Matrix Modelling:
Exploiting Common Patterns in Constraint Programming

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

[1] Department of Information Technology, Uppsala University, Uppsala, Sweden
[2] Department of Computer Science, University of York, York, England
[3] Department of Information Science, Uppsala University, Uppsala, Sweden
[4] Cork Constraint Computation Centre, University College Cork, Cork, Ireland

Abstract. Constraint programs with one or more matrices of decision variables are commonly and naturally used to model real-world problems. We call these matrix models and claim that they can be studied beneficially as a class. In support of this claim we present results in three areas: the systematization of formulating matrix models, the elimination of row and column symmetries from matrix models, and the efficient implementation of constraints for channelling between matrices.

1 Introduction

Though many companies have problems of vital commercial importance that could be solved with a constraint programming toolkit they do not do so because of a lack of expertise in modeling problems as constraint programs. We believe that we can bring the proven power of constraint programming to a wider user base, and thereby help improve industrial competitiveness, by systematising some of this expertise and embedding it in constraint toolkits.

A constraint programming expert is able to recognize patterns that commonly arise in problems and match them to patterns of problem formulation. The expert knows a range of patterns (or idioms) of solution techniques and knows how to match these to a variety of problem formulation patterns. To tackle the modelling bottleneck, we need to identify, formalise, and document these patterns of formulation and solution. We need to understand the properties of these patterns and to formulate heuristics on how to choose between alternative formulations and alternative solution techniques.

We envision a world in which formulation patterns are supported in much the same way that data structures are currently supported. Like data structures, formulation patterns would be published in journals and cataloged in textbooks along with analyses of their properties. Like textbooks on data structures, textbooks on formulation patterns would offer heuristic guidance on how to choose between alternative patterns. As with data structure implementations, libraries of implementations of formulation patterns would be available along with heuristic guidance on how to choose between alternative implementations. In much the

same way that high-level programming languages provide high-level data structures, high level constraint languages would provide high-level patterns. And as smart programming language compilers can choose what implementation to use in compiling a high-level data structure, constraint language compilers would do the same for high-level formulation patterns.

We have observed that one of the most common patterns, quite possibly *the* most common pattern, in constraint programs is the matrix (of one or more dimensions) of decision variables. We call any model of a constraint satisfaction problem (CSP) that employs one or more such matrices a *matrix model*. 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 [21]. In this case, the matrix is obvious in the statement of the problem: we need a *table* of fixtures. However, as we demonstrated elsewhere [7], many problems that are less obviously defined in terms of matrices can be effectively represented and efficiently solved using a matrix model.

There are also patterns that arise commonly *within* matrix models and this paper identifies and studies two. The first common pattern is the matrix in which some (or all) of the rows are interchangeable and some (or all) of the columns are interchangeable. In other words, a solution is still a solution if certain rows are interchanged and certain columns are interchanged. In many cases, problems are intractable unless these so-called *index symmetries* are reduced. The second common pattern in matrix models is *channelling*. This is the use of multiple matrices to encode information redundantly and channelling constraints to maintain consistency between the matrices [20, 3, 16, 22].

We claim that matrix models can be studied as a class, and that by doing so we can discover powerful generalities about how to use matrices of decision variables to formulate and solve CSPs. The discovery of these generalities can lead to a methodology for formulating and solving matrix models that is more systematic and (at least) partly automated. Most studies to date on the formulation of CSPs address the formulation of a particular problem. We believe that the success of this line of research has provided sufficient experience and enough examples of good formulations that we in the constraint community are now in a position where we can begin looking for generalizations.

After supporting, in Sec. 2, our claim of the prevalence of matrix models, this paper exemplifies the benefits of studying matrix models as a class by presenting results in three areas:

Modelling: Ultimately we desire systematic methods for formulating matrix models for any of a wide range of problems and for choosing between alternative models. We are a long way from achieving this ambitious goal. Sec. 3 lends support to the feasibility of this goal, and takes a small first step towards achieving it, by showing how alternative matrix models can be developed somewhat systematically from a high-level specification of a particular optimization problem.

Symmetry Breaking: Sec. 4 surveys our results on how index symmetry can be reduced by introducing extra constraints into a problem formulation.

Channelling Constraints: Sec. 5 identifies four common patterns of channelling constraints and discusses their efficient implementation.

2 The Prevalence of Matrix Models

In order to ascertain the prevalence of matrix models we have surveyed the 31 problems specified (in English) in CSPlib (www.csplib.org) on 18 April 2002. This has revealed that at least 27 of the 31 have natural matrix models, most of them already published and arguably the most natural models. Despite the small size of this sample, this seems to indicate that the corresponding problem class is significant and thus indeed deserves special study.

