Breaking Row and Column Symmetries in Matrix Models

Pierre Flener[1], Alan M. Frisch[2], Brahim Hnich[3], Zeynep Kızıltan[3], Ian Miguel[2], Justin Pearson[1], and Toby Walsh[4]

[1] Dept of Information Tech, Uppsala University, Box 337, 751 05 Uppsala, Sweden
PierreF@csd.uu.se, justin@docs.uu.se
[2] Department of Computer Science, University of York, York YO10 5DD, England
Frisch@cs.york.ac.uk, IanM@cs.york.ac.uk
[3] Dept of Information Science, Uppsala University, Box 513, 751 20 Uppsala, Sweden
Zeynep.Kiziltan@dis.uu.se, Brahim.Hnich@dis.uu.se
[4] Cork Constraint Computation Centre, University College Cork, Cork, Ireland
TW@4c.ucc.ie

Abstract. We identify an important class of symmetries in constraint programming, arising from matrices of decision variables where rows and columns can be swapped. Whilst lexicographically ordering the rows (columns) breaks all the row (column) symmetries, lexicographically ordering both the rows and the columns fails to break all the compositions of the row and column symmetries. Nevertheless, our experimental results show that this is effective at dealing with these compositions of symmetries. We extend these results to cope with symmetries in any number of dimensions, with partial symmetries, and with symmetric values. Finally, we identify special cases where all compositions of the row and column symmetries can be eliminated by the addition of only a linear number of symmetry-breaking constraints.

1 Introduction

Modelling is one of the most difficult parts of constraint programming. Freuder has identified it as the "last frontier" [9]. One source of difficulty is dealing with symmetry efficiently and effectively. Symmetry occurs in many assignment, scheduling, configuration, and design problems. Identical machines in a factory, equivalent time periods, and equally skilled workers are just a few of the items likely to introduce symmetry into a constraint satisfaction problem (CSP). If we ignore symmetry, a solver will waste time considering symmetric, that is essentially equivalent, assignments. As there is often a (super-)exponential number of symmetric solutions, this can be very costly. To help tackle this problem, we identify an important class of symmetries that occur frequently in CSPs. We show how constraints can be added to a model to break some of these symmetries, and analyse the effectiveness of these methods theoretically and experimentally.

2 Matrix Models and Symmetry

A *matrix model* is a constraint program that contains one or more matrices of decision variables. For example, a natural model of the round robin tournament scheduling problem (prob026 in CSPlib, at www.csplib.org) has a 2-dimensional (2-d) matrix of variables, each of which is assigned a value corresponding to the match played in a given week and period [21]. In this case, the matrix is obvious in the modelling of the problem: we need a *table* of fixtures. However, many other problems that are less obviously defined in terms of matrices of variables can be effectively represented and efficiently solved using matrix models [6]. For example, the rack configuration problem (prob031) can be modelled with a 2-d 0/1 matrix representing which cards go into which racks (a model with a 3-d matrix is given in [13]).

Symmetry is an important aspect of matrix models. Symmetry often occurs because groups of objects within a matrix are indistinguishable. For example, in the round robin tournament scheduling problem, weeks and periods are indistinguishable. We can therefore permute any two weeks or any two periods in the schedule. That is, we can permute any two rows or any two columns of the associated matrix, whose index sets are the weeks and periods. A *symmetry* is a bijection on decision variables that preserves solutions and non-solutions. Two variables are *indistinguishable* if some symmetry interchanges their rôles in all solutions and non-solutions.

Two common types of symmetry in matrices are row symmetries and column symmetries. The two examples above have row and column symmetries. A *row* (*column*) *symmetry* of a 2-d matrix is a bijection between the variables of two of its rows (columns) that preserves solutions and non-solutions. Two rows (columns) are *indistinguishable* if their variables are pairwise indistinguishable due to a row (column) symmetry. Note that the rotational symmetries of a matrix are neither row nor column symmetries. A matrix model *has row* (*column*) *symmetry* iff all the rows (columns) of one of its matrices are indistinguishable. A matrix model *has partial row* (*column*) *symmetry* iff strict subset(s) of the rows (columns) of one of its matrices are indistinguishable. All these definitions can be extended to matrices with any number of dimensions. A *symmetry class* is an equivalence class of assignments, where two assignments are equivalent if there is some symmetry mapping one assignment into the other. (In group theory, such equivalence classes are referred to as *orbits.*)

Many row and column symmetries have been observed [6], such as in matrix models for the balanced incomplete block design problem (prob028 in CSPlib, at www.csplib.org), the steel mill slab design problem [6], the social golfers problem (prob010), the template design problem (prob002), the progressive party problem (prob013), and (as argued above) the rack configuration problem (prob031) as well as the round robin tournament scheduling problem (prob026).

3 Breaking Symmetry

There are a number of ways of dealing with symmetry in constraint programming (see Section 7 for a longer discussion). A popular approach is to add constraints that break some of the symmetries [16, 3].

One common method to break symmetry is to impose a constraint that orders the symmetric objects. To break all the row (column) symmetries, we can order the rows (columns) lexicographically. The rows (columns) in a 2-d matrix are *lexicographically ordered* if each row (column) is lexicographically smaller (denoted $\leq_{lex}$) than the next (if any), and *anti-lexicographically ordered* if each row (column) is lexicographically larger than the next (if any). As a lexicographic ordering is total, adding lexicographic (or anti-lexicographic) ordering constraints on the rows (columns) will break all the row (column) symmetries.

