Reducing Symmetry in Matrix Models: SBDS v. Constraints

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

[1] School of Computing and Mathematics, University of Huddersfield, England.
b.m.smith@hud.ac.uk
[2] School of Computer Science, University of St. Andrews, Scotland.
ipg@dcs.st-and.ac.uk

Abstract. We discuss different methods of reducing symmetry in CSPs where the variables correspond to the cells of a matrix, and both the rows and the columns of the matrix are interchangeable. Two basic approaches are considered: adding symmetry-breaking constraints to the model, and breaking symmetry during search. We also show how the two approaches can be combined in this case. We empirically compare several different approximations to eliminating the symmetry and an exact method that eliminates the symmetry completely, but is only feasible for small matrices.

1 Introduction

In [1], Flener *et al.* discuss CSPs in which the variables correspond to the cells of a matrix and the matrix is symmetric. Specifically, the rows of the matrix are indistinguishable and so are the columns.

There are three standard ways of reducing or eliminating the symmetry in a CSP, in order to avoid the wasted search which symmetry can lead to. The first possibility is to remodel the underlying problem and to represent it by a CSP with less symmetry. Here, we are given the CSP model, and not the original problem, so this route is not open to us, although it should be borne in mind that for some types of problem, it might be possible to find a CSP representation with less symmetry than a matrix model. The second way of reducing symmetry is to add constraints to the model: ideally, these constraints will be satisfied by exactly one solution in any set of equivalent symmetric solutions. A third possibility is to break the symmetry during the search for solutions by avoiding exploring choices which are symmetric to choices already explored.

Flener *et al.* consider the second method, i.e. adding constraints to the model. They show that it is correct to impose lexicographic ordering constraints on both the rows and the columns in order to reduce the symmetry, i.e. at least one solution from each equivalence class of symmetric solutions will be found. However, they also show that these constraints are not in general sufficient to eliminate all symmetric solutions, i.e. too many solutions will be found.

Here, we consider other ways of eliminating or reducing the symmetry. We investigate the feasibility of breaking symmetry during search, and combine this approach with symmetry-breaking constraints. We examine the effectiveness of these methods in comparison with lexicographic ordering constraints.

2 Symmetry Breaking During Search

In this section, we outline the SBDS method, described fully in [2, 3]. In SBDS we aim to prevent the search visiting partial assignments which are symmetric to assignments already explored, but we do not prescribe in advance, as symmetry-breaking constraints do, the assignments which are acceptable. We describe the action of each symmetry on the decisions made at choice points during search (the decision to assign a value to a variable, or not to assign that value to the variable): once a decision has been explored, if the search backtracks to explore the alternative decision, we ensure that decisions symmetric to those already explored will not be explored in future.

At choice points during search, suppose that the alternative branches are created by the constraints $var = val$ and $var \neq val$. On backtracking to this point, having explored the $var = val$ branch, the search will explore the alternative branch, $var \neq val$. We add to this branch the constraint $g(A) \Rightarrow g(var \neq val)$ for each symmetry g. This can be interpreted as: if the symmetric equivalent of the assignments in A hold, then $g(var \neq val)$ must also hold along this branch. Since $g(A)$ involves all search variables set so far, it is potentially large. However, we note that if we extend A to the assignment $A^{+} = A + \texttt{vars[i]} = \texttt{j}$, then $g(A^{+}) = g(A) + g(\texttt{vars[i]} = \texttt{j})$. We construct a Boolean variable for each symmetry g representing whether $g(A)$ is satisfied or not: its value for $g(A^{+})$ is the conjunction of its values for $g(A)$ and $g(\texttt{vars[i]} = \texttt{j})$. Hence, we can decide incrementally whether $g(A)$ holds. Further, when the value of one of these Boolean variables becomes false, we know that the corresponding symmetry is permanently broken, and need no longer be considered, on this branch.

The n-queens problem shows how this works in practice. Suppose that the problem is modelled using n variables, $r_1, r_2, .., r_n$, representing the queen on each row, and that the domain of each variable is $\{1,2,...,n\}$, representing the n columns. The problem has seven symmetries other than identity: reflections in the horizontal and vertical axes and the main diagonals, and rotations through 90°, 180° and 270°. The effects of the symmetries on the assignment of a value j to a variable r_i are:

$$
\begin{aligned}
\mathrm{x} &: r_{n+1-i} = j \\
\mathrm{y} &: r_i = n+1-j \\
\mathrm{d1} &: r_j = i \\
\mathrm{d2} &: r_{n+1-j} = n+1-i \\
\mathrm{r90} &: r_j = n+1-i \\
\mathrm{r180} &: r_{n+1-i} = n+1-j \\
\mathrm{r270} &: r_{n+1-j} = i
\end{aligned}
$$