Most of the CSPlib problems are assignment problems, where some set V of "related" decision variables takes values within the set W of values, subject to some constraints. This is almost equivalent to the constraint satisfaction problem itself, except that the decision variables need to be related in some sense, including that they take their values within the same domain, so that they can be collected in an index set for a matrix. Assignment problems can be naturally modelled as matrix models, where a relation in $V \times W$ is encoded and sought, subject to the constraints. This point is elaborated in the next section where we see how a variety of matrix models can be formulated for a particular assignment problem.

The other CSPlib problems are permutation problems and set partitioning problems, which can be encoded as assignment problems so that they also admit matrix models, as well as planning problems, which do not admit (natural) matrix models, and problems with just one decision variable, which admit no (non-degenerate) matrix models.

Another indication of the prevalence of matrix models is the observation that all models in mixed integer programming and integer linear programming use a 2-dimensional matrix of decision variables.

3 Formulation with Matrix Models

This section demonstrates that matrix models can be derived somewhat systematically from a high-level problem specification. In particular, models that are readily executed with current constraint technology are derived from a specification that is not. The demonstration carefully considers the decisions involved in deriving alternative matrix models for the Balanced Academic Curriculum Problem (BACP), a problem proposed in [2] and further studied in [11]. Elsewhere [7] we have shown in much less detail how a variety of problems can be formulated with matrix models.

The BACP involves assigning a given set of courses to periods in which they will be taught in such a way to satisfy certain constraints. An instance of the problem consists of a finite set called $courses$; a finite set $\{1, \ldots, n\}$ called $periods$; a function, $credit$, that maps every course to its credit value; a binary relation, $prereq$, on the set of courses such that $\langle c_1, c_2 \rangle \in prereq$ indicates

that course c_1 is a prerequisite for course c_2; and four non-negative integers, $creditmin$, $creditmax$, $coursemin$ and $coursemax$. A solution to the problem instance maps every course to a period such that the following three constraints are met.

credit load constraint The credit load of any period, which is the sum of the credits of all courses assigned to the period, is no less than $creditmin$ and no greater than $creditmax$.

course load constraint The course load of any period, which is the total number of courses assigned to the period, is no less than $coursemin$ and no greater than $coursemax$.

prerequisite constraint If course c_1 is a prerequisite to course c_2 then c_1 is assigned to a period that is strictly less than that to which c_2 is assigned.

This is the BACP satisfaction problem. In the BACP optimization problem the goal is to find a solution that minimizes the maximum credit load for all periods.

3.1 A High-Level Specification of the BACP Using Functions

A high-level specification of the BACP is shown in Fig. 1. In addition to the input structures (i.e., those that encode the problem instance), the specification uses one more structure to capture the desired output. The output structure represents the desired curriculum and is a total function CUR from $courses$ into $periods$, which captures that every course will be assigned exactly one period.

With these input and output structures it is now straightforward to express the objective function and the three constraints of the problem. The credit load of any period p can be computed from the CUR and $credit$ functions: $\sum_{c \in CUR^{-1}(p)} credit(c)$, where $CUR^{-1}(p)$ denotes the set of all courses assigned to period p. So, we can express the objective as that of finding the function CUR that minimizes

$$\max_{p \in periods} \Big(\sum_{c \in CUR^{-1}(p)} credit(c) \Big) .$$

The credit load constraint is expressed as a set of inequalities. The number of courses of each period p is represented by the cardinality of the set of courses that have p as an image under CUR; by restricting this with inequalities we can specify the course load constraint. Finally, if course c_1 is a prerequisite of course c_2, then we enforce a strict ordering on their corresponding images.

3.2 Choice of Matrices

Starting from our high-level specification of the BACP, a space of matrix models can be generated when considering the decisions involved in modelling the function CUR and the constraints on it. In general, a desired total function F from a given set V into a given set W can be represented by different matrices of decision variables, among which are:

Inputs:	$courses : set(int)$ $periods : set(int)$ $creditmin, creditmax, coursemin, coursemax : int$ $credit : courses \longrightarrow int$ $prereq$: the powerset of $courses \times courses$
Outputs:	$CUR : courses \longrightarrow periods$
Minimize:	$\max_{p \in periods}(\sum_{c \in CUR^{-1}(p)} credit(c)\ )$
Credit load constraint:	$\forall p \in periods \cdot creditmin \leq \sum_{c \in CUR^{-1}(p)} credit(c) \leq creditmax$
Course load constraint:	$\forall p \in periods \cdot coursemin \leq \lvert CUR^{-1}(p)\rvert \leq coursemax$
Prerequisite constraint:	$\forall \langle c_1, c_2\rangle \in prereq \cdot CUR(c_1) < CUR(c_2)$