Whilst breaking all the row symmetries *or* all the column symmetries in a matrix is possible with lexicographic ordering constraints, breaking *both* the row and the column symmetries seems difficult since the rows and columns intersect. Lexicographically ordering the rows will tend to put the columns into lexicographic order. However, it does not always order the columns lexicographically, and lexicographically ordering the columns can then disrupt the lexicographic ordering on the rows.

Example 1. Consider a 3×4 matrix of 0/1 variables, x_{ij}, with the constraints that $\sum_{ij} x_{ij} = 7$ and $\sum_i x_{ij} \cdot x_{ik} \leq 1$ for $j \neq k$ (i.e., the dot product of any two rows is 1 or less). This model has both row and column symmetry. A solution with lexicographically ordered rows is:

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

Lexicographically ordering the columns now gives the solution:

$$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

However, this destroys the lexicographic ordering on the rows. Reordering the last two rows gives a solution that is lexicographically ordered along both the rows and the columns:

$$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

One can even construct examples that need several sequential rounds of ordering the rows and then the columns, although the following theorem shows that this process always terminates. During search, both the row and column lexicographic ordering constraints actually work in parallel. The following theorem shows that, whether this ordering is done sequentially or in parallel, there always is a solution with the rows and columns *both* in lexicographic order.

Theorem 1. *For a matrix model with row and column symmetry in some 2-d matrix, each symmetry class of assignments has an element where both the rows and the columns of that matrix are lexicographically ordered.*

Proof: We order 2-d matrices by lexicographically ordering the sequences formed by appending their rows together in top-down order. Lexicographically ordering two rows replaces a larger row at the front of this sequence by a smaller row from further behind. Hence, ordering two rows moves us down the matrix ordering. Lexicographically ordering two columns also moves us down this matrix ordering. Indeed, the two columns have some values (if any) in common at the top and swapping the columns thus does not affect the matrix ordering when just considering the corresponding top rows; also, in the top-most row (if any) where the two columns differ, the value in the left column is then replaced by a smaller value from the right column, as the latter was lexicographically smaller than the left column, making that row lexicographically smaller. This moves us down the matrix ordering, as the first changed row (if any) is replaced in the sequence by a smaller one. Furthermore, the matrix ordering is finite, as there are only a finite number of permutations of the values in a matrix, and bounded below, namely by a matrix whose rows and columns are lexicographically ordered. So we cannot move down the matrix ordering indefinitely, and will find a matrix in which all the rows and columns are lexicographically ordered. □

This result shows that we can always lexicographically order both the rows and the columns. Dually, we can always anti-lexicographically order both the rows and the columns. However, we cannot always lexicographically order the rows and anti-lexicographically order the columns. Lexicographically ordering the rows will tend to push the largest values to the bottom-left of the matrix. Anti-lexicographically ordering the columns will tend to push the larger values to the top-right. For this reason, the two orders can conflict.

Example 2. Consider a 2×2 matrix of 0/1 variables, x_{ij}, with the constraints that $\sum_i x_{ij} = 1$ and $\sum_j x_{ij} = 1$ (i.e., every row and column has a single 1). This model has both row and column symmetry, and has two symmetric solutions:

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

The first solution has rows and columns that are lexicographically ordered, whilst the second has rows and columns that are anti-lexicographically ordered. There is thus no solution in which the rows are lexicographically ordered and the columns are anti-lexicographically ordered.

Lexicographically ordering the rows (columns) breaks all the row (column) symmetries. However, lexicographically ordering both the rows and the columns does *not* break all the compositions of the row and column symmetries.

Example 3. Consider a 3×3 matrix of 0/1 variables, x_{ij}, with the constraints that $x_{i1} + x_{i2} + x_{i3} \geq 1$ and $\sum_{ij} x_{ij} = 4$. This model has both row and column symmetry. The following two symmetric solutions have lexicographically ordered rows and columns:

$$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}, \quad \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix}$$

These solutions are symmetric, as one can move from one to the other by swapping the first two rows and the last two columns. Swapping any rows or columns *individually* breaks the lexicographic ordering. Thus, lexicographically ordering both the rows and the columns does not break all the compositions of the row and column symmetries.

However, our experimental results (see Section 6) suggest that lexicographically ordering both the rows and the columns breaks enough symmetries to be useful practically.

4 Extensions

We now consider a number of extensions that extend the utility of our results considerably.

4.1 Higher dimensions

Many problems can be effectively modelled and efficiently solved using matrix models with a matrix of more than two dimensions. For example, the social golfers problem can be modelled with a 3-d 0/1 matrix whose dimensions correspond to weeks, groups, and players [17]. A variable x_{ijk} in this matrix is 1 iff in week i player j plays in group k. This matrix model has symmetries along each of the three dimensions: the weeks are indistinguishable, and so are the groups and players. We now generalise the lexicographic ordering constraint to any number of dimensions to break some of these symmetries.

Consider a 2-d matrix. If we look along a particular dimension, we see 1-d vectors at right angles to this axis. To break the symmetries, we order these vectors lexicographically. Now consider a 3-d matrix. If we look along a particular dimension, we see 2-d slices of the matrix that are orthogonal to this axis. To break the symmetries, we need to order these slices. One way is to flatten the slices onto vectors and lexicographically order these. In n dimensions, we see slices that are $n-1$ dimensional hypercubes, which can be compared by flattening onto vectors and lexicographically ordering these.

