $g(pa)$.

By using the complete version of this algorithm, it is possible to obtain an ordering of all 1151 symmetries of the alien tiles problem. Also, using a limited version of this algorithm for just partial assignments with 3 or less variables instantiated i.e. we consider just the partial assignments made at depth 3 or less in the search tree, it is possible to obtain a sorted list of 8000 symmetries for the golfer's problem. The experiments in Section 4 were run again with these symmetries. In Figure 8 we compare the results of using this algorithm with respect to previous experiments. The alien tiles problem takes about the same time to solve using the best random subsets in Figure 6 as the symmetries from the complete algorithm.

Even though we can now solve the alien tiles problem in 10.95 seconds with a subset of 92 symmetries, Algorithm 5.1 took 1232 seconds to order the symmetries. However, the extraordinary performance improvement from 1802.53 seconds to 128.1 seconds in solving the golfer's problem warrants the 364 seconds it took to sort the symmetries. This shows that while the complete algorithm is very time consuming for a CSP like the alien tiles problem, using a limited version produces very good results in a reasonable time. This suggests that this algorithm can be used to solve CSPs that cannot be solved with symmetry breaking and random PSB. Future work should reveal a better method of sort-



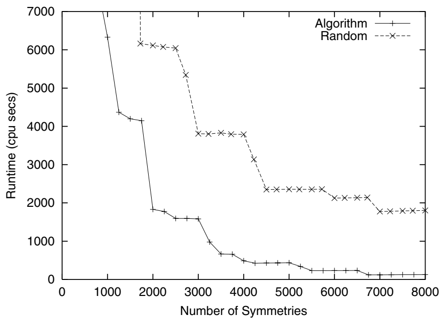


Fig. 8. The alien tiles experiment (left). The golfer's problem (right) is significantly improved over using random symmetries

ing symmetries but this algorithm shows that a generic method of choosing the symmetries that will prune most search is possible.

6 Conclusions and Future Work

This paper is the first systematic study of partial symmetry breaking (PSB) in constraint programming. We have shown that by performing PSB it is possible to significantly decrease run-time in solving symmetric CSPs. Furthermore we have shown that consideration needs to be given to the variable and value ordering heuristics when solving symmetric CSPs suggesting that hand picked symmetries perform better than random symmetries with PSB. Finally we presented a general algorithm for selecting subsets of symmetries. This paper also shows the need for more research into finding general methods for breaking all symmetry with a subset of symmetries.

Future work will look at constructing an efficient symmetry breaking system that implements all three improvements mentioned in Section 3.1. This symmetry breaking system will use group theory to express huge groups, consider non-broken and unique symmetries only and use PSB to minimize run-time.

Acknowledgments

This research is supported by EPSRC grant GR/R29673. The first author is funded by an EPSRC studentship. He would also like to thank his supervisors Ian Gent and Steve Linton as well as Tom Kelsey. Both authors would like to thank all the members of the APES research group, especially Toby Walsh, for their support and helpful comments.

References

[1] Rolf Backofen and Sebastian Will. Excluding symmetries in constraint-based search. In Alex Brodsky, editor, *Principles and Practice of Constraint Programming*, pages 73–87. Springer-Verlag, 1999. 431, 433, 434

[2] Torsten Fahle, Stefan Schamberger, and Meinolf Sellman. Symmetry breaking. In Toby Walsh, editor, *Principles and Practice of Constraint Programming – CP2001*, pages 93–107. Springer-Verlag, 2001. 431, 434, 436

[3] James Crawford, Matthew Ginsberg, Eugene Luks, and Amitabha Roy. Symmetry-breaking predicates for search problems. In *Knowledge Representation'96: Principles of Knowledge Representation and Reasoning*, pages 148–159. Morgan Kaufmann, San Francisco, California, 1996. 431

[4] Pedro Meseguer and Carme Torras. Exploiting symmetries within constraint satisfaction search. In *Artificial Intelligence, Vol 129, No. 1-2*, pages 133–163. 2001. 431

[5] Ian Gent and Barbara Smith. Symmetry breaking in constraint programming. In W. Horn, editor, *Proceedings of ECAI-2000*, pages 599–603. IOS Press, 2000. 431, 433, 434

[6] Filippo Focacci and Michaela Milano. Global cut framework for removing symmetries. In Toby Walsh, editor, *Principles and Practice of Constraint Programming – CP2001*, pages 77–92. Springer-Verlag, 2001. 431

[7] Warwick Harvey. Symmetry breaking and the social golfer problem. In Piere Flener and Justin Pearson, editors, *SymCon'01: Symmetry in Constraints*, pages 9–16, 2001. Available from http://www.csd.uu.se/˜pierref/astra/symmetry/index.html. 432

[8] Iain McDonald. Unique symmetry breaking in CSPs using group theory. In Piere Flener and Justin Pearson, editors, *SymCon'01: Symmetry in Constraints*, pages 75–78, 2001. Available from http://www.csd.uu.se/˜pierref/astra/symmetry/index.html. 432, 434

[9] Cynthia Brown, Larry Finkelstein, and Paul Purdom Jr. Backtrack searching in the presence of symmetry. In *Nordic Journal of Computing*, pages 203–219. Publishing Association Nordic Journal of Computing, 1996. 432

[10] Barbara Smith. Reducing symmetry in a combinatorial design problem. Technical Report Research Report 2001.01, University of Leeds, January 2001. 433, 434, 438

[11] Ian Gent, Steve Linton, and Barbara 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. 436

[12] The GAP Group, Aachen, St Andrews. *GAP – Groups, Algorithms, and Programming, Version 4.2*, 2000. 439