Fig. 1. A high-level specification of the BACP using functions

- $F1$: a 1d matrix $F1$ indexed by V and ranging over W. The interpretation of $F1[i] = j$ is that $F(i) = j$.
- $F2$: a 2d 0/1 matrix $F2$ indexed by V and W together with a constraint for each row restricting the sum of the Booleans to be 1. The interpretation of $F2[i, j] = 1$ is that $F(i) = j$.
- *F1+2*: $F1$ and $F2$ can be used simultaneously and linked through the channelling constraint $\bigwedge_{i \in V, j \in W} F1[i] = j \leftrightarrow F2[i, j] = 1$. Though this encoding uses more variables and introduces a channelling constraint, it might be advantageous when considering the ease of constraint formulation, as will be demonstrated later.

For the BACP it is difficult to compare these three implementations of CUR at this point, except to note that *CUR1+2* introduces redundant variables and a channelling constraint.

3.3 Constraint Formulations

We now analyze some possible formulations of each of the constraints and the objective function.

Credit load constraint. Let us consider how the credit load constraint can be expressed in each of our three matrix models.

Using the facilities of current constraint programming languages we know of no straightforward way to express the credit load for a given period using only the *CUR1* matrix. We therefore drop the matrix *CUR1* on its own as a possible model of the function CUR.

Using the matrix *CUR2*, the credit load of period p can be expressed as $\sum_{c \in courses} credit(c) * CUR2[c, p]$. The objective function also needs to refer to the credit load of each period. Experts recognise this pattern of repeated expressions and know that transforming a problem formulation by introducing new variables to stand in place of common expressions can often increase problem solving efficiency. (For example, this is crucial to the success of a formulation of Golumb ruler problem [19]). One reason for this increased efficiency is because the transformation can reduce the arity of the constraints, which can lead

to increased propagation during search. This variable-introducing transformation has been systematised and incorporated into CGRASS [10], a system that automatically transforms formulations of CSPs.

Performing this transformation, we introduce a new variable to represent the credit load of each period. Taken together, these new variables form a 1d matrix of variables, call it $LOAD$, that is indexed by periods and constrained by

$$\forall p \in periods \cdot LOAD[p] \ = \sum_{c \in courses} credit(c) * CUR2[c,p] \ . \tag{1}$$

The credit load constraint can now be imposed simply by making the domain of each variable in $LOAD$ to be the range $creditmin..creditmax$.

Objective function. The introduction of the $LOAD$ matrix also facilitates the statement of the objective:

$$minimize \quad \max_{p \in periods} (LOAD[p]) \ . \tag{2}$$

Alternatively it can be expressed as $minimize\ C$ where C is a variable constrained to be no smaller than all elements of $LOAD$:

$$\forall p \in periods \cdot LOAD[p] \leq C \ . \tag{3}$$

Course load constraint. In $CUR2$, each column p (denoted here by $CUR2_p$) contains a Boolean for each possible course c and when $CUR2[c,p] = 1$, we have that course c is assigned period p. So, to enforce the course load constraint, the number of 1's (courses) in each column (period) p has to be restricted to be between $coursemin$ and $coursemax$. There are two possible formulations. The first uses the global cardinality constraint gcc proposed by Régin [15]:

$$\forall p \in periods \cdot gcc(1, CUR2_p, [coursemin, coursemax]), \tag{4}$$

which restricts the number of occurrences of the value 1 to be between $coursemin$ and $coursemax$. The second uses linear constraints that restrict the occurrences of the value 1 in each column as follows:

$$\forall p \in periods \cdot coursemin \ \leq \sum_{c \in courses} CUR2[c,p] \ \leq \ coursemax \ . \tag{5}$$

