Matrix Modelling

Pierre Flener
Uppsala University
Uppsala
Sweden

Alan M. Frisch
University of York
York
England

Brahim Hnich, Zeynep Kızıltan
Uppsala University
Uppsala
Sweden

Ian Miguel, Toby Walsh
University of York
York
England

Abstract

We argue that constraint programs with one or more matrices of decision variables provide numerous benefits, as they share many patterns for which general methods can be devised, such as for symmetry breaking. On a wide range of real-life application domains, we demonstrate the generality and utility of such matrix modelling.

1 Introduction

Many companies have scheduling, assignment, supply chain, and other problems that could be solved with a constraint programming (CP) toolkit. Although the solution of these problems is of vital commercial importance, CP toolkits are not widely used because there is insufficient expertise available to model problems as constraint programs. How then can we take the writing of efficient constraint programs out of research laboratories and specialised consultancies, and into everyday programming practice? How can we release the proven power of constraint programming to a wide user base, and thereby help improve industrial competitiveness?

As in other areas of software engineering, many *patterns* occur frequently in constraint programs. To date, these have only been given informally by practitioners, at the level of off-the-cuff remarks in reports. To tackle the modelling bottleneck, we need to identify, formalise, and document these patterns. We identify some patterns common to constraint programs taken from a wide range of domains. The basis for these patterns comes from identifying the central role played in many constraint programs by matrices of decisions variables, and the common types of constraints posted on them.

2 Matrix models

A *matrix model* has (one or more) matrices of decision variables. For example, a natural model of a sports scheduling problem has a 2-d matrix of decision variables, each of which is assigned a value corresponding to the game played in a given week and period [HMPR99]. In this case, the matrix is obvious in the solution to the problem: we need a *table* of fixtures. However, as we demonstrate in the following examples, many problems that are less obviously defined in terms of matrices can be efficiently represented and effectively solved using a matrix model. Sometimes, the matrix model contains multiple matrices of variables. Channelling constraints are then used to link the different matrices together [CCLW99; Smi00; Wal01].

There are a number of benefits of matrix models.

Symmetrical variables: We can call upon some standard methods to deal with rows and/or columns that are symmetrical. For example, we can use lexicographic ordering constraints to partially break row and column symmetry [FFH$^+$01]. Such techniques can also easily be extended to deal with partial symmetries.

Indistinguishable values: The same techniques for dealing with symmetrical variables can be applied to deal with indistinguishable values. A variable X that takes a single value or a set of values from a domain of n indistinguishable values can be replaced by a vector of n variables. each with the 0/1 domain. A value of 1 in position i of the vector indicates that value i is in the set of values assigned to X. The effect of this transformation is that the value symmetry in the domain of X has been replaced by symmetry among the variables in the vector. For example, in the social golfers problem (see Section 5), players, which are the values in the model, are indistinguishable. Converting this value symmetry into variable symmetry adds an extra dimension to the matrix model, which can be handled by our multi-dimensional symmetry method.

Variable indexing: We can give much more compact and efficient models when we can index into matrix models using variables. This is exploited in our matrix model of the progressive party problem (see Section 7).

Improved propagation: Matrix models can allow global constraints to be posted or linear models to be defined that can be passed to a simplex solver. For example, in the warehouse location problem (see Section 7), a 2-d matrix model gives an integer linear program for the cost function that is efficiently handled by a simplex solver.

Ease of statement: Many messy side constraints can be efficiently and effectively represented by channelling into a matrix model. For example, the steel mill slab design problem (see Section 6) is a bin packing problem with a side constraint restricting the number of “colours” of orders placed on any slab. This side constraint is easily implemented by channelling into a 2-d matrix model.