Definition 1. *An n-dimensional matrix X, with $n \geq 1$, is* multi-dimensionally lexicographically ordered *iff the following conditions hold:*

$$\begin{array}{c} \forall i \ \text{flatten}(X[i][\ ]\ldots[\ ]) \leq_{lex} \text{flatten}(X[i+1][\ ]\ldots[\ ]) \\ \forall j \ \text{flatten}(X[\ ][j]\ldots[\ ]) \leq_{lex} \text{flatten}(X[\ ][j+1]\ldots[\ ]) \\ \cdots \\ \forall k \ \text{flatten}(X[\ ][\ ]\ldots[k]) \leq_{lex} \text{flatten}(X[\ ][\ ]\ldots[k+1]) \end{array}$$

where $X[\,]\ldots[\,][i][\,]\ldots[\,]$ *denotes the* $n-1$ *dimensional hypercube obtained by taking the slice of* X *at position* i *in the dimension where* $[i]$ *appears in* $[\,]\ldots[\,][i][\,]\ldots[\,]$*, and where* flatten *is used to flatten a slice of a matrix into a* 1*-d vector and is defined by:*

$$\begin{aligned}
\text{flatten}(X[1..m]) &= X[1..m] \\
\text{flatten}(X[1..m][\,]\ldots[\,]) &= \text{append}(\ \text{flatten}(X[1][\,]\ldots[\,]), \\
&\qquad\qquad\quad \ldots, \\
&\qquad\qquad\quad \text{flatten}(X[m][\,]\ldots[\,]))
\end{aligned}$$

with append$(V_1, \ldots, V_n)$ *denoting the left-to-right concatenation of the* 1*-d vectors* $V_1, \ldots, V_n$.

As in the 2-d case, we can show that this multi-dimensional lexicographic ordering breaks some of the symmetries. Unfortunately, it does not break all the symmetries as the 2-d counter-examples generalise to other numbers of dimensions.

Theorem 2. *For a matrix model with symmetry along each dimension in some* n*-dimensional matrix, where* $n \geq 1$*, each symmetry class of assignments has an element where that matrix is multi-dimensionally lexicographically ordered.*

Proof: A proof for the 3-d case is in [5]; it generalises to any number of dimensions. □

4.2 Partial symmetry

We may only have partial row or column symmetry in a matrix model, namely when only strict subset(s) of the rows or columns of one of its matrices are indistinguishable. We here show through an example how to address this.

Example 4. In a 2-d 0/1 matrix model of the rack configuration problem, only the columns that correspond to racks of the same type are indistinguishable. Suppose there are 10 racks, where the first 4 racks are of a first type, the next 3 racks are of another type, and the last 3 racks are of a third type. Then the following candidate solutions:

$$\begin{array}{r}
\longrightarrow \text{racks} \longrightarrow \\
0\,0\,0\,0\ \ 0\,1\,0\ \ 0\,0\,0 \\
0\,0\,0\,0\ \ 1\,0\,0\ \ 0\,0\,0 \\
\downarrow 0\,1\,0\,0\ \ 0\,0\,0\ \ 0\,0\,0 \\
\text{cards}\ 1\,0\,0\,0\ \ 0\,0\,0\ \ 0\,0\,0 \\
\downarrow 0\,0\,0\,0\ \ 0\,0\,0\ \ 1\,0\,0 \\
1\,0\,0\,0\ \ 0\,0\,0\ \ 0\,0\,0
\end{array}
\qquad
\begin{array}{r}
\longrightarrow \text{racks} \longrightarrow \\
0\,0\,0\,0\ \ 0\,1\,0\ \ 0\,0\,0 \\
1\,0\,0\,0\ \ 0\,0\,0\ \ 0\,0\,0 \\
\downarrow 0\,1\,0\,0\ \ 0\,0\,0\ \ 0\,0\,0 \\
\text{cards}\ 0\,0\,0\,0\ \ 1\,0\,0\ \ 0\,0\,0 \\
\downarrow 0\,0\,0\,0\ \ 0\,0\,0\ \ 1\,0\,0 \\
0\,0\,0\,0\ \ 1\,0\,0\ \ 0\,0\,0
\end{array}$$

are not symmetric, because the first and fifth columns have been swapped although they do not pertain to the same rack type. We cannot lexicographically order all the columns in such a situation, as that would here amount to requiring that all the racks are of the same type. However, we can use fewer lexicographic ordering constraints to break some of the underlying symmetries: for each subset of rows (columns) that are indistinguishable, we only state lexicographic ordering constraints between these rows (columns).

We can also extend the 0/1 domain of the decision variables in the matrix, and add a first row for a dummy card that is constrained as follows, say:

$$
\begin{array}{r|cccc|ccc|ccc}
 & \multicolumn{10}{c}{\longrightarrow \text{racks} \longrightarrow} \\
 & 2 & 2 & 2 & 2 & 3 & 3 & 3 & 4 & 4 & 4 \\
 & . & . & . & . & . & . & . & . & . & . \\
 & . & . & . & . & . & . & . & . & . & . \\
\downarrow & . & . & . & . & . & . & . & . & . & . \\
\text{cards} & . & . & . & . & . & . & . & . & . & . \\
\downarrow & . & . & . & . & . & . & . & . & . & . \\
 & . & . & . & . & . & . & . & . & . & .
\end{array}
$$

Lexicographically ordering all the columns will now keep the columns pertaining to racks of the same type together and thus only break all the symmetries arising from indistinguishable rack types.

