The Cardinality Matrix Constraint *

Jean-Charles Régin[1] and Carla Gomes[2]
regin@ilog.fr, gomes@cs.cornell.edu

[1]ILOG, 1681, route des Dolines, 06560 Valbonne, FRANCE
[2]Computing and Information Science, Cornell University, Ithaca NY 14850 USA

Abstract. Cardinality matrix problems are the underlying structure of several real world problems such as rostering, sports scheduling , and timetabling. These are hard computational problems given their inherent combinatorial structure. Constraint based approaches have been shown to outperform other approaches for solving these problems. In this paper we propose the cardinality matrix constraint, a specialized global constraint for cardinality matrix problems. The cardinality matrix constraint takes advantage of the intrinsic structure of the cardinality matrix problems. It uses a global cardinality constraint per row and per column and one cardinality (0,1)-matrix constraint per symbol. This latter constraint corresponds to solving a special case of a network flow problem, the transportation problem, which effectively captures the interactions between rows, columns, and symbols of cardinality matrix problems. Our results show that the cardinality matrix constraint outperforms standard constraint based formulations of cardinality matrix problems.

1 Introduction

In recent years Constraint Programming (CP) techniques have been shown to effectively solve hard combinatorial problems. In fact, constraint based methods excel at solving problems that are inherently combinatorial, clearly outperforming traditional Operations Research (OR) techniques. Sports scheduling and rostering problems are good examples of highly combinatorial problems, for which CP based techniques have been shown to be very successful (see e.g., [11, 5]).

In a rostering problem, given a set of workers, a set of tasks, and a set of days (typically a week), the goal is to assign the tasks per person and per day satisfying various constraints. Among them, typical constraints require the workload per day to be constrained by the number of times each task has to be performed, and the schedule of each person to be constrained by the number of times each task has to be performed.

Sports scheduling problems and rostering problems are particular cases of what we refer to as *cardinality matrix problems*. Cardinality matrix problems are expressed by a matrix of variables where each row and each column are constrained by cardinality constraints, that is by constraints that define the

* Supported by the Intelligent Information Systems Institute, Cornell University (AFOSR grant F49620-01-1-0076) and EOARD grant FA8655-03-1-3022.

number of times each value in a row or in a column has to be assigned to variables. We can model the rostering problem as a cardinality matrix problem in which each row of the matrix corresponds to a worker and each column of the matrix corresponds to a day of the week. The values in the matrix correspond to tasks. The cardinality constraints on the rows constrain the number of tasks to be assigned per worker and the cardinality constraints on the columns constrain the number of task to be assigned daily.

A straightforward model for representing cardinality matrix problems, e.g., rostering problems, consists of:

• a matrix of variables, in which each variable corresponds to a cell that takes as value the task to be performed on a given day by a given person. The variable domains are the set of tasks that can be performed by a given person for a given day.

• a global cardinality constraint (GCC) for every row, which constrains the number of times each task has to be performed by the person corresponding to the row.

• a global cardinality constraint for every column, which constrains the number of times each task has to be performed for the day corresponding to the column.

This formulation uses several global constraints and can give good results in practice. However, it suffers from some major drawbacks, namely:

1. There is poor communication between the variables constraining the number of times a value has to be taken, called cardinality variables.
2. The communication between the rows and the columns is poor. In fact, any GCC defined on a row and any GCC defined on a column have only one variable in common. This means that we have an efficient way to deal with all the variables of a row (or a column) as a whole, but we are not able to really deal with the notion of a matrix.
3. The GCCs deal with a set of predefined intervals constraining for each value the number of times the value has to be assigned. In real-life problems, variables defining these intervals are more often used. Even if it is easy to deduce intervals from these variables, because it corresponds to the boundaries of these variables, we do not have filtering algorithms to reduce their ranges in the general case (such a filtering algorithm has been proposed when the domains of the variables on which the GCCs are defined are ranges [9]).