Even though matrix $CUR1$ was not suitable for the period load computation, the course load constraint can be formulated with a global cardinality constraint by restricting the number of times a value p appears in $CUR1$ to be between $coursemin$ and $coursemax$:

$$\forall p \in periods \cdot gcc(p, CUR1, [coursemin, coursemax]) \ . \tag{6}$$

Prerequisite constraint. If course c_1 is a prerequisite of course c_2 we need to make sure that the period assigned to course c_1 is strictly less than that assigned to c_2. Using $CUR2$, to get the period assigned to a course c can use

the sum expression $\sum_{p \in periods} CUR2[c,p] * p$. Hence, a possible formulation of the prerequisite constraint can be achieved through the linear constraints:

$$\forall \langle c_1, c_2 \rangle \in prereq \cdot \sum_{p \in periods} CUR2[c_1,p] * p \; < \sum_{p \in periods} CUR2[c_2,p] * p \; . \quad (7)$$

In a model with *CUR1+2*, we can use *CUR1* to state the prerequisite constraint more compactly. The period assigned to a course c is simply $CUR1[c]$, so we can state the constraint as:

$$\forall \langle c_1, c_2 \rangle \in prereq \cdot CUR1[c_1] < CUR1[c_2] \; . \quad (8)$$

Since the constraints in (8) are of lower arity than those in (7), it might be more efficient to use (8) in a *CUR1+2* model.

3.4 Discussion of Matrix Models

Throughout this modelling exercise, we have seen that alternate matrix models of the BACP can be generated in a systematic manner by walking through the space of modeling decisions concerning the choice of matrices and the choice of constraint formulations on these matrices and by taking into consideration the solution methods to be employed. Though, as we have seen, some formulation choices can be made at the modeling time, others require experimentation.

In [11], some of the matrix models of the BACP have been compared experimentally on three real-life instances. Here we discuss the performance of the three most successful models, Models A, B and C, as defined in this table:

	Model A	Model B	Model C
Matrices	*CUR1+2*	*CUR1+2*	*CUR2*
Credit load constraint	(1)	(1)	(1)
Objective function	(3)	(3)	(3)
Course load constaint	(6)	(5)	(5)
Prerequisite constraint	(8)	(8)	(7)
Solver	CP	CP + ILP	ILP

Model A was executed with the constraint programming system OPL and Model C was executed with the integer linear programming system CPLEX. Model B was executed with a hybrid facility of OPL that performs a CP-style search but invokes CPLEX at each node of the search tree to produce a lower bound by using a linear relaxation on all linear constraints in the model. Notice that constraints (1), (2), (5), (7) and (8) are linear, but (3), (4) and (6) are not. Hence, other than the channelling constraint, all of the constraints used in Model B are linear.

Model A is the quickest to find optimal solutions on all instances, while Model B is the quickest to prove optimality on two instances and Model C is the quickest to prove optimality on one instance. Model A differs from Model B by using a global constraint for the course load constraint instead of a linear formulation and by using a CP solver instead of a hybrid one. Model A outperforms Model B in

finding optimal solutions because it performs more inference and thus eliminates more search space. Model B outperforms Model A in proving optimality because of the linear relaxation, which bounds and guides the search. Model B differs from Model C by channelling into *CUR1* and using constraints of lower arity to enforce the prerequisite constraint. It also employs a hybrid solver rather than an ILP one. Model B outperforms Model C in proving optimality on two instances because it improves the linear relaxation by reducing more of the search space due to improved propagation resulting from using constraints of lower arity and the use of a CP solver. However, due to the overhead caused by the increase in number of variables and channelling constraints, Model C is quicker in one instance.

4 Breaking Row and Column Symmetry

A common pattern in matrix models is that some (or all) of the rows of the matrix are interchangeable and some (or all) of the columns or interchangeable. In other words, a solution is still a solution if certain rows are interchanged and/or certain columns are interchanged. In many cases, problems are intractable unless these symmetries are reduced. We survey our results on how these symmetries can be eliminated by introducing extra constraints into the problem formulation. Full details, including a proof of Theorem 1, appear in [6].

Symmetry occurs because objects within the model are indistinguishable. For example, consider the tournament scheduling problem (problem 026 of CSPlib at www.csplib.og) in which a match must be scheduled in every period of every week. As the constraints of the problem do not distinguish among the weeks or among the periods, the members of each of these sets are interchangeable. A natural model of this problem is to have a 2-d matrix of variables, indexed by weeks and periods, in which each variable is assigned a match. Hence, this model has both row and column symmetry. As it is the values of the matrix's indices that exhibit the symmetry, we refer to this as *index symmetry*, a term that generalizes beyond two dimensions.

Many examples of index symmetry have been observed [7], such as in models for the balanced incomplete block design problem (prob028 of CSPlib), the steel mill slab design problem [7], the social golfers problem (prob010 of CSPlib), the template design problem (prob002 of CSPlib), the progressive party problem (prob013 at CSPlib), and the rack configuration problem (prob031 of CSPlib).