Matrix:

$$\begin{array}{cccccccccccc}
 & & & & \multicolumn{5}{c}{\longrightarrow \text{blocks} \longrightarrow} & & & \\
X_{ij} & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\
\downarrow & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\
\text{objects} & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
\downarrow & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\
 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0
\end{array}$$

Constraints:

(row sum) $\forall j \sum_i X_{ij} = r$
(column sum) $\forall i \sum_j X_{ij} = k$
(scalar product) $\forall j \neq k \ \sum_i X_{ij} * X_{ik} = \lambda$

Benefit:
partial row and column symmetry breaking

Figure 1: Matrix model for the BIBD generation problem

In the next five sections, we catalogue a wide range of problems that can be formulated using matrix models. In each case, we identify which of the benefits above applies. Despite the diversity of application domains, a number of common patterns are very apparent. For example, symmetry can often be broken using the same methods. The matrix models we give are not necessarily the best in each case. This is not the point. Rather, the focus is on the generality and utility of this modelling method.

3 Combinatorial problems

Many combinatorial problems can be naturally modelled as constraint satisfaction problems using matrix models. For example, **Balanced Incomplete Block Design** (BIBD) generation is a standard combinatorial problem from design theory. It has applications in experimental design and cryptography, and is prob028 in CSPLib (www.csplib.org).

As presented in Figure 1, 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 explained by its parameters $\langle v, b, r, k, \lambda \rangle$. One way of modelling a BIBD is in terms of its incidence matrix X, which is a v by 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.

This matrix model has row and column symmetry since we can permute any row or column freely without affecting any of the constraints. We can partially break this symmetry by lexicographically ordering both the rows and columns [FFH$^+$01].

The benefit of the matrix modelling for this problem is to be able to partially break a large number of symmetries. In a recent study on BIBD, a binary CSP model encoded in SAT is proposed to solve several BIBD instances using SATZ, WSAT, and CLS [Pre01]. All instances in [Pre01] could be solved much faster with our model. For instance, the instance $\langle 8, 14, 7, 4, 3 \rangle$ that was impossible to solve in a reasonable time with any algorithm or encoding could easily be solved with the matrix model in 238 seconds.

Many other combinatorial problems can be naturally modelled using matrix models; e.g., problems concerning block codes, quasigroup existence (prob003 in CSPLib), magic squares (prob019 in CSPLib), and projective planes (prob025 in CSPLib).

4 Configuration problems

Many configuration problems can be naturally formulated as matrix models. For example, the **Rack Configuration Problem** (discussed in [VH99]) is to plug cards into racks so that the total power demand and number of connectors required by the cards do not exceed those available for a rack of the chosen type. The maximum number of racks that can be deployed is n.

This can be represented with a matrix model. First, we use a 0/1 2-d matrix W indexed by the rack types and $1..n$, such that W_{ij} is 1 if there is a j^{th} occurrence of rack type i. Since at most n racks can be deployed and every rack must be of some rack type, each rack type may occur at most n times in a solution. In this model, we thus do not represent racks explicitly. We instead represent different occurrences of each rack type. Second, we use a 3-d 0/1 matrix R indexed by the rack types, $1..n$, and the cards, so that R_{ijk} is 1 if card k is plugged into the j^{th} occurrence of rack type i (see Figure 2).

Matrices:

$$\begin{array}{cccccc}
 & \multicolumn{5}{c}{\longrightarrow \text{rack types} \longrightarrow} \\
W_{ij} & 1 & 1 & 0 & 0 & 0 \\
\downarrow & 1 & 0 & 0 & 0 & 0 \\
1..n & 0 & 0 & 0 & 0 & 0 \\
\downarrow & 0 & 0 & 0 & 0 & 0
\end{array}$$

$$\begin{array}{cccccccc}
 & \text{cards} & & 0 & 1 & 0 & 0 & 0 \\
R_{ijk} & \nearrow & 0 & 0 & 0 & 0 & 0 & 0 \\
 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
\downarrow & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1..n & 0 & 0 & 0 & 0 & 0 & 0 & \\
\downarrow & 0 & 0 & 0 & 0 & 0 & & \\
 & \multicolumn{6}{c}{\longrightarrow \text{rack types} \longrightarrow} &
\end{array}$$