The communication between constraints mentioned in (1) and (2) can be improved by adding implied constraints. An implied constraint for a given CSP is a constraint that can be deduced from the other constraints of the CSP, but which introduces a filtering algorithm that can reveal inconsistencies which are not discovered by the combination of the filtering algorithms of the other constraints. So the introduction of implied constraints can lead to a reduction of the number of backtracks needed to find one solution or to prove that there is none. The introduction of implied constraints can improve dramatically the efficiency of search since it allows for the detection of inconsistencies earlier than it would be possible if such constraints were not stated explicitly (see e.g., [3]).

The limitation stated in point (2) deserves a more careful study. Consider a restricted form of the cardinality matrix problems: the alldiff matrix problem[8]. In this case, each value has to be assigned at most once in each row and each column. The alldiff matrix characterizes the structure of several real world problems, such as design of scientific experiments or fiber optics routing. Consider the following example: a 6x6 matrix has to be filled with numbers ranging from 1 to 6 (this is a latin square problem). A classical model in CP consists of defining one variable per cell, each variable can take a value from 1 to 6, and one alldiff constraint per row and one alldiff constraint per column. Now, consider the following situation:

		1	2		
		2	1		
		3	4		
		4	5		
•	•			•	•
•	•			•	•

In this case, the alldiff constraints are only able to deduce that:

- only the values 5 and 6 can be assigned to the cells $(5,3)$ and $(6,3)$
- only the values 3 and 6 can be assigned to the cells $(5,4)$ and $(6,4)$.

However, with a careful study we can see that the value 6 will be assigned either to $(5,3)$ and $(6,4)$ or to $(5,4)$ and $(6,3)$ this means that the other columns of rows 5 and 6 cannot take these values and therefore we can remove the value 6 from the domains of the corresponding variables (the ones with a • in the figure). We will show how our approach, using what we refer to as the cardinality (0,1)-matrix, automatically performs these inferences.

One of the key successful approaches in CP has been the identification of typical constraints that arise in several real-world problems and associate with them very specialized and efficient filtering algorithms, so-called *global constraints*. In recent years several global constraints have been proposed and shown to boost dramatically the performance of CP based techniques.

We propose the *cardinality matrix constraint* to capture the structure of cardinality matrix problems such as the rostering problem. A cardinality matrix constraint (cardMatrix) C is specified in terms of an $n \times m$ matrix M of variables which take their values from a set of s symbols, and two sets ($rowCard$ and $colCard$) of cardinality variables that specify the number of times each symbol has to appear in a row ($rowCard$) and the number of times each symbol has to appear in a column ($colCard$). More specifically, the set of cardinality variables $rowCard$ constrains the number of variables of a row i of M instantiated to a symbol p to be equal to $rowCard[i,p]$ and the set of cardinality variables $colCard$ constrains the number of variables of a column j of M instantiated to a symbol q to be equal to $colCard[j,q]$. In order to take advantage of the structure underlying the *cardinality matrix constraint* we introduce a constraint named cardinality (0,1)-matrix. The cardinality (0,1)-matrix is a particular case of a network flow problem, the transportation problem. This constraint effectively captures the interactions between rows, columns, and symbols in a cardinality matrix problem. We also develop a simple filtering algorithm for the cardinality matrix constraint with a low complexity that enables us to reduce the ranges of the cardinality variables. As we show in our experimental section, we obtain very promising results which allow us to solve problems that could not be solved before with constraint programming techniques. We also compare the performance of our approach against standard formulations of a cardinality matrix problems. We obtain dramatic speed ups with our approach.

The rest of the paper is organized as follows: In the next section we define our notation and present definitions concerning constraint programming and graph theory. We then roughly present the cardinality matrix constraint and propose a simple filtering algorithm for reducing the ranges of cardinality variables of a GCC. Next, we introduce the Cardinality (0,1)-Matrix Constraint followed by the description of a filtering algorithm for the Cardinality Matrix Constraint. We present experimental results in section 7, followed by conclusions.