There are a number of ways of dealing with symmetry in constraint programming. The one that we adopt here is to add to the initial model constraints that break the symmetry [4, 14]. This method is best explained by thinking about *symmetry classes* (or *orbits* as they are called in group theory), which are obtained by partitioning the set of all assignments to decision variables such that the members of each partition are symmetric with each other. One can reduce or eliminate the symmetry in a model by adding extra constraints to the model that are satisfied by some, but not all, members of each symmetry class of the model. If exactly one member of each symmetry class satisfies the added con-

straints then all symmetry has been broken. As is usually done, the constraints that we add enforce an order on the symmetric objects. It is also possible to combine this approach to breaking index symmetry with another, as Gent and Smith [18] have done.

Let us begin by considering a 2-d matrix in which either all the rows are indistinguishable *or* all the columns are indistinguishable. To break all row (column) symmetry, we can order the rows (columns) lexicographically. The rows (columns) in a 2-d matrix are *lexicographically ordered* if each row (column) is lexicographically larger than the previous. As a lexicographical ordering is total, adding a lexicographical ordering constraint to the rows (columns) breaks all row (column) symmetry.

Now consider a 2-d matrix in which all the rows are indistinguishable *and* all the columns are indistinguishable. It is reasonable to conjecture that in this case all symmetry could be broken by imposing lexicographic ordering constraints on both rows and columns. We have proven that simultaneously meeting both lexicographic constraints is always possible.

Theorem 1 *Every symmetry class for a 2-dimensional matrix with row and column symmetry always has a member whose rows and columns are both lexicographically ordered.*

Though we can force lexicographic ordering on both rows and columns, unfortunately—and perhaps surprisingly—this does not break all symmetry. For example, observe that the following three members of a row/column symmetry class have lexicographically ordered rows and columns:

$$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix} \Leftarrow \begin{matrix} \text{swap rows 2 \& 3} \\ \text{swap columns 1 \& 2} \end{matrix} \Rightarrow \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \Leftarrow \begin{matrix} \text{swap rows 1 \& 2} \\ \text{swap columns 2 \& 3} \end{matrix} \Rightarrow \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix}$$

Though we do not have a practical way to break all row and column symmetries in all cases,[5] we have identified three conditions, which arise naturally and commonly, such that for each we can break all row and column symmetries by introducing a linear number of easily-imposed constraints, such as lexicographic ordering of rows and columns. In cases where lexicographic ordering does not break all row and column symmetry we have given experimental results showing that sometimes it does break enough symmetries to make an intractable instance tractable.

More dimensions: Though this section has only considered index symmetry on 2-d matrices, we have shown that these results extend to matrices in any number of dimensions in which there is symmetry on any subset of the indices.

Partial symmetry: Some matrix models have index symmetry that is only partial in that only some subsets of an index's values are interchangeable. For example, in one model of the rack design problem only those columns that correspond to racks of the same type are interchangeable with each other. To handle

[5] Following the approach of Crawford et. al. [4], we have shown how all row and column symmetries in an n by m matrix can be broken by imposing $n! \cdot m! - 1$ lexicographic ordering constraints, but clearly this is not practical.

such a situation one can add constraints that enforce a lexicographic ordering on each subset of rows (or columns) that are interchangeable.

Value symmetry: Many CSP models have value symmetry in that some variables have indistinguishable domain elements. For example, the social golfers problem requires that a set of golfers be partitioned in a particular way. The golfers, which in many models of the problem are the domains of the variables, are indistinguishable. We have shown how a model with value symmetry can be transformed into a model that has index symmetry instead of value symmetry. Our techniques for breaking variable symmetry can then be used and our experimental results show that this can be a highly-effective method for handling value symmetry.

Enforcing lexicographic ordering: The utility of reducing index symmetry by introducing lexicographic ordering constraints depends upon having a method to enforce these constraints efficiently during search. Elsewhere [9] we have introduced the first linear time algorithm for incrementally enforcing generalized arc consistency on a lexicographic ordering constraint. We have shown that some problem instances can be solved many times faster by using this algorithm over its competitors.

In summary, index symmetry and value symmetry are common patterns in matrix models and the ability to solve many problems requires greatly reducing the amount of symmetry. We have shown that imposing a lexicographic ordering on each dimension of a multi-dimensional matrix can always be used to reduce index symmetry and—since we have shown that value symmetry can be mapped to index symmetry—value symmetry as well. In some special cases we have identified methods for breaking all index symmetry. In our future work, we intend to look at ways of identifying row and column symmetry automatically, and at methods for reducing symmetry even more efficiently and completely.