Constraints:

(matrix sum) $\sum_{ij} W_{ij} \leq n$
(slice sum) $\forall k \sum_{ij} R_{ijk} = 1$
(row sum) $\forall ij \sum_k R_{ijk} \leq Connector_i$
(weighted row sum) $\forall ij \sum_k R_{ijk} * Power_k \leq Power_i$
(channelling) $\forall ijk\ R_{ijk} = 1 \rightarrow W_{ij} = 1$

Benefits:
partial 3-d symmetry breaking
improved propagation
indistinguishable values

Figure 2: Matrix model for the rack configuration problem

The 2-d matrix W has symmetry since different occurrences of each rack type are indistinguishable. We can break this symmetry by ordering the different occurrences of each rack type. The 3-d matrix R also has symmetry since we can permute any occurrences of the same type, or any cards with the same number of connectors and power demand (i.e., with the same card type). We can break some of this symmetry by a lexicographic ordering of part of the matrix [FFH$^+$01].

The problem can be modelled in a different way by representing the racks explicitly. Since at most n racks are to be used, we create n variables, each representing a potential rack. A 1-d matrix indexed by $1..n$ thus gives the rack type for each potential rack. If such a rack is not used in the solution, then a dummy rack type is assigned to that rack. Additionally, we use a 2-d matrix indicating how many cards of a given card type are plugged into a rack. This model does not have the card symmetry that the model above has because card types instead of individual cards are represented in the 2-d matrix.

Since each card is to be plugged into exactly one rack, the assignment of racks to cards can also be modelled as a 1-d matrix indexed by cards ranging over racks. This 1-d matrix has partial column symmetry since cards of the same card type are indistinguishable. We can easily break this symmetry by a lexicographic ordering of the corresponding columns. On the other hand, values assigned to cards, namely racks, also are indistinguishable. In this model, we cannot easily break this symmetry. By introducing a 2-d 0/1 matrix, whose second dimension corresponds to the racks, we convert indistinguishable values into indistinguishable variables on which symmetry can be broken easily. Another advantage of this 2-d matrix model is the ease with which the problem constraints can be stated. The weighted sum constraint for each rack, which states that the total power of the cards assigned to a rack does not exceed its power capacity, in the 2-d matrix model corresponds to a weighted-occurrences constraint on the racks in the 1-d model, which would be inefficient to state in the absence of a weighted-occurrences global constraint.

However, better propagation is achieved in the 3-d model than in the 2-d model, resulting in a much more efficient problem formulation. For instance, an instance that was solved in 7.5 hours using the 2-d model could easily be solved using the 3-d model in 45 seconds [KH01].

5 Scheduling problems

The **Social Golfers Problem** is a class of problem instances parameterised by $\langle w, g, s\rangle$. The decision problem is to determine if it is possible for $g * s$ golfers to play in g groups, each of size s, in each of w weeks in such a way that any two golfers play in the same group at most once. Observe that this problem has three sets of indistinguishable objects: the set of golfers, the set of groups, and the set of weeks.

A straightforward way of modelling this problem has been employed by Stephano Novello in his Eclipse program (available at www.icparc.ic.ac.uk/eclipse/examples/golf.pl.txt). The model employs a 2-d matrix A of sets, where each row is a different week and each column is a different group. Each element A_{ij} of the matrix is a set of s golfers that play together as group i in week j. We call this Model A.

A solution to the $\langle 2, 3, 3\rangle$ instance of the Social Golfers Problem is shown in Figure 3. The figure also shows the general constraints for all instances of the problem class.