2 Preliminaries

$\mathcal{D}_0 = \{D_0(x_1), \ldots, D_0(x_n)\}$ to represent the set of initial domains of $\mathcal{N}$. Indeed, we consider that any constraint network $\mathcal{N}$ can be associated with an initial domain $\mathcal{D}_0$ (containing $\mathcal{D}$), on which constraint definitions were stated.
A **constraint** C on the ordered set of variables $X(C) = (x_{i_1}, \ldots, x_{i_r})$ is a subset $T(C)$ of the Cartesian product $D_0(x_{i_1}) \times \cdots \times D_0(x_{i_r})$ that specifies the **allowed** combinations of values for the variables $x_1, \ldots, x_r$. An element of $D_0(x_1) \times \cdots \times D_0(x_r)$ is called a **tuple on** $X(C)$. $\tau[x]$ denotes the value of x in the tuple τ.
Let C be a constraint. A tuple τ on $X(C)$ is **valid** if $\forall x \in X(C), \tau[x] \in D(x)$. C is **consistent** iff there exists a tuple τ of $T(C)$ which is valid. A value $a \in D(x)$ is **consistent with** C iff $x \notin X(C)$ or there exists a valid tuple τ of $T(C)$ with $a = \tau[x]$. A constraint is **arc consistent** iff $\forall x_i \in X(C), D(x_i) \neq \emptyset$ and $\forall a \in D(x_i)$, a is consistent with C.

The **value graph** of a set of variables X is the bipartite graph $GV(X) = (X, \cup_{x_i \in X} D(x_i), E)$ where $(x, a) \in E$ iff $a \in D(x)$.

We recall the formal definition of a global cardinality constraint:

Definition 1 *A* **global cardinality constraint** C *defined on* X *and associated with a set of values* V *with* $D(X) \subseteq V$ *is a constraint in which each value* $a_i \in V$ *is associated with two positive integers* l_i *and* u_i *with* $l_i \leq u_i$ *and*
$T(C) = \{\ \tau$ *s.t.* τ *is a tuple on* $X(C)$
and $\forall a_i \in V : l_i \leq \#(a_i, \tau) \leq u_i\}$
It is denoted by $gcc(X, V, l, u)$.

Note that an alldiff constraint can be defined by a GCC in which all lower bound are equals to 0 and all upper bounds are equal to 1.

An instantiation of all variables that satisfies all the constraints is called a solution of a CN. Constraint Programming (CP) proposes to search for a solution by associating with each constraint a filtering algorithm that removes some values of variables that cannot belong to any solution. These filtering algorithms are repeatedly called until no new deduction can be made. Then, CP uses a search procedure (like a backtracking algorithm) where filtering algorithms are systematically applied when the domain of a variable is modified.

2.1 Graph Theory

These definitions are based on books of [2, 16, 1].

A **directed graph** or **digraph** $G = (X, U)$ consists of a **node set** X and an **arc set** U, where every arc (u, v) is an ordered pair of distinct nodes. We will denote by $X(G)$ the node set of G and by $U(G)$ the arc set of G.

A **path** from node v_1 to node v_k in G is a list of nodes $[v_1, ..., v_k]$ such that (v_i, v_{i+1}) is an arc for $i \in [1..k-1]$. The path **contains** node v_i for $i \in [1..k]$ and arc (v_i, v_{i+1}) for $i \in [1..k-1]$. The path is **simple** if all its nodes are distinct. The path is a **cycle** if $k > 1$ and $v_1 = v_k$. An undirected graph is **connected** if there is a path between every pair of nodes. The maximal connected subgraphs of G are its **connected components**. A directed graph is **strongly connected** if there is a path between every pair of nodes. The maximal strongly connected subgraphs of G are its strongly connected components. A **bridge** is an edge whose removal increases the number of connected components.

Let G be a graph for which each arc (i, j) is associated with two integers l_{ij} and u_{ij}, respectively called the **lower bound capacity** and the **upper bound capacity** of the arc. A **flow** in G is a function f satisfying the following two conditions[1] :

• For any arc (i, j), f_{ij} represents the amount of some commodity that can "flow" through the arc. Such a flow is permitted only in the indicated direction of the arc, i.e., from i to j. For convenience, we assume $f_{ij} = 0$ if $(i, j) \notin U(G)$.

• A **conservation law** is observed at each node: $\forall j \in X(G) : \sum_i f_{ij} = \sum_k f_{jk}$.