5 Channelling Constraints

Another common pattern that arises within matrix models is the use of multiple matrices that represent the same information redundantly and *channelling constraints* that enforce consistency among the representations [3]. Despite the increase in the number of variables and constraints, there are a number of benefits of channelling, which are identified in this section.

5.1 Types of Channelling Constraints

There are a variety of types (or patterns) of channelling constraints, each characterized by a schema. The schemas differ in the type of matrices involved and the constraints imposed on them. This section presents four of these schemas and explains why each is useful.

The first type of channelling constraint is very useful in permutation problems, a common and well-studied problem pattern that arises in many assignment, scheduling, and sequencing problems. In a permutation problem a set of

n variables must be assigned distinct values from a domain of size n. Such a problem can be modelled by a matrix X of n variables, taking distinct values from $\{1, \ldots, n\}$. For instance, in the n-queens problem each $X[i]$ represents the queen on row i, and the value assigned represents the column on which that queen is placed. Since the queens are to be on different columns, and there are in total n columns, the queens are to be assigned distinct values from $1 \ldots n$. In a permutation problem, every variable is assigned a value and every value in the domain of the variables is assigned a variable. Hence, as observed in [16], one can enforce the distinctness constraint by introducing a dual matrix Y of n variables taking values from $\{1, \ldots, n\}$, and asserting the channelling constraint

$$\bigwedge_{v,w \in \{1,\ldots,n\}} X[v] = w \leftrightarrow Y[w] = v \ . \tag{9}$$

Matrices X and Y are duals of each other in the sense that the roles of the variables and the values are interchanged. Even though arc-consistency on the channelling constraint is not as tight as arc-consistency on an *alldifferent* constraint on the elements of X [22], in practice channelling constraints may significantly reduce the run-time with a slight increase in the number of backtracks [17]. Furthermore, the use of dual matrices opens the possibility of using both sets of variables as search variables, which can sometimes reduce search [17].

A second type of channelling that is often used is 1-d to 1-d channelling of the form $X[v] = w \rightarrow Y[w] = 1$. For instance, the channelling in a model of the warehouse location problem is of this form [20]. The assignment of stores to warehouses is modelled as a 1-d matrix X, indexed by the set of stores V, taking values from the set of warehouses W. The set of warehouses to be opened is modelled as a 1-d 0/1 matrix Y indexed by the set of warehouses W. If a warehouse is supplying a store then that warehouse must be open. Thus, we channel between the matrices as follows:

$$\bigwedge_{v \in V, w \in W} X[v] = w \rightarrow Y[w] = 1 \ .$$

The main motivation behind this type of channelling is to facilitate the statement of the problem constraints based on the objects that are assigned (e.g., stores in the warehouse problem), as well as the constraints based on the values that are assigned to any of objects (e.g., warehouses being opened in the warehouse problem).

A third type of channelling, which is similar to the second type, arises when an assignment problem is modelled by a 2-d 0/1 matrix with an additional constraint that every object is assigned exactly one value, and the values being assigned are also represented. For instance, the assignment of stores to warehouses can also be modelled by a 2-d 0/1 matrix X indexed by the set of stores V and the set of warehouses W. The channelling constraint in this case is

$$\bigwedge_{v \in V, w \in W} X[v, w] = 1 \rightarrow Y[w] = 1 \ .$$

A fourth type of channelling links a 1-d matrix and a 2-d with channelling of the form $X[i] = j \leftrightarrow Y[i, j] = 1$. For instance, the channelling in the *CUR1+2* model of BACP is of this form. In one model, the assignment of courses to periods is modelled as a 1-d matrix X, indexed by the set of courses V, taking values from the set of periods W. In the other model, the assignment is modelled as a 2-d 0/1 matrix Y, indexed by the set of courses V and the set of periods W. The combined model is obtained by channelling the models as follows:

$$\bigwedge_{v \in V, w \in W} X[v] = w \leftrightarrow Y[v, w] = 1 \ .$$

This fourth type of channelling also is used to turn value symmetry into variable symmetry. If a 1-d matrix of variables take values from a set of indistinguishable values, we can convert value symmetry into variable symmetry by channelling into a 2-d matrix of 0/1 variables whose first dimension is the same as the original matrix and second dimension corresponds to the set of values. This allows us to employ the techniques for breaking variable-symmetry to tackle the value symmetry.