4.3 Value symmetry

We can deal with symmetric values using the techniques we have developed above for dealing with symmetric variables. A variable x of an n dimensional matrix that takes a single value (or a set of values) from a domain of indistinguishable values $v_1, \ldots, v_m$ can be replaced by a vector $[x_1, \ldots, x_m]$ of 0/1 variables, with the semantics $x_i = 1 \leftrightarrow v_i = x$ (or $x_i = 1 \leftrightarrow v_i \in x$). In other words, a variable taking a value (or a set of values) among indistinguishable values is replaced by a characteristic function, whose variables take values that are not indistinguishable. This converts indistinguishable values into indistinguishable variables, which become a new dimension in the now $n+1$ dimensional matrix.

Example 5. Consider a 2-d matrix model of the progressive party problem [19]. A variable x_{ij} in its matrix takes as value the host boat visited by guest i in period j. Now, host boats of the same capacity are indistinguishable. We can turn this partial value symmetry into a partial variable symmetry by channelling into a new 3-d 0/1 matrix that has no value symmetry. A variable y_{ijk} in this new matrix is 1 iff the host boat k is visited by guest i in period j. Channelling constraints of the form $y_{ijk} = 1 \leftrightarrow k = x_{ij}$ can thus link the two matrices. The new matrix model has partial symmetry along the third dimension of its 3-d matrix. We can therefore use lexicographic ordering constraints to break these symmetries. Note that we do not always need to channel between the two matrices and could thus replace the old matrix by the new one. However, it is quite often the case that some constraints are more easily expressed on the original matrix, and this is the case here.

The advantage of this approach is that we can use the multi-dimensional lexicographic ordering to deal simultaneously with symmetric variables and symmetric values. An alternative approach to breaking value symmetry is described in [11], but this method currently assumes that all values in a domain are symmetrical. We can also use the techniques outlined in the previous sub-section to deal with values that are only partially symmetric. Freuder addresses the case of interchangeable values [8], but with respect to individual variables as opposed to symmetries that hold globally between values. Again, we can support this situation by ordering sub-rows or sub-columns.

5 Breaking All the Symmetries

It is always possible to break all the symmetries. In [3], a method is presented for adding a lexicographic ordering constraint for each symmetry of the problem.

Example 6. The set of all compositions of the row and column symmetries of a 2×3 matrix

$$\begin{pmatrix} x_1 & x_2 \\ x_3 & x_4 \\ x_5 & x_6 \end{pmatrix}$$

can be broken by the following 11 constraints:

$$[x_1, x_2, x_3, x_4, x_5, x_6] \leq_{lex} [x_3, x_4, x_1, x_2, x_5, x_6], \; \textit{that is } [x_1, x_2] \leq_{lex} [x_3, x_4]$$
$$[x_1, x_2, x_3, x_4, x_5, x_6] \leq_{lex} [x_1, x_2, x_5, x_6, x_3, x_4], \; \textit{that is } [x_3, x_4] \leq_{lex} [x_5, x_6]$$
$$[x_1, x_2, x_3, x_4, x_5, x_6] \leq_{lex} [x_2, x_1, x_4, x_3, x_6, x_5], \; \textit{that is } [x_1, x_3, x_5] \leq_{lex} [x_2, x_4, x_6]$$
$$[x_1, x_2, x_3, x_4, x_5, x_6] \leq_{lex} [x_4, x_3, x_6, x_5, x_2, x_1], \; \textit{that is } [x_1, x_2, x_3, x_4, x_5] \leq_{lex} [x_4, x_3, x_6, x_5, x_2]$$
$$[x_1, x_2, x_3, x_4, x_5, x_6] \leq_{lex} [x_6, x_5, x_2, x_1, x_4, x_3], \; \textit{that is } [x_1, x_2, x_3, x_4, x_5] \leq_{lex} [x_6, x_5, x_2, x_1, x_4]$$
$$[x_1, x_2, x_3, x_4, x_5, x_6] \leq_{lex} [x_2, x_1, x_6, x_5, x_4, x_3], \; \textit{that is } [x_1, x_3, x_4] \leq_{lex} [x_2, x_6, x_5]$$
$$[x_1, x_2, x_3, x_4, x_5, x_6] \leq_{lex} [x_4, x_3, x_2, x_1, x_6, x_5], \; \textit{that is } [x_1, x_2, x_5] \leq_{lex} [x_4, x_3, x_6]$$
$$[x_1, x_2, x_3, x_4, x_5, x_6] \leq_{lex} [x_6, x_5, x_4, x_3, x_2, x_1], \; \textit{that is } [x_1, x_2, x_3] \leq_{lex} [x_6, x_5, x_4]$$
$$[x_1, x_2, x_3, x_4, x_5, x_6] \leq_{lex} [x_5, x_6, x_3, x_4, x_1, x_2], \; \textit{that is } [x_1, x_2] \leq_{lex} [x_5, x_6]$$
$$[x_1, x_2, x_3, x_4, x_5, x_6] \leq_{lex} [x_3, x_4, x_5, x_6, x_1, x_2], \; \textit{that is } [x_1, x_2, x_3, x_4] \leq_{lex} [x_3, x_4, x_5, x_6]$$
$$[x_1, x_2, x_3, x_4, x_5, x_6] \leq_{lex} [x_5, x_6, x_1, x_2, x_3, x_4], \; \textit{that is } [x_1, x_2, x_5, x_6] \leq_{lex} [x_5, x_6, x_1, x_2]$$