Suppose that $n = 8$ and the first assignment is $r_1 = 1$. The symmetric assignments are: $r_8 = 1$ (x); $r_1 = 8$ (y); $r_1 = 1$ (d1); $r_8 = 8$ (d2); $r_1 = 8$ (r90); $r_8 = 8$ (r180); $r_8 = 1$ (r270). Most of these assignments are inconsistent with $r_1 = 1$ and so the corresponding symmetry is already broken. The only remaining symmetry is d1. If we next assign $r_2 = 3$, this symmetry is also broken, since applying d1 to that assignment gives $r_3 = 2$, which is forbidden by the constraints. Further down this branch, we need no longer consider any of the symmetries.

On returning to the choice point between $r_2 = 3$ and $r_2 \neq 3$ (on failure, or to find all solutions) and taking the right branch, the symmetric equivalent of $r_2 \neq 3$ under d1 is posted, i.e. $r_3 \neq 2$.

If instead the first assignment made is $r_1 = 2$, the constraints added by SBDS take a different form. The effect of the symmetries on $r_1 = 2$ are $r_8 = 2$, $r_1 = 7$, $r_2 = 1$, $r_7 = 8$, $r_2 = 8$, $r_8 = 7$, $r_7 = 1$, respectively. Only the last three, corresponding to the rotation symmetries, are consistent with $r_1 = 2$; the others need no longer be considered.

Suppose the next assignment made is $r_2 = 4$, and having explored that branch, the search returns to explore the alternative $r_2 \neq 4$. At this point, it is not known whether the rotational symmetries are broken: it depends on whether the assignments which are the symmetric equivalents of $r_1 = 2$ will be made in the future. Hence, on the right branch, SBDS adds *conditional* constraints: for instance, for r90 we have *if* $r_2 = 8$ *then* $r_4 \neq 7$. If the search assigns $r_2 = 8$ further down this branch, the value 7 is removed from the domain of r_4.

To implement SBDS in ILOG Solver, we provide a new search function to replace Solver's `IlcGenerate`. At present, we assume that all branching points are of the form `vars[i] = j` for some array of integer variables `vars`, array index `i` and value `j` (although the implementation could be adapted for other branching strategies). 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 the search function.

3 SBDS and Matrix Models

Suppose the variables of a CSP are X_{ij} for $1 \leq i \leq m, 1 \leq j \leq n$, corresponding to the elements of a two-dimensional matrix, and in any solution both the rows and the columns can be permuted to give an equivalent solution. Any assignment is of the form $X_{ij} = k$. Any individual symmetry, g, is a permutation of the rows of the matrix, say σ_r, combined with a permutation of the columns, σ_c. (Either σ_r or σ_c, but not both, may be the identity permutation.) g transforms any matrix element i, j into another element $\sigma_r(i), \sigma_c(j)$. To specify the effect of g, we give σ_r and σ_c, and the SBDS function which takes the equivalent of $X_{ij} = k$ as input and returns the equivalent of $X_{\sigma_r(i)\sigma_c(j)} = k$.

Since there is no difficulty in specifying the effect of each individual symmetry, we can in theory use SBDS to eliminate the symmetry of the matrix altogether, and find just a set of nonsymmetric solutions. However, if we have an $m \times n$ matrix, there are $n!m! - 1$ symmetries (all row permutations, all column permutations, and all combinations) excluding identity. Eliminating the symmetry exactly would currently require that many SBDS functions, and clearly this can only be done for very small matrices: the maximum number of functions that have been used so far is around 4000, although this is a compiler limit rather than an inherent limitation of SBDS.

Hence, SBDS will in theory eliminate the symmetry completely but is impractical except for small problems; imposing lexicographic ordering constraints on the rows and columns is feasible for larger matrices but does not eliminate the symmetry. Can we find some method which will both eliminate more symmetric solutions than lexicographic ordering constraints do, in general, and at the same time be able to handle large matrices?

If we cannot use SBDS to eliminate symmetry entirely, except for small matrices, we can still use a subset of the full set of SBDS functions to reduce the symmetry to a greater or lesser extent. We know that using SBDS functions by themselves (i.e. without other forms of symmetry breaking) does not lose solutions, and if we choose the subset carefully, it may remove most of the symmetry.