As seen, in many cases channelling is useful, and thus is frequently used. Consequently, the efficient implementation of channelling constraints is important and so we now turn our attention to how we can efficiently maintain generalized arc-consistency on channelling constraints.

5.2 Generalized Arc Consistency on Channelling Constraints

Each channelling constraint is a conjunction where each conjunct is of the form $E \rightarrow E'$ or each is of the form $E \leftrightarrow E'$. Each of E and E' is an equation of the form $Var = c$, where Var is an indexed variable (that is, a variable in a matrix) and c is a constant.

The insight in how to enforce generalized arc-consistency (GAC) on any channelling constraint of the above forms comes from two observations:

Proposition 1. *No two conjuncts of a channelling constraint have variables in common.*

Proposition 2. *Let $\{C_1, \ldots, C_n\}$ be a set of constraints such that no two have a variable in common and let $\mathcal{C}$ be the constraint $C_1 \wedge \cdots \wedge C_n$. Constraint $\mathcal{C}$ is GAC if and only if either*

- *every variable occurring in $\mathcal{C}$ has an empty domain, or*
- *no variable occurring in $\mathcal{C}$ has an empty domain and each member of $C_1, \ldots, C_n$ is GAC.*

By these observations, the task of maintaining GAC on a channelling constraint of the form $E \rightarrow E'$ (resp. $E \leftrightarrow E'$) can be decomposed into a finite set of tasks of maintaining GAC on an implication (resp. bi-implication) constraint. We shall not discuss the algorithms for maintaining GAC on an implication or

bi-implication constraint as they are straightforward and are provided by most constraint toolkits.

Thus, using existing constraint toolkit facilities it is easy to implement a channelling constraint by explicitly posting an implication or bi-implication constraint for every conjunct in the constraint. However, the space and runtime required by this implementation will be proportional to the number of conjuncts. Notice that the number of conjuncts can be quite large. For example, for a permutation involving n elements, the channelling constraint (9) has n^2 conjuncts.

We have implemented a generic channelling constraint whose input is a schema of the form $E \rightarrow E'$ or $E \leftrightarrow E'$. Rather than instantiate the schema to generate all the conjuncts, it uses a generic demon that performs the appropriate action whenever the domain of a variable among the (implicitly represented) conjuncts is modified. The time and space required by this schematic implementation is independent of the number of conjuncts in the channelling constraint.

This schematic implementation dramatically improves the runtime of the channelling constraint, which is reflected in the efficiency of models that use channelling. As an example, consider Langford's problem (prob024 in CSPlib), parameterised by (k, n). This permutation problem is to compose a sequence containing exactly k occurrences of each integer from 1 to n so that each occurrence of the number m is m positions from the last. An efficient model represents the permutations with primal and dual matrices and channelling of the form of (9). On the instance $(4, 16)$ a naive implementation, which imposes 4096 bi-implications, takes 635.6 seconds to prove that there is no solution. In contrast, an efficient schematic channelling constraint takes 32.9 seconds. On the instance (4, 17) the results are 2259.6 and 107.9 seconds respectively.

6 Conclusions

We have identified the matrix of decision variables as a common pattern in constraint programming and one, we claim, whose study would reveal powerful generalizations. In support of this claim we have shown how alternative matrix models of the BACP can be derived in a somewhat systematic manner and how two common patterns within matrix models—index symmetry and channelling constraints—can be handled. Stepping back, let us now see how these are first steps towards a world in which the formulation bottleneck is reduced or eliminated and also see what other steps are necessary.

From the somewhat systematic derivation of matrix models for the BACP one can see that formulations for similar problems could be derived in a similar manner. By examining a range of such derivations one could formulate general, more systematic, patterns of formulation. Ultimately we would like to automate this process in the form of a compiler that translates a high-level specification into structures—such as matrices of decision variables—that are supported by current constraint programming toolkits. The field is beginning to make progress in this direction through the development of systems such as CGRASS [10], ESRA [5, 8] and NP-SPEC [1].

A closely-related problem is the design of an appropriate high-level constraint language. As suggested by our high-level specification of the BACP, we envision a language that supports sets, functions and relations as well as related concepts such as partitions, projections, subsets, and relational composition and transposition. ESRA [5, 8], and OPL [20] can be seen as providing a step in this direction.