Matrix:

$$\begin{array}{cccc} A_{ij} & & \longrightarrow \text{groups} \longrightarrow & \\ \downarrow & \{1,2,3\} & \{4,5,6\} & \{7,8,9\} \\ \text{weeks} & \{1,4,7\} & \{2,5,8\} & \{3,6,9\} \\ \downarrow & & & \end{array}$$

Constraints:

(socialisation) $\forall (i,j) \neq (i',j')\ \ |A_{ij} \cap A_{i'j'}| \leq 1$

(group size) $\forall ij\ \ |A_{ij}| = s$

Benefit:

partial 3-d symmetry breaking

Figure 3: Matrix model for the social golfers problem

In this model there are three kinds of symmetry. The weeks (i.e., the rows of the matrix) are indistinguishable, the groups (i.e., the columns of the matrix) are indistinguishable, and the players (i.e., the elements of the value sets) are indistinguishable. Notice that if we employed a model in which each group were represented by a list (rather than a set) of s distinct players, then there would also be symmetry among the positions in this list.

Our model, called Model B, is a modification of Model A in which each set is replaced by a 0/1 vector of length $g * s$ representing the characteristic function of the set. For example, the set of golfers $\{1, 5, 9, 13\}$ is represented by the vector [1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0]. Thus Model B has a 3-d matrix, which we call B. As in Model A, the indices of the first dimension are indistinguishable, as are those of the second dimension. The indices of the third dimension of B, which represent the players, are also indistinguishable. Thus, the value symmetry in Model A that results from indistinguishable players has been replaced in Model B by variable symmetry in the third dimension. This is an advantageous replacement as we have effective ways to deal with variable symmetry. In particular, Model B partially breaks the symmetry in all three dimensions by imposing a triple-lexicographic ordering constraint.

Into Model A we have introduced some simple partial symmetry breaking. In particular, player symmetry is partially broken by fixing the assignments for the first week. In particular, for all $1 \leq j \leq g$ we set X_{1j} to the set of golfers $\{(j-1) * s + 1, (j-1) * s + 2, \ldots, j * s)\}$. Furthermore, we add the constraints that for every week except the first, the first group must contain golfer 1 and the second group must contain golfer 2. This partially breaks the symmetry caused by the indistinguishable groups.

A simple experiment shows that Model B eliminates significantly more symmetrical solutions than does Model A. On instance $\langle 3, 4, 4\rangle$, Model A finds a total of 1,327,104 solutions and takes 4,137 seconds to do so, whereas Model B finds only 576 solutions and takes 79 seconds to do so.

6 Design problems

Many design problems can be efficiently and effectively formulated as matrix models. For example, the **Steel Mill Slab Design Problem** is a difficult problem that reduces to variable-sized bin-packing with colour side-constraints [FMW01]. We are given a number of orders, each with a particular weight and colour, and a fixed number of slab sizes. We want to assign orders to slabs and sizes to slabs so that the total weight of orders assigned to a slab does not exceed the slab capacity, and so that each slab contains at most p colours (p is usually 2). Potentially redundant variables are used to cope with the fact that the number of slabs in an optimal solution is unknown. If we assume that the greatest order weight does not exceed the maximum slab size, the worst case assigns each order to an individual slab. Hence, we need as many slab variables as orders. As some slabs may remain unused, the zero element is added to the domain of each slab variable to represent when a slab is not used.

Matrices:

$$\begin{array}{c|ccccccccc} & \multicolumn{9}{c}{\longrightarrow \text{slabs} \longrightarrow} \\ s_i & 4 & 3 & 3 & 3 & 0 & 0 & 0 & 0 & 0 \end{array}$$

$$\begin{array}{c|ccccccccc} & \multicolumn{9}{c}{\longrightarrow \text{orders} \longrightarrow} \\ O_{ij} & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ \downarrow & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ \text{slabs} & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \downarrow & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

$$\begin{array}{c|ccccc} & \multicolumn{5}{c}{\longrightarrow \text{colours} \longrightarrow} \\ C_{ij} & 0 & 0 & 0 & 1 & 1 \\ \downarrow & 0 & 0 & 1 & 1 & 0 \\ \text{slabs} & 0 & 1 & 0 & 0 & 0 \\ \downarrow & 1 & 1 & 0 & 0 & 0 \end{array}$$

