Symmetry Breaking in Constraint Programming

Ian P. Gent[1] and **Barbara M. Smith**[2]

Abstract. We describe a method for symmetry breaking during search (SBDS) in constraint programming. It has the great advantage of not interfering with heuristic choices. It guarantees to return a unique solution from each set of symmetrically equivalent ones, which is the one found first by the variable and value ordering heuristics. We describe an implementation of SBDS in ILOG Solver, and applications to low autocorrelation binary sequences and the n-queens problem. We discuss how SBDS can be applied when there are too many symmetries to reason with individually, and give applications in graph colouring and Ramsey theory.

1 INTRODUCTION

Many constraint satisfaction problems (CSPs) contain symmetries, so that for every solution, there are equivalent solutions. Symmetries divide the set of possible assignments into equivalence classes. Each class contains either only solutions or no solutions. When finding all solutions to a CSP, it is only necessary to find one solution from each class: the others can be retrieved using the symmetries, if required. When we only want one solution, by restricting the search to one member of each class, we hope to avoid redundant search effort. We will assume that the symmetries in the problem have already been recognised, so that the issue is how to deal with them. We want to ensure that whenever a partial assignment is shown to be inconsistent, no symmetrical assignment is ever tried, and that if we are finding all solutions we only find distinct solutions, i.e. never two from the same equivalence class.

Several ways of dealing with symmetries have been proposed. Brown, Finkelstein and Purdom [3] define an ordering on the set of assignments and find only the smallest solution in this ordering in each equivalence class; this is done by. pruning branches of the search tree during search, if it can be shown that a symmetrical branch has been or will be explored. Roy & Pachet [9] discuss a particular form of symmetry (*intensionally permutable* variables); when retracting the assignment of a value to a variable, the value can be removed from the domain of any intensionally permutable variable.

Another approach is to add constraints to the CSP to convert it into an asymmetrical, or less symmetrical, one. This is the method most commonly used in practice. Puget [8] gives a formal framework for the introduction of ordering constraints to break symmetries. Crawford, Ginsberg, Luks & Roy [4] adopt a similar approach in satisfiability problems and show how to construct a symmetry-breaking predicate automatically. Although adding constraints to the problem is often successful, in general it has some serious drawbacks. First, it appears to require the instantiation order to be at least partly fixed in advance. This restricts the freedom of the search algorithm, so that although in theory this approach makes symmetry-breaking independent of the search for a solution, in practice it is not. Second, as we discuss in section 3.1, it is hard to add constraints at the start to deal with symmetries which may persist even after several assignments have been made.

We suggest three aims for a symmetry-exclusion method. The first is to guarantee that we never allow search to find two symmetrically equivalent solutions. The second is to respect heuristic choice as much as possible. This would in general exclude methods which post constraints at the root of the search tree, as they may exclude solutions which the heuristic search strategy will find early in favour of solutions the heuristic finds only after much search. Finally we should allow arbitrary forms of symmetry. The approaches discussed above do not meet these aims.

A third approach is to break symmetries during the search, by detecting which symmetries remain unbroken when the search tries a new branch after backtracking. We add constraints at that point which will prune parts of the search tree symmetrical to those already explored. Symmetry breaking therefore acts alongside the search strategy, co-operating with it.

We present a general method for symmetry breaking during search (SBDS). The approach was introduced by Backofen and Will [2], with an application to geometric constraints in protein structure prediction in an earlier paper [1]. A less general method which also removes symmetries during search is described by Meseguer and Torras [7].

We show in detail how SBDS can be implemented with minimal overhead in ILOG Solver, a constraint programming library, provided that the symmetries can be explicitly stated. This is successful in finding all solutions to n-queens, and in the low autocorrelation binary sequences (LABS) problem. We also discuss the current limitations of this approach. Where there are many symmetries, full SBDS is impractical. We describe a new shortcut method which again works with the search strategy, but may not eliminate all symmetries, and describe successful implementations for graph colouring and Ramsey theory.