We have identified index symmetry as a pattern that is common in matrix models and have developed methods of reducing it. Though these methods have proved to be effective in a range of problems, we suspect that more complete methods will be required to effectively solve certain other problems. More ambitiously, we envision a system that can identify symmetries automatically and choose appropriate methods of reducing them. Whilst identifying symmetry is computationally expensive in general, it is much easier to identify index symmetry. Finally, an ideal reformulation system would also be able to choose variable and value orderings for search that do not adversely interact with the symmetry-breaking constraints it introduces.

We have identified channelling as a constraint pattern that arises frequently in matrix models. This can be taken much further by addressing these additional questions: What other constraint patterns arise frequently in matrix models? Computationally, what is the best way to handle these constraints? Linguistically, what is the best way to package these patterns in a constraint modelling language?

Our long-term aim is to advance the capability of modelling tools to the point where programming with today's tools looks like programming in assembler language.

Acknowledgements

This research is supported by EPSRC grant GR/N16129 and the Swedish Research Council (VR), under Research Grant number 221-99-369. We are grateful for the valuable discussions we have had with Justin Pearson, Warwick Harvey, Barbara Smith and Ian Gent.

References

1. M. Cadoli and A. Schaerf. Compiling problem specifications into SAT. In *Proc. of ESOP'01*, Springer Verlag, LNCS 2028, April 2001.
2. C. Castro and S. Manzano. Variable and value ordering when solving balanced academic curriculum problem. In *Proc. of the ERCIM Working Group on Constraints*, Prague, June 2001.
3. 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.
4. J. Crawford, G. Luks, M. Ginsberg, and A. Roy. Symmetry breaking predicates for search problems, in *Proc. of the 5th Int. Conf. on Knowledge Representation and Reasoning, (KR '96)*, pp. 148–159, (1996).

5. 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.
6. P. Flener, A.M. Frisch, B. Hnich, Z. Kızıltan, I. Miguel, J. Pearson, and T. Walsh. Symmetry in matrix models. In *Principles and Parctice of Constraint Programming—CP 2002*. Springer Verlag, LNCS 2470, 2002.
7. P. Flener, A.M. Frisch, B. Hnich, Z. Kızıltan, I. Miguel, and T. Walsh. Matrix modelling. In *Proc. of the CP-01 Workshop on Modelling and Problem Formulation*, 2001.
8. 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, LNCS 1990, 2001.
9. A.M. Frisch, B. Hnich, Z. Kızıltan, I. Miguel, and T. Walsh. Global constraints for lexicographic orderings. In *Principles and Parctice of Constraint Programming—CP 2002*. Springer Verlag, LNCS 2470, 2002.
10. A.M. Frisch, I. Miguel, and T. Walsh. CGRASS: A system for transforming constraint satisfaction problems. In *Proc. of the ERCIM/Colognet Workshop on Constraint Solving and Constraint Logic Programming*, pp. 23-36, 2002.
11. B. Hnich, Z. Kızıltan, and T. Walsh. Modelling a balanced academic curriculum problem. *Proc. of CP-AI-OR02*, 2002.
12. 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.
13. S.D. Prestwich. Balanced incomplete block design as satisfiability. In *Proceedings of the 12th Irish Conference on Artificial Intelligence and Cognitive Science*, 2001.
14. J.-F. Puget. On the satisfiability of symmetrical constrained satisfaction problems. In *Int. Syposium on Methodologies for Intelligent Systems (ISMIS 1993)*, pp. 350–361, Springer Verlag, LNCS 689, 1983.
15. J.C. Régin. Generalized Arc consistency for global cardinality constraints, in *Proc. of AAAI'96*, AAAI Press/The MIT Press, pages 209–215, 1996.
16. B.M. Smith. Modelling a permutation problem. In *Proceedings of ECAI'2000 Workshop on Modelling and Solving Problems with Constraints*, 2000.
17. B.M. Smith. Dual models of permutation problems. In *Principles and Practice of Constraint Programming - CP 2001*, pp. 615-619, Springer-Verlag, LNCS 2239, 2001.
18. B.M. Smith and I.P. Gent. Reducing symmetry in matrix models: SBDS v. constraints, in *Proc. of the CP-2001 Workshop on Symmetry in Constraints*, (2001).
19. B.M. Smith and K. Stergiou and T. Walsh. Modelling the Golomb ruler problem, In *Proc. of the IJCAI-99 Workshop on Non-Binary Constraints*, 1999.
20. P. Van Hentenryck. *The OPL Optimization Programming Language*. The MIT Press, 1999.
21. 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, LNCS 1702, 1999.
22. T. Walsh. Permutation problems and channelling constraints. In *Proceedings of the 8th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2001)*, Springer Verlag, LNCS 2250, 2001.