Constraints:

(row weighted sum)	$\forall j \sum_i Weight_i * O_{ij} \leq s_j$
(column sum)	$\forall i \sum_j O_{ij} = 1$
(row sum)	$\forall j \sum_i C_{ij} \leq p$
(channelling)	$\forall ij \; O_{ij} = 1 \rightarrow C_{Colour_i j} = 1$

Benefits:
partial row and column symmetry breaking
ease of statement
indistinguishable values

Figure 4: Matrix model for the steel mill slab design problem

A 2-d 0/1 matrix is used to represent which orders are assigned to which slabs (see Figure 4). This matrix has partial row symmetry since slabs of the same size are indistinguishable and partial column symmetry since orders of the same size and colour are also indistinguishable. As in the rack configuration problem, we can break this symmetry by a lexicographic ordering on the corresponding rows and columns. Breaking this symmetry is more difficult in a 1-d matrix that simply assigns orders to slabs. On a 16-order subset of real industrial data, a model employing lexicographic ordering finds an optimal solution in 94,000 fails and 18 seconds, whereas a model without lexicographic ordering requires nearly 600,000 fails and 50 seconds. In addition, a 1-d representation necessitates the use of specialised 'weighted occurrence' constraints, not commonly found in constraint toolkits, to express the slab capacity constraints.

A second 0/1 matrix is used to model the colour constraints. Channelling constraints are used to connect this to the order matrix. In this case, the benefit of matrix modelling is in the ease with which the colour constraints can be stated. Without this second matrix, we have large-arity constraints on the first matrix that can only be efficiently implemented by means of a complex daemon.

Another example of a design problem that can naturally be formulated as a matrix model is the **Template Design Problem** (prob002 in CSPLib). It arises from printing products (e.g., cartons for cat food) of same brand with several variations (e.g., flavours for cat food) from thin board. Such variations have different colour and/or text displayed on them, but are identical in shape and size so they can be printed on the same mother sheet of board. Each mother sheet is printed from a template that has s slots on each of which the design of the variations is imprinted. The problem is to decide how many pressings of each template are needed, and how many copies of which variation to include on each template such that the minimum number of pressings for each variation is met, every slot in each template is occupied by a variation, and the total number of templates being produced is minimised.

One way of tackling this problem is to fix the number of templates and then to minimise the total number of pressings. This can be modelled using a 1-d matrix Run giving the number of pressings of each template, and a 2-d matrix T specifying how many copies of which variation are included on which template (see Figure 5).

The 1-d matrix has column symmetry since all templates are indistinguishable. This symmetry can easily be broken by ordering the number of pressings. The 2-d matrix has partial row and column symmetry because variations with equal demands and templates with equal number of pressings are indistinguishable. We can break some of this symmetry by a lexicographic ordering on the corresponding rows and the columns.

The benefit of the matrix modelling for the template design problem is first the ease in which the problem constraints can be stated. Every template is related to more than one variation, and every variation can be related to more than one template. The matrix model thus provides a very natural way of representing the problem. Second, assume that every variation is to be assigned to exactly one template, in which case a 1-d matrix giving the template assigned to every variation would suffice. This model would suffer from value symmetry because templates are indistinguishable. However, some of this symmetry can easily be broken in the 2-d matrix formulation by translating value symmetry into variable symmetry.

The 2-d matrix model of the problem was previously studied in [PS98], though the symmetry between the templates with equal number of pressings was not tackled. We tested our model with the instances provided in [PS98], and found the same optimal solutions as the CP formulation in [PS98], up to the different scales we were working to. For instance, the cat food problem was solved with 2 templates with the objective value being $417,143$, and with 3 templates with the objective value being $407,778$ as opposed to the $418,000$ and $408,000$ found in [PS98].