A **feasible flow** is a flow in G that satisfies the **capacity constraint**, that is, such that $\forall (i, j) \in U(G)\ l_{ij} \leq f_{ij} \leq u_{ij}$.

Definition 2 *The* **residual graph** *for a given flow f, denoted by $R(f)$, is the digraph with the same node set as in G. The arc set of $R(f)$ is defined as follows: $\forall (i, j) \in U(G)$:*

- $f_{ij} < u_{ij} \Leftrightarrow (i, j) \in U(R(f))$ *and upper bound capacity* $r_{ij} = u_{ij} - f_{ij}$.
- $f_{ij} > l_{ij} \Leftrightarrow (j, i) \in U(R(f))$ *and upper bound capacity* $r_{ji} = f_{ij} - l_{ij}$.

All the lower bound capacities are equal to 0.

---

1 Without loss of generality (see p.45 and p.297 in [1]), and to overcome notation difficulties, we will consider that if (i, j) is an arc of G then (j, i) is not an arc of G, and that all boundaries of capacities are nonnegative integers.

2.2 Notation

• $max(x)$ (resp. $min(x)$) denotes the maximum (resp. minimum) value of $D(x)$.
• $D(X)$ denotes the union of domains of variables of X (i.e. $D(X) = \cup_{x_i \in X} D(x_i)$).
• $\#(a,\tau)$ is the number of occurrences of the value a in the tuple τ.
• $\#(a,X)$ is the number of variables of X such that $a \in D(x)$.
• $Row(M)$ (resp. $Col(M)$) is the set of indices of the rows (resp. columns) of the matrix M.
• If X is a $n \times m$ array, that is X=x[i,j]), then $vars(i,*,X) = \{x[i,j], j = 1..m\}$ and $vars(*,j,X) = \{x[i,j], i = 1..n\}$.

3 Cardinality Matrix Constraint: Presentation

Definition 3 *A* **cardinality matrix** *constraint is a constraint C defined on a Matrix $M = x[i,j]$ of variable s taking their values in a set V, and on two sets of cardinality variables $rowCard[i,j]$ and $colCard[i,j]$ and*
$T(C) = \{\ \tau$ *s.t.* τ *is a tuple on* $X(C)$
and $\forall a_k \in V, \forall i \in Row(M) : \#(a_k, vars(i,*,M)) = rowCard[i,k]$
and $\forall a_k \in V, \forall j \in Col(M) : \#(a_k, vars(i,*,M)) = colCard[j,k]$
It is denoted by card-$Matrix(M,V,rowCard,colCard)$.

In order to show how a cardinality matrix constraint is represented we need first to introduce cardinality variables. The GCCs consider that the lower and the upper bounds are integer. There is no problem to use variables instead of integers. In this case, the lower bound is the minimal value of the domain of the variable and the upper bound is the maximal value of the domain. We will call such variables **cardinality variables**. Thus, we can define a global cardinality constraint involving cardinality variables (abbreviated cardVar-GCC):

Definition 4 *A* **global cardinality constraint involving cardinality variables** *defined on X and card and associated with a set of values V with $D(X) \subseteq V$ is a constraint C in which each value $a_i \in V$ is associated with a cardinality variable card[i] and*
$T(C) = \{\ \tau$ *s.t.* τ *is a tuple on* $X(C)$
and $\forall a_i \in V : card[i] = \#(a_i,\tau)\}$
It is denoted by $gcc(X,V,card)$.

We propose to represent a cardinality matrix constraint by:
• one cardVar-GCC per row and one cardVar-GCC per column;
• a sum constraint involving the previous cardinality variables stating that the number of symbols taken by all the rows (resp. all the columns) is the size of the matrix;
• one cardinality (0,1)-matrix constraint involving cardinality variables per symbol. Such a constraint involves boolean variables corresponding to the presence or the absence of the symbol for a cell of the matrix, and combines the rows and the columns for the symbol.

Thus, with such a representation the communication is improved in two ways:

• by the presence of cardinality variables

• by the introduction of a new constraint combining the rows and the columns for each symbol.