The first two constraints arise from the indistinguishability of the first two rows and the last two rows, respectively, whereas the third constraint arises from the indistinguishability of the two columns. The remaining constraints arise from the compositions of these row and column symmetries. These constraints were obtained by first determining the $2! \cdot 3! = 12$ permutations of the vector $[x_1, x_2, x_3, x_4, x_5, x_6]$ obtained by building the 3! concatenations of the row vectors for each of the 2! permutations inside the rows, and then constraining an arbitrary one of these 12 permutations, namely $[x_1, x_2, x_3, x_4, x_5, x_6]$ here, to be the lexicographically smallest one.

In general, an $m \times n$ matrix has $m! \cdot n! - 1$ symmetries except identity, generating thus a super-exponential number of lexicographic ordering constraints. Hence this approach is not always practical, so we now identify three special cases where all compositions of the row and column symmetries can be broken by a polynomial (and even linear) number of constraints.

First consider the case where all the values in the matrix are distinct. Such matrix models are common. For example, this happens in the single-round tournament scheduling problem, when the matrix entries are ordered pairs of teams.

Theorem 3. *If a matrix model with row and column symmetry in some* 2*-d matrix, as well as with a constraint requiring all the values in that matrix to be distinct, has a solution, then each symmetry class of solutions has a unique member with the largest value placed in the bottom-right corner as well as the last row and the last column ordered.*

Proof: Given a solution, the row occupied by the largest value contains distinct values that can be permuted by ordering the columns. By ordering this row, we break all possible column symmetries and fix the sequence of the columns. Similarly, the column occupied by the largest value contains distinct values that can be permuted by ordering the rows. By now ordering this column, we break all possible row symmetries, and fix the sequence of the rows, while placing the largest value in the bottom-right corner of the matrix. All the compositions of the row and column symmetries are thus broken, because we have constructed a unique symmetric solution. □

It is therefore the symmetries between identical values that make it difficult to break all the compositions of the row and column symmetries.

In fact, our proof shows that we break all the symmetries even if the other rows and columns contain repeated values. Ordering the row and column with the largest value will fix all the other values in the matrix in a unique way. So we do not need every value in the matrix to be distinct (although this is sufficient to make the row and column with the largest value contain no repeated values).

Next, even when matrices have repeated values, it is still possible in certain situations to break all symmetries by means of a polynomial number of symmetry-breaking constraints. In particular, this is the case for 2-d 0/1 matrices with a single 1 in each row. Such matrix models are quite common. For example, the 2-d matrix we used in the rack configuration problem has this form.

Theorem 4. *If a matrix model with row and column symmetry in some* 2*-d* 0/1 *matrix, as well as with a constraint requiring a single* 1 *in each row of that matrix, has a solution, then each symmetry class of solutions has a unique solution with the rows ordered lexicographically as well as the columns ordered lexicographically and by their sums.*

Proof: Given a solution, by Theorem 1, there is a symmetric solution with the rows and columns lexicographically ordered. In that solution, the top-right corner must contain a 1. Suppose that in the next row down, the 1 occurs to the right of where it does in this row. Then the next row is not lexicographically larger. Suppose that it occurs more than one column across to the left. Then the columns in between are not lexicographically larger. Hence, the 1 in the next row down must occur either directly below or one column to the left. The only freedom is in how many consecutive rows have 1s in the same column. This symmetry is broken by ordering the sums of the columns. All the compositions of the row and column symmetries are broken, because we have constructed a unique symmetric solution. $\Box$

Note that we can have the column sums in increasing or decreasing order, depending on which is preferable.

Finally, all the symmetries can be broken with a linear number of constraints when all the rows, seen as multisets, are distinct. We say that a vector v_1 is *multiset-lexicographically smaller than* another vector v_2 if $sort(v_1) \leq_{lex} sort(v_2)$, where $sort(v)$ denotes the ordered permutation of vector v. For instance, the vector $[0, 1, 2, 1, 1]$ is multiset-lexicographically smaller than the vector $[0, 3, 1, 1, 1]$ because $[0, 1, 1, 1, 2] \leq_{lex} [0, 1, 1, 1, 3]$.

Theorem 5. *If a matrix model with row and column symmetry in some 2-d matrix, as well as with a constraint requiring all the rows of that matrix to be distinct as multisets, has a solution, then each symmetry class of solutions has a unique solution with the rows multiset-lexicographically ordered and the columns lexicographically ordered.*

Proof: Given a solution, we can first multiset-lexicographically order the rows. Because the rows are distinct as multisets, this fixes the order of the rows. We can now order the columns lexicographically without changing the multiset of any row. All the compositions of the row and column symmetries are broken, because we have constructed a unique symmetric solution. $\Box$

6 Experimental Results

To test the ability of lexicographic ordering constraints to break the compositions of row and column symmetries, we ran some experiments on balanced incomplete block design (BIBD) generation. This is a standard combinatorial problem from design theory. It has applications in experimental design and cryptography (see prob028 at www.csplib.org for more details).