Matrices:

$$\begin{array}{rccc} & & \longrightarrow \text{templates} \longrightarrow & \\ Run_i & 51,000 & 107,000 & 250,000 \end{array}$$

$$\begin{array}{rccc} & & \longrightarrow \text{templates} \longrightarrow & \\ T_{ij} & 0 & 0 & 1 \\ & 5 & 0 & 0 \\ \downarrow & 3 & 1 & 0 \\ \text{variations} & 0 & 0 & 2 \\ \downarrow & 0 & 0 & 2 \\ & 1 & 7 & 0 \\ & 0 & 1 & 4 \end{array}$$

Constraints:

(row weighted sum) $\forall j \sum_i Run_i * T_{ij} \geq Demand_j$
(column sum) $\forall i \sum_j T_{ij} = s$

Benefits:
partial row and column symmetry breaking
indistinguishable values
ease of statement

Figure 5: Matrix model for the template design problem

7 Assignment Problems

Many assignment problems can be formulated as matrix models. For example, the **Warehouse Location Problem** [VH99] arises when a company considers opening warehouses on some candidate locations in order to supply its existing stores. Each warehouse has a maintenance cost, and a capacity designating the maximum number of stores that it can supply. Each store must be supplied by exactly one open warehouse. The supply cost to a store depends on the warehouse. The objective is to determine which warehouses to open, and which of them should supply which stores such that the total cost is minimised.

This problem can be represented by a 1-d 0/1 matrix $Open$ giving the open warehouses, and a 2-d 0/1 matrix $Supply$ designating if a store is supplied by a warehouse (see Figure 6). One could replace this 2-d matrix with a 1-d one indexed by stores ranging over warehouses. Stating the problem constraints on the 2-d matrix, however, generates only linear constraints. This results in a pure ILP model that could efficiently be handled by powerful MIP solvers like CPLEX, using advanced techniques such as cutting planes and presolve reductions that speed up the proof of optimality. We experimented with the small instance provided in [VH99] and the CP model is as efficient as the ILP model. We expect that the ILP model will be much more efficient than the CP model on the instances where proving optimality is harder.

Matrices:

$$\begin{array}{rccccc} & & & \longrightarrow \text{warehouses} \longrightarrow & & \\ Open_i & 1 & 1 & 1 & 0 & 1 \end{array}$$

$$\begin{array}{rccccc} & & & \longrightarrow \text{warehouses} \longrightarrow & & \\ Supply_{ij} & 0 & 0 & 0 & 0 & 1 \\ & 0 & 1 & 0 & 0 & 0 \\ \downarrow & 1 & 0 & 0 & 0 & 0 \\ \text{stores} & 0 & 0 & 1 & 0 & 0 \\ \downarrow & 0 & 1 & 0 & 0 & 0 \\ & 1 & 0 & 0 & 0 & 0 \end{array}$$

Constraints:

(row sum) $\forall j \sum_i Supply_{ij} = 1$
(channelling) $\forall ij\ Supply_{ij} \leq Open_i$
(column sum) $\forall i \sum_j Supply_{ij} \leq Capacity_i$

Benefit:
improved propagation

Figure 6: Matrix model for the warehouse location problem

Another assignment problem is the **Progressive Party Problem** [SBHW96] that arises in the context of organising the social programme for a yachting rally (prob013 in CSPLib). There are a set of host boats, each with a capacity, and a set of guest boats, each with a crew size. The problem is to assign guests to hosts over a number of time periods, such that a guest crew never visits the same host twice and no two guest crews meet more than once.