This communication will be efficent if some powerful filtering algorithms are available to reduce the domains of the cardinality variables and the domains of the boolean variables on which cardinality (0,1)-matrix constraints are defined. This is what we study in the next sections.

4 Filtering Algorithm for costVar-GCC

A GCC C is consistent iff there is a flow in the the value network of C [12]. The consistency of $gcc(X, V, card)$ is equivalent to the consistency of the constraint $gcc(X, V, l, u)$ where for every $a_i \in V$ $l[i] = min(card[i])$ and $u[i] = max(card[i])$. When the minimum or the maximum value of the domain of a cardinality variable is modified then C is modified and so the consistency of the constraint must be established again. Since the flow algorithms are incremental, a new feasible flow can be computed in $O(m)$, where m is the number of arcs of the network.

Arc consistency for cardVar-GCC can be established for the variables of X by using the method of GCCs, because the problem remains the same for these variables. For the cardinality variables we are more interested in the validity of the minimum and the maximum value of the domains. Bound consistency can be established by searching for the minimum and the maximum value such that a feasible flow exists. However, the cost of this method is high and its practical advantage has not been proved in general. Therefore, we propose a simple filtering algorithm whose cost is low and which is worthwhile in practice:

Property 1 *Let $C = gcc(X, V, card)$ be a cardVar-GCC. Then, we have:*

• $\forall a_i \in V$ $card[i] \leq \#(a_i, X)$

• $\sum_{a_i \in V} card[i] = |X|$

The second point is a classical sum constraint and bound consistency can be established in $O(|V|)$. Then, we immediately have the property:

Property 2 *Let $C = gcc(X, V, card)$ be a cardVar-GCC, $GV(X)$ be the value graph of X. Then for every connected component CC of $GV(X)$ we have:*
$\sum_{a_i \in vals(CC))} card[i] = |vars(CC)|$,
where $vals(CC)$ denotes the values of V belonging to CC and $vars$ denotes the variables of X belonging to CC.

proof: All the connected components are disjoint by definition, thus the problem is equivalent to a disjunction of GCCs, each of them corresponding to a connected component. Then, Property 1 can be independently applied on each GCC. $\odot$

The filtering algorithm associated with cardinality variables is defined by Property 1 and by Property 2. Its complexity is in $O(|V|)$ for all the sum constraints that can be defined and $O(m)$ for the search for connected components, where m is the number of edges of the value graph of X [15].

At first glance, Property 2 seems weak, but in fact this is not true, as shown by the following property:

Property 3 *Let $C = gcc(X, V, card)$ be a cardVar-GCC, cy be a cardinality variable, and k be an integer. If $cy = k$ in every solution of C then the domain of cy is set to k after establishing arc consistency of the X variables and after establishing bound consistency of sum constraints defined by Property 2*

In order to prove this property we need first to introduce a theorem which is a generalization of a property used to establish arc consistency for a GCC, because it deals with any kind of lower and upper bound capacities, and not only (0,1).

Theorem 1 *Let f be a feasible flow in N, and (x, a) be an arc of N. Then, for every feasible flow f' in N: $f_{xa} = f'_{xa}$ if and only if one of the following property is satisfied:*

(i) $(x, a) \notin R(f)$ and $(a, x) \notin R(f)$

(ii) $R(f)$ contains (x, a) or (a, x) but not both and x and a belong to two different strongly connected components of $R(f)$

(iii) $(x, a) \in R(f)$ and $(a, x) \in R(f)$ and (x, a) is a bridge of $ud(scc(R(f), x))$, where $ud(scc(R(f), x))$ is the undirected version of the strongly connected component of $R(f)$ containing x.

proof: (i) From definition of $R(f)$, this means that $l(x, a) = u(a, x)$, so the flow value cannot be changed in any feasible flow.