2 SYMMETRY-BREAKING DURING SEARCH

As an example, consider the 8-queens problem, where the variable Q_i represents the queen on the ith row, and the value assigned represents the column where it is placed. One of the symmetries of this problem is rotational symmetry through $180°$. Suppose that the first two assignments are $Q_1 = 2$ and $Q_2 = 4$. On backtracking from the second assignment, and making the alternative choice $Q_2 \neq 4$, should we allow its symmetric equivalent, i.e. $Q_7 = 5$? It depends whether rotational symmetry still exists in the problem or not. If we eventually place $Q_8 = 7$, the symmetrical version of $Q_1 = 2$, then the decisions $Q_2 = 4$ and $Q_7 = 5$ are symmetri-

[1] School of Computer Science, University of St Andrews, North Haugh, St Andrews, Fife KY16 9SS, U.K.

[2] School of Computer Studies, University of Leeds, Leeds LS2 9JT, U.K.

cal and we should not try the second if the first fails. On the other hand, if $Q_8 \neq 7$ then the decisions $Q_2 = 4$ and $Q_7 = 5$ are not equivalent and we must consider $Q_7 = 5$ if $Q_2 \neq 4$. The complication is that when the node of the search tree is created with the choice between $Q_2 = 4$ and $Q_2 \neq 4$, we *don't know* yet whether $Q_8 = 7$ or not. The solution is to post a *conditional constraint*, that $Q_1 = 2$ & $Q_2 \neq 4$ & $Q_8 = 7 \Rightarrow Q_7 \neq 5$.

The antecedents of this constraint play three rôles. The first, $Q_1 = 2$, limits the constraint to descendants of the node where the branching choice $Q_2 = 4$ was made. The second, $Q_2 \neq 4$, restricts to nodes which backtrack from that decision. While these are unnecessary if constraints are only posted locally, we include them to make our symmetry constraint valid globally. The third antecedent, $Q_8 = 7$, ensures that the constraint will only take effect where the rotational symmetry still exists.

To generalise this example, assume that some branching decision $var = val$, as opposed to $var \neq val$, is being made in the context of some partial assignment A, and that we are considering some particular symmetry g. We take a symmetry g to be a function which maps full or partial assignments to other assignments, and which is a one-to-one function on full assignments. We also assume that a symmetry function g can be applied to constraints. A symmetry is solution-preserving: that is, for any full assignment A and any symmetry g of the problem, $g(A)$ is a solution iff A is. We also assume that a symmetry function g acts piecewise. Formally, if we extend an assignment A to $A + (var = val)$, then $g(A + (var = val)) = g(A) + g(var = val)$. This assumption is necessary if we are to reason about symmetric versions of assignments in the middle of the search tree when assignments are only partial. Writing A for the constraint that all assignments in A hold, and $g(A)$ for the constraint that the symmetric versions of these assignments hold, our general constraint is:

$$A \ \& \ g(A) \ \& \ var \neq val \Rightarrow g(var \neq val) \tag{1}$$

The SBDS method is to post (1) at each branching point and for each symmetry g identified by the programmer. The method could be extended to other forms of branching decision, by expressing the effect of each symmetry on the branching constraint. For instance, if the branching decision were between $var \leq val$ and $var > val$, in (1) $\neq$ is replaced by $>$. We describe how SBDS can be implemented with minimal overhead in cpu-time and minimal effort from the programmer. We rely on [2] for proofs of correctness of the method.

3 IMPLEMENTATION OF SBDS

To implement SBDS in ILOG Solver, we provide a new search function IlcGenerateSym to replace Solver's IlcGenerate. IlcGenerate searches by first choosing a variable *var* according to some user-specified strategy. Then it chooses a value *val* similarly and forms a choice point giving two branches. Along the left branch, *var* = *val*; if this leads to failure, the right branch, *var* $\neq$ *val*, is explored instead.