A 2-d matrix is used to represent the assignment of guests to hosts in time periods (see Figure 7). All-different constraints on the rows of this matrix ensure that no guest ever revisits a host. A set of 1-d 0/1 matrices are used to represent when two guests meet on a host boat at a time period. An occurrence constraint on each 1-d matrix allows at most one such meeting. Finally, a 3-d 0/1 matrix is used. This matrix replicates the information (assignment of guests to hosts in periods) held in the 2-d matrix, but allows capacity constraints to be stated concisely via weighted sums of its columns. Furthermore, symmetry can be broken in all 3 dimensions of this matrix: periods are all symmetrical, as are host boats with equal capacity and guest boats with equal crew sizes. A 3-d lexicographic ordering [FFH+01] can be employed to remove much of this symmetry.

A simple means of channelling between the 2-d and 3-d matrices is to use a set of implication constraints of the form: $A_{ij} = k \leftrightarrow C_{ijk} = 1$. The disadvantage of this approach is that $i * j * k$ such constraints are necessary. An alternative approach makes use of variable indexing into the 3-d matrix, as presented in Figure 7. Just $i * j$ of these constraints are necessary along with $i * j$ row constraints to ensure that the remaining elements are set to 0 (also shown in the figure).

Matrix modelling provides two key advantages for the progressive party problem. Firstly, each of the matrices enables one of the problem constraints to be stated easily, whereas the same constraint would be much more difficult to state on the other matrix. The second advantage is again the translation from value symmetry (of the hosts in the 2-d matrix) to variable symmetry (in the 3-d matrix), which can be dealt with using lexicographic ordering. These advantages are underlined when we consider a problem with 7 periods, 7 hosts, and 13 guests: the model without 3-d symmetry-breaking solves this problem in 50,000 fails whereas the model with 3-d symmetry breaking makes just 2,000 fails.

Matrices:

$$\begin{array}{llll} & \multicolumn{3}{l}{\rightarrow \text{periods}} \\ Meet_i^j & 0 & 1 & 0 \end{array}$$

$$\begin{array}{rlll} & \multicolumn{3}{l}{\rightarrow \text{periods}} \\ Assign2d_{ij} & 0 & 1 & 2 \\ \downarrow & 2 & 1 & 0 \\ \text{guests} & 0 & 2 & 1 \\ \downarrow & 1 & 0 & 2 \end{array}$$

$$\begin{array}{rlllll} & \text{hosts} & 0 & 0 & 1 & \\ Assign3d_{ijk} & \nearrow & 0 & 1 & 0 & 0 \\ & 1 & 0 & 0 & 0 & 0 \\ \downarrow & 0 & 0 & 1 & 1 & 1 \\ \text{guests} & 1 & 0 & 0 & 0 & \\ \downarrow & 0 & 1 & 0 & & \\ & \multicolumn{5}{l}{\longrightarrow \text{periods} \longrightarrow} \end{array}$$

Constraints:

(row sum)	$\forall j \sum_i Meet_i^j \leq 1$
(row all-diff)	$\forall j$ all-different($Assign2d_{ij}$)
(col weighted sum)	$\forall ik \sum_j Size_j.Assign3d_{ijk}$ $\leq Capacity_k$
(row sum)	$\forall ij \sum_k Assign3d_{ijk} = 1$
(channelling)	$\forall ij\ Assign3d_{ijAssign2d_{ij}} = 1$
	$\forall ijk\ Assign2d_{ij} = Assign2d_{ik}$ $\leftrightarrow Meet_i^{jk} = 1$

Benefits:
partial 3-d symmetry breaking
ease of statement
indistinguishable values
variable indexing

Figure 7: Matrix model for the progressive party problem

8 Conclusions

Matrix modelling is more than just a way to view constraint programs containing arrays of variables. It suggests that constraint modelling languages should adopt matrices as first-class objects. Constraint toolkits should provide the same sort of matrix operations as found in a matrix manipulation language like MATLAB. For example, matrix symmetry breaking predicates and the scalar product operator should be primitives of the language and efficient propagators should be provided for reasoning about them.