(ii) The flow theory claims that:

- the flow value of (x, a) can be increased if and only if $(x, a) \in R(f)$ and there is path from a to x in $R(f) - \{(a, x)\}$, that is in $R(f)$ in this case because we have $(x, a) \in R(f) \Rightarrow (a, x) \notin R(f)$.
- the flow value of (x, a) can be decreased if and only if $(a, x) \in R(f)$ and there is path from x to a in $R(f) - \{(x, a)\}$, that is in $R(f)$ in this case because we have $(a, x) \in R(f) \Rightarrow (x, a) \notin R(f)$.

So in this case, a flow value is constant if and only if a and x belong to two different strongly connected components.

(iii) We will call non trivial (u, v) cycle, a directed cycle which contains (u, v) but not (v, u). There are two possibilities:

1) there is a non trivial (x, a) cycle or a non trivial (a, x) cycle. This means that the flow can be increased or decreased, therefore it has not the same value for every feasible flow. Moreover, there exists a directed cycle which is non trivial, so this cycle is also a cycle in the undirected version and the arc (x, a) is not a bridge and conversely.

2) there does not exist a non trivial (x, a) cycle and there does not exist a non trivial (a, x) cycle. Let $X(x)$ be the set of nodes of $scc(R(f) - \{a\}, x)$, and $X(a)$ be the set of nodes of $scc(R(f) - \{x\}, a)$. Then $\forall p \in X(x), p \neq x$ and $\forall q \in X(a), q \neq a$, we can prove that the arcs (p, q), (q, p), (x, q), (q, x), (a, p), (p, a) do not exist. Suppose that (p, q) exists. Then, there is a path from x to p which does not contain a and an arc (p, q) and a path from q to a which does not contain x, therefore this means that we

identify a non trivial (a, x) cycle, which contradicts the hypothesis. A similar reasoning is valid for all the arcs. Hence, if (x, a) and (a, x) are removed from $R(f)$ then x and a will belong to two different connected component of the undirected version of $R(f)$. This is equivalent to saying, that (x, a) is a bridge.$\odot$

Now, we can give a **proof of Property 3:** Let a be the value whose cardinality is cy, and f be a feasible flow of $N(C)$, the value network of C. For convenience we will use $USCC = ud(scc(R(f), a))$, $SCC = scc(R(f), a)$, $CC = cc(GV(X), a)$, $X_S = vars(SCC)$, and $V_S = vals(SCC)$. If the flow value of (a, t) is the same for every feasible flow, then from Theorem 1 either a and t belong to different connected components or (a, t) is a bridge of $USCC$.
In the first case, this means that all the arcs between a value of SCC and t have the same direction. In other words, the flow value of these arcs is either equal to the lower bound capacity or is equal to the upper bound capacity. So we have either $|X_S| = \sum_{a_i \in V_S} min(card[i])$ or $|X_S| = \sum_{a_i \in V_S} max(card[i])$. In both cases the bound consistency of the constraint $|X_S| = \sum_{a_i \in V_S} card[i]$ will instantiate all these cardinality variables to the current flow value of their corresponding arc.
In the second case, (a, t) is a bridge of $USCC$ and the value graph does not contain t, so CC is a subgraph of $USCC$. If SCC contains t and if (a, t) is a bridge of $USCC$ then (a, t) and (t, a) exist in $R(f)$ and there is no other arc between $vals(CC)$ and t. Thus, the lower and the upper bound capacities are equal for every value of $vals(CC)$ which is not equal to a. In this case, the bound consistency of the sum constraint involving cy will instantiate cy to the current flow value of its corresponding arc. $\odot$

5 Cardinality (0,1)-Matrix Constraint

5.1 Absence of Cardinality Variables

Definition 5 *Let $M = x[i, j]$ be a matrix of (0,1)-variables. A* **Cardinality (0,1)-Matrix** *constraint is a constraint C defined on M in which*

- *every row i is associated with two positive integers $lr[i]$ and $ur[i]$ with $lr[i] \leq ur[i]$*
- *every column j is associated with two positive integers $lc[i]$ and $uc[i]$ with $lc[i] \leq uc[i]$, and*

$T(C) = \{\ \tau$ *s.t.* τ *is a tuple on* $X(C)$
and $\forall i \in Row(M) : lr[i] \leq \sum_{j \in Col(M)} x[i, j] \leq ur[i]$
and $\forall j \in Col(M) : lc[j] \leq \sum_{i \in Row(M)} x[i, j] \leq uc[j]\}$
It is denoted by card-(0,1)-$Matrix(M, lr, ur, lc, uc)$.