Our new function eliminates symmetries, yet can be used with any variable and value ordering strategy compatible with IlcGenerate. The main argument is an array of search variables `vars`. We assume that all branching points are of the form `vars[i] = j` for some array index `i` and value `j`. The programmer using IlcGenerateSym implements one function for each symmetry in the problem. The function for the symmetry g takes three arguments, representing `vars`, `i` and `j` and returns a constraint representing $g(\texttt{vars}[\texttt{i}] = \texttt{j})$. An array of these functions is passed to IlcGenerateSym.

To implement IlcGenerateSym, the constraint given by (1) is added to the $\texttt{vars}[\texttt{i}] \neq \texttt{j}$ branch for each symmetry g. We omit the condition that A holds, because the constraint will be local to this node. The main subtlety is in the constraint $g(A)$. Since this involves all search variables set so far, it is potentially large. However, we note that if we extend A to the assignment $A^+ = A + \texttt{vars}[\texttt{i}] = \texttt{j}$, then $g(A^+) = g(A) + g(\texttt{vars}[\texttt{i}] = \texttt{j})$. We construct a new boolean variable for each symmetry g representing whether $g(A)$ is satisfied or not. The value of this variable for $g(A^+)$ is the conjunction of its value for $g(A)$ and $g(\texttt{vars}[\texttt{i}] = \texttt{j})$. Hence, we can compute $g(A)$ incrementally. These boolean variables have a further advantage. When one is proven to be false, we know that g, the corresponding symmetry, is permanently broken on this branch. This makes it safe to discard g from further consideration on the branch. Hence, we only incur the overhead of posting symmetry constraints for those symmetries that have not yet been broken; these constraints are precisely those with the potential to reduce search in the future. This suggests that the overheads incurred will usually be repaid by search reductions, and this is so in the applications described below. The exception is when there are many symmetries in a problem, an issue we address later.

3.1 Application: All Solutions of n-Queens

While the n-queens problem is no longer seen as a challenge to constraint solvers, finding *all* solutions remains hard. The chessboard has 7 rotational and mirror symmetries, excluding the identity. To eliminate symmetric solutions, it is only necessary to write 7 functions, each returning a constraint representing the symmetric equivalent of `vars[i]=j`. Two of these are shown in Fig. 1; `vars[i]=j` corresponds to $Q_i = j$, and `nQueen` is a global variable. Given IlcGenerateSym, we need only declare the relevant symmetry functions and call IlcGenerateSym rather than IlcGenerate. Results using this code are shown in Table 1. As n increases, the number of backtracks is greatly reduced and the runtime is cut by up to 75%. The solutions found are all symmetrically distinct.

```
IlcConstraint r90 (IlcIntVarArray vars,
                   IlcInt i, IlcInt j)
     {return vars[i] == nQueen-1-j;}
IlcConstraint d1 (IlcIntVarArray vars,
                  IlcInt i, IlcInt j)
     {return vars[j] == i;}
```

Figure 1. Symmetry functions for rotation through 90° and reflection in one of the main diagonals for the n-queens problem.

It is instructive to compare our approach with that of adding explicit symmetry-breaking constraints, at the outset, to the formulation of the n-queens problem. The fact that some symmetries may still exist after one or more assignments have been made must again be addressed. Consider 180° rotational symmetry, and suppose that the variables are assigned in the order $Q_1, Q_2, Q_3, \ldots$. We could add the constraint $Q_1 \leq n+1-Q_n$ to prevent, in most cases, finding two equivalent solutions. But this could still happen if, say, $Q_1 = 2$ and $Q_n = n-1$; to eliminate these, we could add a new conditional constraint: *if* $Q_1 = n+1-Q_n$ *then* $Q_2 \leq n+1-Q_{n-1}$. We might need further constraints, *if* $Q_1 = n+1-Q_n$ *and* $Q_2 = n+1-Q_{n-1}$ *then* $Q_3 \leq n+1-Q_{n-2}$, etc. To be sure of eliminating this symmetry entirely, we would have to add up to *n*/2 constraints to the formulation, each with one more condition than the last. The predicates produced

	n	No. solutions	Fails	Cpu sec
SBDS	8	12	61	0.03
	10	92	888	0.42
	12	1,787	17,940	8.29
	14	45,752	487,948	229
	16	1,846,955	17,383,754	8,400
	17	11,977,939	113,631,630	59,700
	18	83,263,591	788,699,220	392,000
No SBDS	8	92	289	0.10
	10	724	5,072	1.60
	12	14,200	103,956	34.4
	14	365,596	2,932,626	1,000
	16	14,772,512	105,445,065	35,700
	17	95,815,104	696,830,655	239,000
	18	-	-	-