As an alternative, the adoption of *sets* and *relations* as first-class objects in constraint modelling languages is advocated [F01; FHK01], together with operations such as relational image retrieval as well as relation composition and transposition. This is akin to ER-style conceptual modelling and OCL-style object modelling. Such *relational modelling* is higher-level than matrix modelling, and thus more ambitious, but the latter can be argued to be a perfect implementation technology for the former [FHK01]. It would ultimately be up to the modellers to choose their favourite level of abstraction.

In many of our examples, symmetry was an important feature. In addition to offering standard methods to deal with symmetry in matrix models, we should be able to identify such symmetries automatically. Whilst identifying symmetry is hard in general, it is much easier to identify row or column symmetry in a matrix (or relation) model.

Acknowledgements

This research is supported by EPSRC grant GR/N16129 and the Swedish Research Council (VR), under Research Grant number 221-99-369. The last author is supported by an EPSRC advanced research fellowship.

References

[CCLW99] B.M.W. Cheng, K.M.F. Choi, J.H.M. Lee, and J.C.K. Wu. Increasing constraint propagation by redundant modeling: An experience report. *Constraints*, 4:167–192, 1999.

[F01] P. Flener. Towards relational modelling of combinatorial optimisation problems. In *Proceedings of IJCAI-2001 Workshop on Modelling and Solving Problems with Constraints*. International Joint Conference on Artificial Intelligence, 2001.

[FHK01] P. Flener, B. Hnich, and Z. Kızıltan. Compiling high-level type constructors in constraint programming. In I.V. Ramakrishnan, editor, *Practical Applications of Declarative Languages*, pages 229–244. Springer-Verlag, 2001. Lecture Notes in Computer Science 1990.

[FFH+01] P. Flener, A.M. Frisch, B. Hnich, Z. Kızıltan, 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. In *Proceedings of the CP-01 Workshop on Symmetry in Constraints*. International Conference on the Principles and Practice of Constraint Programming, 2001.

[FMW01] A.M. Frisch, I. Miguel, and T. Walsh. Modelling a steel mill slab design problem. In *Proceedings of IJCAI-2001 Workshop on Modelling and Solving Problems with Constraints*. International Joint Conference on Artificial Intelligence, 2001.

[HMPR99] P. Van Hentenryck, L. Michel, L. Perron, and J.-C. Régin. Constraint programming in OPL. In G. Nadathur, editor, *Principles and Practice of Declarative Programming*, pages 97–116. Springer-Verlag, 1999. Lecture Notes in Computer Science 1702.

[KH01] Z. Kızıltan and B. Hnich. Symmetry breaking in a rack configuration problem. In *Proceedings of IJCAI-2001 Workshop on Modelling and Solving Problems with Constraints*. International Joint Conference on Artificial Intelligence, 2001.

[Pre01] S.D. Prestwich. Balanced incomplete block design as satisfiability. In *Proceedings of the 12th Irish Conference on Artificial Intelligence and Cognitive Science*, 2001.

[PS98] L. Proll and B.M. Smith. ILP and constraint programming approaches to a template design problem. In *INFORMS Journal of Computing*, 10:265–275, 1998.

[Smi00] B.M. Smith. Modelling a permutation problem. In *Proceedings of ECAI'2000 Workshop on Modelling and Solving Problems with Constraints*, 2000. Also available as Research Report from http://www.comp.leeds.ac.uk/bms/papers.html.

[SBHW96] B.M. Smith, S.C. Brailsford, P.M. Hubbard, and H.P. Williams. The progressive party problem: integer linear programming and constraint programming compared. In *Constraints*, 1:119–138, 1996.

[VH99] P. Van Hentenryck. *The OPL Optimization Programming Language*. The MIT Press, 1999.

[Wal01] T. Walsh. Permutation problems and channelling constraints. Technical Report APES-26-2001, APES group, 2001. Available from http://www.dcs.st-and.ac.uk/∼apes/reports/apes-26-2001.ps.gz.