A BIBD is an arrangement of v distinct objects into b blocks, such that each block contains exactly k distinct objects, each object occurs in exactly r different blocks, and every two distinct objects occur together in exactly λ blocks. A BIBD instance is thus determined by its parameters $\langle v, b, r, k, \lambda \rangle$. One way of modelling a BIBD is in terms of its incidence matrix, which is a $v \times b$ 0/1 matrix with exactly r ones per row, k ones per column, and with a scalar product of λ between any pair of distinct rows [6]. This matrix model has row and column symmetry since we can permute any rows or columns freely without affecting any of the constraints. This kind of symmetry is often partially broken by setting the

Instance	distinct #sol	row & col lex #sol	row & col lex time	set 1st row & col #sol	set 1st row & col time	row lex #sol	row lex time	col lex #sol	col lex time
$\langle 7, 7, 3, 3, 1\rangle$	1	1	1.05	216	8	30	3	30	4
$\langle 6, 10, 5, 3, 2\rangle$	1	1	0.95	$17, 280$	332	$60, 480$	$3, 243$	12	2
$\langle 7, 14, 6, 3, 2\rangle$	4	24	10.63	$\geq 90, 448$	–	$\geq 68, 040$	–	465	55
$\langle 9, 12, 4, 3, 1\rangle$	1	8	28.14	$\geq 5, 340$	–	≥ 342	–	840	$1, 356$
$\langle 8, 14, 7, 4, 3\rangle$	4	92	171.00	$\geq 5, 648$	–	$\geq 2, 588$	–	$\geq 5, 496$	–
$\langle 6, 20, 10, 3, 4\rangle$	unknown	21	10.30	$\geq 538, 272$	–	$\geq 429, 657$	–	73	20

Table 1. Experimental results on BIBD instances

first row and the first column, as this is a cheap but effective method. However, this breaks less symmetry than lexicographically ordering both the rows and the columns, as shown next.

Table 1 shows our experimental results on some BIBD instances. We used the ECLIPSE toolkit as it has a lexicographic ordering constraint. The instances in this table are also used in [14, 15]. We only present a representative sample of our experiments. We enforced a lexicographic ordering between neighbouring pairs of rows and columns (row & col lex). We also include the results when we set the first row and the first column (set 1st row & col), as well as when we impose lexicographic ordering constraints only on the rows (row lex) or only on the columns (col lex). For each instance, we show the number of distinct solutions (distinct #sol), the number of symmetric solutions being always in excess of 2.5 million, as well as the total number of solutions found (#sol) and the run-times (time, in seconds, or a "–" whenever 1 clock hour was exceeded, in which case we report the number of solutions found at that moment) for each of the four symmetry-breaking techniques listed above.

With the row and column lexicographic ordering constraints, we labelled along one row and then down one column, and so on, as this is more efficient than labelling just along the rows or just down the columns, on these instances. However, there are some instances (not shown in the table) where labelling along the rows is much more efficient than labelling along the rows and columns. With the first row and column set, the best labelling strategy varies from instance to instance; we report the best results achieved among the three strategies. Indeed, the objective was to get, within reasonable amounts of time, numbers of solutions that can be compared, rather than to compare the times needed to do so. The times are only indicated to reveal that our symmetry-breaking techniques are cost-effective compared to an existing one. With row lexicographic ordering constraints, the best strategy is to label the columns, and with column lexicographic ordering constraints, the best strategy is to label the rows.

The table reveals that the column lexicographic ordering constraints are much more efficient than the row ones. This is true for many other instances (that are not shown in the table). We conjecture that the scalar product constraint so tightly constrains the rows that little work is left to be done by the row lexicographic ordering constraints. The column lexicographic ordering constraints act orthogonally and so are more constraining. The results also confirm that lexicographically ordering the rows and columns can break most of the compositions of the row and column symmetries.

In [15], a binary CSP model encoded in SAT that breaks symmetries in a different way was proposed to solve several BIBD instances using SATZ, WSAT, and CLS. All its instances could be solved fast enough with our 2-d 0/1 matrix model using row and column lexicographic ordering constraints. For example, our model solves the instance $\langle 8, 14, 7, 4, 3 \rangle$ in 171 seconds, while this instance was not solved in several hours with any algorithm or encoding in [15].

6.1 Experiments with Value Symmetry

To test the efficacy of channelling to an 0/1 matrix in order to break value symmetry with lexicographic ordering constraints, we experimented with Schur's Lemma (prob 015 in CSPlib). The problem is to put n balls, labelled $\{1,...n\}$, into 3 boxes so that for any triple of balls (x, y, z) with $x + y = z$, not all are in the same box. A natural model consists of a one-dimensional matrix of variables with domain size 3, each element of which corresponds to a particular box. The boxes, and therefore the values, are symmetrical. We tested this model with no symmetry breaking and with Gent's method [11]. A second model channels to an 0/1 matrix of balls $\times$ boxes. In this model, a row corresponds to the contents of a box. Hence, we can use lexicographic row ordering to break the symmetry.