Table 1. Results of SBDS applied to the n-queens problem, using ILOG Solver 4.3 on Sun Ultra 1/140's.

by Crawford *et al.* [4] also suffer from this problem. Explicit conditional constraints are cumbersome and have the disadvantage that we are committed to a fixed instantiation order. In contrast, the SBDS method automates all consideration of conditional constraints, does not produce any when they become unnecessary, and imposes no restrictions on instantiation order.

3.2 Application: LABS

The problem of low autocorrelation binary sequences (LABS) is of interest in physics, e.g. in interplanetary radar measurements. Involving only n binary variables, it has not been solved beyond about $n = 50$, the current state of the art being a branch and bound solver by Mertens [6]. The problem is to construct a binary sequence S_i of length n. Each bit in the sequence takes the value +1 or -1. The kth autocorrelation, C_k, is defined to be $\sum_{i=0}^{n-k-1} S_i * S_{i+k}$. The aim is to minimize $\sum_{k=1}^{n} {C_k}^2$. The problem is not ideally suited to constraint programming, as little propagation is possible; nevertheless, we show that symmetry can be captured efficiently and easily, resulting in significant speedup.

The first task is to model the problem. The product $S_i * S_{i+k}$ is 1 if $S_i = S_{i+k}$ and -1 if they are different. This leads to a model with an array for each k: the ith variable in the kth array is 0 if S_i and S_{i+k} are the same and 1 otherwise. The correlation C_k is the difference between the numbers of 0's and 1's in the kth array. If d_k is the number of 1's in the array, s_k is the number of 0's, and l_k is the length of the array, then d_k is the sum of the array and $l_k = s_k + d_k$. Hence, $C_k = s_k - d_k = l_k - 2d_k$. Pruning rules for this model are described in [5]. We search by setting the outermost bits first, and try the value -1 before +1.

There are three basic symmetries in LABS. The correlations C_k are unchanged if we either negate each bit in the sequence or reverse the sequence or negate the bits in even positions. Composing these symmetries yields 7 in total, excluding the identity symmetry. Any symmetry involving reversal cannot be dealt with by simple constraints, because it might not break until deep in search, and Mertens did not break all the symmetries in the problem. However, it is easy to do so using SBDS. For example, consider the most complex symmetry, the composition of reversal, flipping all bits, and flipping even bits. If the bits are numbered 0 to $n-1$, and we set bit i to +1, the symmetric decision is to set bit $n-1-i$ to -1 if $n-1-i$ is odd, and to +1 otherwise. This can be implemented straightforwardly in a one-statement Solver function.

It is easy to exclude non-reversal symmetries by insisting that the first two bits are both -1, and we did this in the non-SBDS program to ensure a fair comparison. Table 2 shows that SBDS reduces runtime by up to 1/3. Unfortunately, we were not able to beat Mertens' solver: using the same cpu resources, we would expect to find optimal sequences about 7 bits shorter than he did. LABS remains a hard challenge problem for constraint programming.

		SBDS		Non-SBDS	
n	cost	Fails	Cpu sec	Fails	Cpu sec
5	2	4	0.02	2	0.01
10	13	19	0.07	29	0.08
15	15	264	0.81	423	1.11
20	26	3,356	9.78	4,754	16.50
25	36	46,250	214	74,309	329
30	59	752,880	2,580	1,151,433	6,680
35	73	6,589,437	45,780	11,043,967	76,970

Table 2. Results of SBDS applied to determining the optimal correlation cost of the LABS problem.