If we just had row symmetry, or just had column symmetry, applying SBDS to the transpositions of pairs of rows or the transpositions of pairs of columns, respectively, would be sufficient to eliminate it. So a promising subset of the SBDS functions to try is the subset consisting of both the row transpositions *and* the column transpositions ($n(n-1)$ symmetries for matrices of size $n \times n$). If we can afford to be more expansive, we could add all combinations of a row transposition and a column transposition. This increases the number of SBDS functions considerably (to $(n(n-1)/2+1)^2 - 1$), though at least it is still polynomial.

The main factor that prevents us using the full set of SBDS functions in this case is the huge number of possible combinations of a row permutation with a column permutation. We might therefore consider de-coupling the rows and columns in some way. For instance, we might apply SBDS to the rows *or* the columns, and deal with the other dimension by adding constraints to the model. However, if we add lexicographic ordering constraints on the rows, we can no longer freely permute the columns. Hence we should not combine SBDS functions describing the column permutations with lexicographic constraints on the rows, since we might lose solutions. It is legitimate to add constraints which do not get in the way of permuting the columns; one possibility is to add constraints that the sums of the elements in each row are non-decreasing. This is a neat way of partly eliminating the row symmetry, while still leaving all the column symmetry. We can then eliminate column permutations by adding the $n! - 1$ SBDS functions. More simply still, we can get the same effect just by eliminating the transpositions of pairs of columns, which requires only $O(n^2)$ functions.

Like the lexicographic ordering constraints, these approaches reduce symmetry rather than eliminating it, and so we should expect to find that some of the matrices produced are symmetric to each other.

4 Finding nonsymmetric matrices

We have carried out a series of experiments to compare different ways of reducing symmetry, by using SBDS and/or symmetry-breaking constraints. We only consider small matrices, to allow different approximations to be compared with the results of the exact method, i.e. using SBDS to eliminate all the symmetry.

In the experiments described here, we ignore any constraints on how the matrix should be constructed: we just specify the size of the matrix and how many possible values there are for each element, and then want to find a set of matrices such that no two can be transformed into each other by permuting rows and/or columns.

For small matrices we can specify all the SBDS functions, and we are then guaranteed to get a set of nonsymmetric matrices. For our experiments, we also require the number of possible values for each matrix element to be small, so that the number of nonsymmetric matrices is not too great. Which matrices we find will depend on the variable and value ordering we use, but we will get the same number of matrices regardless. There are for instance 36 nonsymmetric matrices of size 3×3 when there are two possible values for each matrix element.

Note that the SBDS functions do not hinder the variable and value ordering; if we order the matrices in any symmetry equivalence class according to the variable & value ordering being used for the search, then the first matrix in each equivalence class is the one that will be found. On the other hand, if we are adding symmetry-breaking constraints to the model, we have to be more careful in choosing the search strategy: the variable and value ordering should be compatible with the constraints. Otherwise, the first matrix in an equivalence class according to the ordering may not satisfy the symmetry-breaking constraints. In that case, the symmetry-breaking constraints may slow down the search because an alternative solution from the equivalence class must be sought.

The lexicographic ordering constraints discussed in [1] are compatible with a variable ordering which considers the top row, left to right, 2nd row, left to right, and so on, and the values in ascending order. Hence, this ordering has been used for all the experiments described here. That this is indeed the appropriate ordering for the lexicographic ordering constraints is shown by the fact that the matrices found by SBDS (i.e. the first solution met in each equivalence class) are also found when these constraints are used.

5 Results

We compare exact SBDS with the different approximations discussed above, on 3×3 matrices with 2, 3 or 4 possible values and 4×4 matrices with 2 or 3 possible values.

We are primarily interested in how many extra matrices each approximation finds, in comparison with the exact number of nonsymmetric matrices, as given by SBDS. We also show the approximate running time for the largest set of matrices.

Strategy	3×3 matrices				4×4 matrices			
	SBDS functions	2 values	3 values	4 values	SBDS functions	2 values	3 values	Time (sec.)
Exact SBDS	35	36	738	8240	575	317	90492	186
Lexicographic constraints	-	45	1169	14155	-	650	250841	46
Non-decreasing row sums + column transpositions	3	42	1007	11174	6	567	190671	36
Row & column transpositions	6	65	1658	18822	12	1175	403809	79
Row & column transpositions + combinations	15	36	786	8985	48	353	114966	30

Table 1. Number of matrices found using different symmetry-breaking strategies

Exact SBDS is included here mainly as a benchmark, since we cannot currently use it for larger matrices. It is noteworthy that the running time is also much longer than for the other methods, even though there is more pruning of symmetric branches from the search tree.