Table 2 summarises the results. Both symmetry breaking methods result in a dramatic reduction in the number of solutions discovered and the size of search tree. Gent's method appears to propagate slightly before the lexicographic approach, hence the (negligible) difference in terms of fails and choices. Given three boxes, we require just two lexicographic ordering constraints between adjacent rows of the 0/1 matrix. Although Gent's method requires fewer extra variables than the lexicographic approach, each has a relatively large domain. This coupled with $O(n)$ extra constraints results in the gap in overall performance.

n	No Symmetry Breaking				Gent's Method				Lexicographic			
	Fails	Choices	Time	Solns	Fails	Choices	Time	Solns	Fails	Choices	Time	Solns
15	7878	25451	0.6s	17574	1313	4241	0.6s	2929	1317	4245	0.2s	2929
16	10356	25067	0.6s	14712	1726	4177	0.6s	2452	1730	4181	0.2s	2452
17	11970	24029	0.6s	12060	1995	4004	0.7s	2010	1999	4008	0.2s	2010
18	11970	19025	0.6s	7056	1995	3170	0.7s	1176	1999	3174	0.2s	1176
19	12132	16391	0.6s	4260	2022	2731	0.7s	710	2026	2735	0.2s	710
20	11976	14117	0.5s	2142	1996	2352	0.8s	357	2000	2356	0.2s	357
21	10878	11783	0.5s	906	1813	1963	0.7s	151	1817	1967	0.2s	151
22	10206	10397	0.5s	192	1701	1732	0.8s	32	1705	1736	0.2s	32
23	9738	9755	0.5s	18	1623	1625	0.8	3	1627	1629	0.2s	3
24	9072	9071	0.5s	0	1512	1511	0.8	0	1516	1515	0.2s	0

Table 2. Experimental results on Schur's Lemma

7 Related Work

There is currently much interest in symmetry in constraint satisfaction problems. The existing approaches can be broadly categorised into five types.

The first approach, deployed here, adds symmetry-breaking constraints to the model in an attempt to remove some symmetries *before* search starts [16, 3].

A second method adds symmetry-breaking constraints *during* search to prune symmetric branches (e.g., [1], the global cut framework (GCF) [7], and symmetry-breaking during search (SBDS) [12]). A disadvantage of methods like SBDS is that, at each node in the search tree, a constraint for each symmetry is added, but that, for matrix models, there is a super-exponential number of symmetries that have to be treated. Recently, promising results on combining the dynamic SBDS with our static pre-search approach [5] have been reported for matrix models [20], especially for combined methods that break some of the symmetries using row sum ordering and column lexicographic ordering.

Third, in symmetry-breaking via dominance detection (SBDD) [4], the *search procedure* is modified by adding a dominance check that checks if the current assignment is symmetric to a previously encountered assignment. Such a dominance check is problem-specific.

A fourth approach is to break symmetry by means of a *heuristic variable-ordering* that directs the search towards subspaces with a high density of non-symmetric solutions (e.g., [14]).

Lastly, it is sometimes possible to *remodel* a problem to remove some symmetries, for example via the use of set variables. However, this can produce a more complex model [18].

All of these approaches would benefit from an efficient means of automatic symmetry detection. However, symmetry detection has been shown to be graph-isomorphism complete in the general case [2]. Therefore, it is often assumed that the symmetries are known by the user. Since matrices of decision variables are common in constraint programs [6], and since rows and columns in such matrices are often indistinguishable, making matrices first-class objects in the modelling language would give a heuristic symmetry-detection technique obvious clues as to where to look.

8 Conclusions

We have identified an important class of symmetries in constraint models: row and column symmetries. We have shown that we can lexicographically order both the rows and the columns to break some of these symmetries. Whilst lexicographically ordering the rows breaks all the row symmetries and lexicographically ordering the columns breaks all the column symmetries, lexicographically ordering both the rows and the columns fails to break all the compositions of these symmetries. Nevertheless, our experimental results show that this can be effective at dealing with these compositions of the row and column symmetries.

We have extended these results to cope with symmetries in any number of dimensions, with partial symmetries, and with symmetric values. Finally, we have identified a number of special cases where all compositions of the row and column symmetries can be broken by means of adding only a linear number of constraints.

Having established the utility of lexicographic ordering, there is a clear need for efficient methods for establishing generalised arc consistency on constraints that impose this ordering. A first step is to consider lexicographic ordering between a pair of vectors, which is our current focus [10]. We can then consider enforcing generalised arc consistency on sets of such constraints. Furthermore, in Example 6 the choice of which permutation is to be the lexicographically smallest is arbitrary, but the performance of the variable-and-value-ordering depends on this choice. Work on this topic is in progress.

In other future work, we intend to find ways of detecting the row and column symmetries automatically. Also, given several matrices with symmetry and with channelling constraints in-between them, it is not clear how lexicographic orderings on the matrices interact. Finally, we will investigate ways of detecting redundancies among the super-exponential number of lexicographic ordering constraints that are necessary for breaking all the symmetries. For instance, in Example 6, the last three constraints are logically redundant.

Acknowledgements This work is partially supported by grant 221-99-369 of VR (the Swedish Research Council), by institutional grant IG2001-67 of STINT (the Swedish Foundation for International Cooperation in Research and Higher Education), and grant GR/N16129 of EPSRC (the UK Engineering and Physical Sciences Research Council). The last author was supported by an EPSRC advanced research fellowship. We thank Warwick Harvey and the members of the APES research group (apes.cs.strath.ac.uk), especially Barbara Smith, for their helpful discussions.