4 A SHORTCUT SBDS METHOD

Given our implementation, it is straightforward to add symmetry exclusion to search, as described in section 3, if the number of symmetries is small.. Our implementation can be used easily for the restricted method proposed by Backofen and Will [2]. For example, in graph colouring, they show that it is only necessary to consider the $\binom{k}{2}$ pairwise swaps of colours rather than all $k!$ permutations. If the pairwise swaps were encoded our implementation could be used without change. Unfortunately, even a quadratic number of symmetries may be too many. In other problems, no simple restrictions on the set of symmetries may be available. In either case, direct use of SBDS is then impractical. Apart from the large number of symmetry functions to implement, many conditional constraints may be added to the constraint store, slowing down search unacceptably.

We address this difficulty by basing a methodology for developing special purpose methods on SBDS. Since this is application- and context-specific, the implementation described earlier can no longer be used, and we need to write a separate search method for each application, which will construct the appropriate symmetry-breaking constraints to add to the left branch at each choice point. The advantage of writing a special purpose search method is that symmetries need not be represented explicitly: instead symmetries can be checked by a special purpose program, typically quite cheaply. Furthermore, special purpose programs need not check all symmetries: instead we can decide which symmetries it is worthwhile to consider at each node. We call this methodology the shortcut SBDS method. The shortcut is in work done by the constraint solver compared to the full method.

We first note that if $g(var \neq val)$ is the same as $var \neq val$, the constraint (1) will be vacuous; we only consider symmetries giving non-vacuous constraints. Second, we only post unconditional constraints, i.e. $g(A)$ must provably hold, not depend on future assignments. Finally, we may restrict our attention to symmetries for which it is easy to check that $g(A)$ holds: this involves a trade-off between the effort we put into constructing the symmetry-breaking constraints and the search effort which these constraints may save. The last two simplifications may exclude some symmetries, so we can no longer guarantee that only one assignment from each symmetry class will be found. While implementation is not as easy as using IlcGenerateSym, the programmer need only consider: what symmetries leave A unchanged, and which of these change the current assignment and are easy to check?

4.1 Application: Graph Colouring

The symmetries in graph colouring are the permutations of the available colours (assuming that the graph is connected and that there are no symmetries in the graph itself). If the number of colours, k, is small, we can represent the symmetries explicitly, as described above. For large k, this will be impracticable, and we follow the approach just outlined.

Suppose a (possibly empty) set of assignments A has already been made, and that some subset of the k colours has been used in A. We first decide for which symmetries $g(A) = A$. Only permutations which change none of the colours in A will leave A unchanged. Next we consider for which of these permutations $g(var \neq val)$ is not the same as $var \neq val$. g does not change any colour that has already been assigned, so val must be an unused colour, and g must change it to another unused colour, val'.

In short, we should only add symmetry-breaking constraints if the current assignment uses a new colour. In that case, the symmetries to break are those which leave all the colours previously assigned unchanged, and change val into another colour not yet assigned. This can be expressed as a common-sense rule: if we colour a node with a previously unused colour, and this assignment fails, we should not try a different new colour for this node.

It is straightforward to modify a Solver graph colouring program to incorporate these constraints. We use an extra constrained variable for each colour to keep track of whether this colour has been used yet or not. When a choice point is created, as a disjunction between $var = val$ and $var \neq val$, we check whether val has previously been assigned in A. If not, we conjoin to $var \neq val$ a constraint $var \neq val'$ for every unused colour val'.

Problem name	cols	initial clique	Shortcut SBDS Fails	Shortcut SBDS cpu s.	No SBDS Fails	No SBDS cpu s.
myciel4	5	2	125	0.05	240	0.07
myciel5	6	2	47408	28.7	-	-
anna	11	7	20	0.12	48	0.12
miles750	31	24	660	0.70	-	-
fpsol2.i.2	30	15	2080	11.3	-	-
le450_5a	5	4	5436	33.6	12078	71.8
queen7_7	6	5	323	0.36	441	0.44

Table 3. Results for graph colouring on DIMACS instances (using a cut-off value of 100,000 fails).