It is clear that in these experiments, lexicographic ordering constraints did not perform as well as the combination of column transpositions in SBDS with constraints that the row sums should be non-decreasing. The lexicographic constraints allowed far more matrices to be found, and the running time was longer. The running time will depend on how the lexicographic ordering constraints are implemented: here, a weighted sum was computed for each row or column, with weights $n^{n-1}, .., n, 1$, and the weighted row or column sums were constrained to be non-decreasing. This approach would not be practicable for large n. In practice, lexicographic ordering constraints may require a large number of ordering constraints on the individual variables: if so, constraints on the row sums will be easier to post, for large n.

Table 1 shows two ways of approximating SBDS by using only a subset of the SBDS functions. Using just row and column transpositions does not work well: it finds far too many matrices, and the running time is higher than all other strategies except exact SBDS, even though the number of symmetry functions is small. On the other hand, adding all combinations of a row transposition and a column transposition gives the best approximation: it finds the smallest number of additional matrices, and the running time is small. This strategy might be a good choice, if the number of SBDS functions required were manageable. With a limit of just over 4000 functions, 11×11 matrices would be easily within reach, although the running time with so many SBDS functions might no longer be competitive with the other strategies.

Based on these results, row-sum constraints with column transpositions would seem to be the best choice for large matrices. We can evidently exchange the roles of rows and columns, so we can assume that if the matrix is rectangular, there are fewer columns (n) than rows (m). We need $n(n-1)/2$ SBDS functions, which would currently allow n to be up to 90. We also need $m-1$ ordering constraints on pairs of consecutive rows, and clearly m could be much larger than 90. Pairs of symmetric matrices are generated by this method when there is more than one row with the same sum: there is then no means to eliminate matrices which permute these rows. More complex constraints on the rows might be added to deal with some of these cases, which would give a better approximation to SBDS, though the running time might then also increase.

6 Discussion

We have so far ignored any additional constraints on what matrices are acceptable; in any application, we should expect such constraints. We can imagine that sometimes adding constraints on the sums of the rows may not distinguish between the rows at all. For instance, if we are required to find a Latin square, all the rows have the same sum (although just finding a Latin square, with no other requirement, is not difficult, and would not require all the symmetry-breaking machinery discussed here). On the other hand, additional constraints may make it impossible or unlikely to have pairs of rows with equal sums, which would make the constraints on the row sums more successful than in the cases discussed here. So although there may in theory be cases where the row-sum constraints would not break any of the row symmetry, whereas lexicographic ordering constraints on the rows would, it is not clear how practically significant this limitation would be.

In theory, we can eliminate row and column symmetry completely in matrix models with interchangeable rows and columns using SBDS, but in practice this is only possible for very small matrices. However, it is possible either to eliminate some of the row and column symmetry using SBDS, or to use SBDS to reduce the column symmetry and symmetry-breaking constraints to reduce the row symmetry (or v.v.). We have shown that these approaches give better results for the cases considered than using lexicographic ordering constraints on the rows and columns.

Sometimes, not all rows will be interchangeable, or not all columns. The methods described here would be easily adaptable to such a case. Of course, if *only* the rows or *only* the columns are symmetric, SBDS can be used directly to eliminate all the symmetry, for much larger matrices than those considered here, since it is the number of possible combinations of a row permutation with a column permutation that makes it impossible to apply SBDS exactly.

More generally, our experiments with symmetric matrices have demonstrated that when there are too many individual symmetries to write an SBDS function for each one, using the SBDS functions describing some small, suitably chosen, subset of the symmetries can in practice eliminate most of the symmetric solutions: another example where this approach is successful is described in [4]. It is also possible to combine SBDS with symmetry-breaking constraints added to the model, so long as we ensure that the two different methods do not interfere with each other. In matrix models, this has been shown to be a cost-effective way of removing much of the symmetry. In future work, we intend to make more use of group theory in symmetry-breaking: for instance, this will allow us to keep track more systematically of the symmetries which remain unbroken when constraints have been added to the original model.

References

1. P. Flener, A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Symmetry in matrix models. Technical Report APES-30-2001, APES group, 2001. Available from http://www.dcs.st-and.ac.uk/~apes/reports/apes-30-2001.ps.gz. Submitted to SymCon'01 (Symmetry in Constraints), CP2001 post-conference workshop.
2. 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, Jan. 1999.
3. I. P. Gent and B. M. Smith. Symmetry Breaking During Search in Constraint Programming. In W. Horn, editor, *Proceedings ECAI'2000*, pages 599–603, 2000.

4. B. M. Smith. Reducing Symmetry in a Combinatorial Design Problem. Research Report 2001.01, School of Computing, University of Leeds, Jan. 2001.