References

1. R. Backofen and S. Will, 'Excluding symmetries in constraint-based search', in *Proc. of CP'99, the 5th Int. Conf. on Principles and Practice of Constraint Programming*, ed., J. Jaffar, LNCS 1713, pp. 73–87. Springer-Verlag, (1999).
2. J. Crawford, 'A theoretical analysis of reasoning by symmetry in first-order logic', in *Proc. of the AAAI'92 workshop on tractable reasoning*, (1992).
3. J. Crawford, G. Luks, M. Ginsberg, and A. Roy, 'Symmetry breaking predicates for search problems', in *Proc. of KR'96, the 5th Int. Conf. on Knowledge Representation and Reasoning*, pp. 148–159, (1996).
4. T. Fahle, S. Schamberger, and M. Sellmann, 'Symmetry breaking', in *Proc. of CP'01, the 7th Int. Conf. on Principles and Practice of Constraint Programming*, ed., T. Walsh, LNCS 2239, pp. 93–107. Springer-Verlag, (2001).
5. P. Flener, A. Frisch, B. Hnich, Z. Kızıltan, I. Miguel, J. Pearson, and T. Walsh, 'Symmetry in matrix models', in *Proc. of SymCon'01, the CP'01 Workshop on Modelling and Problem Formulation*, (2001). Also available as technical report APES-36-2001 from http://www.dcs.st-and.ac.uk/~apes/reports/apes-36-2001.ps.gz. Available at http://www.csd.uu.se/~pierref/astra/SymCon01/.

6. P. Flener, A. Frisch, B. Hnich, Z. Kızıltan, I. Miguel, and T. Walsh, ‘Matrix modelling’, in *Proc. of Formul’01, the CP’01 Workshop on Modelling and Problem Formulation*, (2001). Also available as technical report APES-36-2001 from http://www.dcs.st-and.ac.uk/∼apes/reports/apes-36-2001.ps.gz. Followup paper also submitted to CP’02.
7. F. Focacci and M. Milano, ‘Global cut framework for removing symmetries’, in *Proc. of CP’01, the 7th Int. Conf. on Principles and Practice of Constraint Programming*, ed., T. Walsh, LNCS 2239, pp. 77–92. Springer-Verlag, (2001).
8. E. Freuder, ‘Eliminating Interchangeable Values in Constraint Satisfaction Problems’, om *Proc. of AAAI’91, Xth National Conf. on Artificial Intelligence*, pp. 227–233, (1991).
9. E. Freuder, ‘Modelling: The final frontier’, in *Proc. of PACLP’99, the 1st Int. Conf. on The Practical Application of Constraint Technologies and Logic Programming*, (1999).
10. A. Frisch, B. Hnich, Z. Kızıltan, I. Miguel, and T. Walsh, ‘Global constraints for lexicographic orderings’, also submitted to CP’02, (2002).
11. I.P. Gent, ‘A symmetry breaking constraint for indistinguishable values’, in *Proc. of SymCon’01, the CP’01 workshop on Symmetry in Constraints*, (2001). Available at http://www.csd.uu.se/∼pierref/astra/SymCon01/.
12. I.P. Gent and B.M. Smith, ‘Symmetry breaking in constraint programming’, in *Proc. of ECAI’00, the 14th European Conf. on AI*, ed., W. Horn, pp. 599–603, IOS Press, (2000).
13. Z. Kızıltan and B. Hnich, ‘Symmetry breaking in a rack configuration problem’, in *Proc. of IJCAI’01 Workshop on Modelling and Solving Problems with Constraints*, (2001).
14. P. Meseguer and C. Torras, ‘Exploiting symmetries within constraint satisfaction search’, *Artificial Intelligence*, **129**(1–2):133–163, (2001).
15. S.D. Prestwich, ‘Balanced incomplete block design as satisfiability’, in *Proc. of the 12th Irish Conf. on AI and Cognitive Science*, (2001).
16. J.-F. Puget, ‘On the satisfiability of symmetrical constrained satisfaction problems’, in *Proc. of ISMIS’93*, eds., J. Komorowski and Z.W. Ras, LNAI 689, pp. 350–361. Springer-Verlag, (1993).
17. B.M. Smith, ‘Reducing symmetry in a combinatorial design problem’, in *Proc. of CP-AI-OR’01, the 3rd Int. Workshop on Integration of AI and OR Techniques in CP*, (2001). Available as Research Report 2001.01, School of Computing, University of Leeds.
18. B.M. Smith, ‘Reducing symmetry in a combinatorial design problem’, in *Proc. of the IJCAI’01 workshop on Modelling and Solving Problems with Constraints*, pp. 105–112, (2001).
19. B.M. Smith, S.C. Brailsford, P.M. Hubbard, and H.P. Williams, ‘The progressive party problem: Integer linear programming and constraint programming compared’, *Constraints*, **1**:119–138, (1996).
20. B.M. Smith and I.P. Gent, ‘Reducing symmetry in matrix models: SBDS vs. constraints’, in *Proc. of SymCon’01, the CP’01 workshop on Symmetry in Constraints*, (2001). Available at http://www.csd.uu.se/∼pierref/astra/SymCon01/.
21. P. Van Hentenryck, L. Michel, L. Perron, and J.-C. Régin, ‘Constraint programming in OPL’, in *Proc. of PPDP’99, the Int. Conf. on Principles and Practice of Declarative Programming*, ed., G. Nadathur, LNCS 1703, pp. 97–116. Springer-Verlag, (1999).