A comparison between this program and one without the symmetry-breaking rule is given in Table 3 on several DIMACS graph colouring instances (http://mat.gsia.cmu.edu/COLOR/instances.html). We have to find a colouring for these graphs with the optimal number of colours, and prove that the graph cannot be coloured with fewer. To allow a fair comparison, the programs use the same variable ordering heuristic and the second program includes some simple symmetry breaking: as long as the heuristic chooses nodes at the start which are connected to each other (i.e. form a clique), the colours first assigned to these nodes are fixed. From Table 3, our method does much better, unless the size of the initial clique is close to the optimal number of colours.

The common-sense rule implemented here is one which has been implemented in specialized graph colouring programs. We stress that we derived the rule from application of a general methodology. The rule can be used in any CSP in which the values are indistinguishable and so permutable. We next turn to a domain in which the rule is less obvious.

4.2 Application: Ramsey Theory

Ramsey theory is an area of graph theory: the problem we consider is to colour the edges of the complete graph with n nodes, K_n, using c colours in such a way that there is no monochromatic triangle. The three colour problem was used by Puget [8] as an example application. For this problem, there is no solution if $n \geq 17$.

Any permutation of the colours and any permutation of the nodes is a valid symmetry. Puget dealt with symmetry by adding the following three constraints to a straightforward formulation of the problem (not all of these appear in [8]). First, the number of edges from node 0 with colour 0 is not less than the number of edges from any node i assigned colour j, for any i, j. Second, the values (i.e. colours) assigned to the edges from node 0 are non-decreasing. Third, the numbers of edges from node 0 assigned to colours $0, 1, \ldots, c-1$ are non-decreasing. These constraints imply that the edges from node 0 are assigned first, but we accepted that restriction and kept the constraints.

We modified Puget's program to apply the shortcut SBDS method. The symmetry between the colours can be dealt with as in graph colouring, but the node symmetry is more significant. We need to find permutations of the nodes which will both leave A unchanged and affect the assignment that we are about to make. However, there are too many possible permutations to consider them all. We therefore look only for permutations that can be checked easily. A transposition of two nodes l and m, which we write g_{lm}, will leave A unchanged if for all nodes h ($h \neq l, m$), e_{lh} and e_{mh} are *either* both so far uncoloured, *or* both assigned the same colour. Suppose we are about to colour edge e_{ij}, the edge joining nodes i and j, with colour v. If we can find a node i' such that transposing i' and i will leave A unchanged, i.e. $g_{i'i}(A) = A$, then we conjoin the constraint $g_{i'i}(e_{ij} \neq v)$, i.e. $e_{i'j} \neq v$, to $e_{ij} \neq v$. Similarly, if we can find a node j' such that $g_{j'j}(A) = A$, we add $e_{ij'} \neq v$. If both transpositions leave A unchanged, i.e. $g_{i'i}(A) = g_{j'j}(A) = A$, then we can also add $g_{i'i}(g_{j'j}(e_{ij} \neq v))$, i.e. $e_{i'j'} \neq v$.

To test the shortcut SBDS method, we adapted Puget's program. We used his first and third constraints, and his search strategy. The effect of his second constraint is replaced by our method. Results are shown in Table 4. Our symmetry-breaking constraints are able to replace one set of Puget's explicit constraints and indeed improve on them. Table 4(a) shows that SBDS eliminates more symmetric solutions than Puget's constraints. In Table 4(b), the same colouring for K_{16} is found in fewer fails by SBDS, though in similar runtimes, and proving that K_{17} cannot be coloured without a monochromatic triangle takes far fewer fails. Table 4(b) also gives results for a naive program in which Puget's second constraint is removed but not replaced: these results show the necessity of good symmetry reasoning. There is still a great deal of symmetry in the problem which has not been eliminated. K_4 for instance has only 9 distinct solutions. The remaining symmetries combine a permutation of the colours with a permutation of the nodes. There are clearly very many of these, and they are hard to eliminate by the methods discussed so far. Like the LABS problem, Ramsey theory remains a challenge for constraint programming, in which symmetry considerations must play a vital part.

n	SBDS			Puget		
	Solutions	Fails	Cpu	Solutions	Fails	Cpu
3	2	3	0.01	3	2	0.01
4	12	6	0.03	22	3	0.06
5	138	16	0.08	343	6	0.13
6	1284	39	0.52	7697	27	2.52
7	29027	182	11.2	252325	135	81.7

(a) Finding all solutions

n	SBDS		Puget		Naive	
	Fails	Cpu	Fails	Cpu	Fails	Cpu
16	2,030	1.61	2,437	1.40	385,608	163
17	161	0.26	636	0.27	678,976	295

(b) Finding first solution

Table 4. The (shortcut) SBDS method applied to the 3-colour Ramsey K_n problem, compared with Puget's method and a program with naive symmetry constraints.

5 CONCLUSIONS & FURTHER WORK

We have described a method for breaking symmetries during search. Using our implementation in ILOG Solver, programmers need only specify the effect of each symmetry. If the problem has many symmetries, we have given a shortcut methodology which considers only symmetries that it will be useful to break. Although we can no longer guarantee a single solution in each class, this may sometimes be achieved. Both methods are independent of the search strategy and have been shown to be worthwhile in applications.

Many avenues of further work are open. For example, computational group theory might allow us to simplify the modelling of symmetries and make reasoning with them more efficient. This may also help in cases like Ramsey theory where there are very many compositions of basic symmetries which are currently hard to deal with. Not only backtracking search can take advantage of symmetry: for example, in backjumping and learning algorithms which find nogoods, the same mechanism could rule out symmetric equivalents. We could also develop search strategies based on symmetry, which might seek to break the problem's symmetries as soon as possible, as suggested in [7], or perhaps to do precisely the opposite. Finally, as in Ramsey theory, it can be beneficial to break symmetries using both the shortcut SBDS method and ordering constraints; we should investigate how best to integrate the two approaches.

ACKNOWLEDGEMENTS

We thank Jean-Francois Puget for sending us his Solver program for the Ramsey problem. The authors are members of the APES research group (`www.apes.cs.strath.ac.uk`) and we thank the other members.

REFERENCES

[1] R. Backofen, 'Constraint Techniques for Solving the Protein Structure Prediction Problem', in *Principles and Practice of Constraint Programming - CP98*, eds., M. Maher and J.-F. Puget, LNCS 1520, pp. 72–86. Springer, (1998).

[2] R. Backofen and S. Will, 'Excluding Symmetries in Concurrent Constraint Programming', in *Principles and Practice of Constraint Programming - CP'99*, ed., J. Jaffar, LNCS 1713, pp. 73–87. Springer, (1999).

[3] C. A. Brown, L. Finkelstein, and P. W. Purdom, Jr. 'Backtrack Searching in the Presence of Symmetry', in *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, ed., T. Mora, LNCS 357, pp. 99–110. Springer-Verlag, (1988).

[4] J. Crawford, M. Ginsberg, E. Luks, and A. Roy, 'Symmetry-Breaking Predicates for Search Problems', in *Proceedings KR'96*, pp. 149–159, (1996).

[5] I. P. Gent and B. M. Smith, 'Symmetry Breaking During Search in Constraint Programming', Research Report 99.02, School of Computer Studies, University of Leeds, (1999).

[6] S. Mertens, 'Exhaustive search for low-autocorrelation binary sequences', *J. Phys. A.*, **29**, L473–L481, (1996).

[7] P. Meseguer and C. Torras, 'Solving Strategies for Highly Symmetric CSPs', in *Proceedings IJCAI'99*, pp. 400–405, (1999).

[8] J.-F. Puget, 'On the Satisfiability of Symmetrical Constrained Satisfaction Problems', in *Methodologies for Intelligent Systems (Proceedings of ISMIS'93)*, eds., J. Komorowski and Z. W. Ras, LNAI 689, pp. 350–361. Springer-Verlag, (1993).

[9] P. Roy and F. Pachet, 'Using Symmetry of Global Constraints to Speed up the Resolution of Constraint Satisfaction Problems', in *Workshop on Non Binary Constraints, ECAI-98*, (1998).