This constraint corresponds to a generalization of a well known problem named "Matrices composed of 0's and 1's" by Ford and Fulkerson [6]. In this latter problem, there is no lower bound for the rows and no upper bound for the columns. Both Ryser [14] and Gale [7] independently showed that this problem can be solved by using a flow. The introduction of lower bounds on rows and upper bounds on columns only slightly modified the flow:

Definition 6 *Given $M = x[i,j]$ a matrix of (0,1)-variables and C = card-(0,1)-$Matrix(M, lr, ur, lc, uc)$ a cardinality (0,1)-matrix constraint; the bipartite network of C, denoted by $N(C)$, consists of a node set defined by:*

- *a set of nodes $SR = \{r_1, ..., r_n\}$ corresponding to the rows of M.*
- *a set of nodes $SC = \{c_1, ..., c_m\}$ corresponding to the columns of M.*
- *a source node s and a sink t*

and an arc set A defined by:

- *$\forall r_i \in SR$ $(s, r_i) \in A$ with a lower bound capacity equal to $lr[i]$ and an upper bound capacity equal to $ur[i]$.*
- *$\forall c_j \in SC$ $(c_j, t) \in A$ with a lower bound capacity equal to $lc[j]$ and an upper bound capacity equal to $uc[j]$.*
- *$\forall r_i \in SR$, $\forall c_j \in SC$ $(r_i, c_j) \in A$ with a capacity equal to $x[i,j]$, that is the lower bound capacity is equal to $min(x[i,j])$ and the upper bound capacity is equal to $max(x[i,j])$.*
- *an arc (t, s) without capacity constraint.*

Note that the (0,1)-variables define the capacity constraints of the arcs between nodes corresponding to rows and nodes corresponding to columns.

Proposition 1 *C is consistent if and only if there is a feasible flow in the bipartite network of C.*

We can establish arc consistency of the card-(0,1)-Matrix constraint by a similar method to the one used for GCCs[2]:

Proposition 2 *Let C be a consistent cardinality (0,1)-Matrix constraint and f be a feasible flow in the bipartite network of C. Then we have:*
$\forall r_i \in SR$, $\forall c_j \in SC$: r_i and c_j do not belong to the same strongly connected component in $R(f)$ if and only if $x[i,j] = f_{r_i c_j}$.

proof: Immediate from Properties (i) and (ii) of Theorem 1 (Property (iii) cannot be applied because the capacity of the arcs between rows and columns are 0 or 1). ⊙

Thus, arc consistency can be established by only one identification of the strongly connected components in $R(f)$, that is in $O(|M|)$.

The advantage of the cardinality (0,1)-matrix constraint is emphasized by the following theorem:

Theorem 2 *Consider C = card-(0,1)-matrix(M, lr, ur, lc, uc) a cardinality (0,1)-matrix constraint. Establishing arc consistency for C ensures that for every $p \times q$ rectangle, denoted by T we simultaneously have:*

$$\sum_{(i,j)\in T} x[i,j] \geq \sum_{i \in Row(T)} lr[i] \quad - \sum_{j \in (Col(M) - Col(T))} uc[j] \tag{1}$$

$$\sum_{(i,j)\in T} x[i,j] \geq \sum_{i \in Col(T)} lc[i] \quad - \sum_{j \in (Row(M) - Row(T))} ur[j] \tag{2}$$

[2] A similar constraint, althrough expressed in a quite different way, with the same kind of algorithm to establish arc consistency, is given in [10].

proof: C is consistent. Consider Q the rectangle containing the same rows as T and the columns that are not contained in T. Every feasible flow of $N(C)$ satisfied the constraints on the rows: $\sum_{(i,j)\in(T\cup Q)} x[i,j] \geq \sum_{i\in Row(T)} lr[i]$. We have $\sum_{(i,j)\in(T\cup Q)} x[i,j] = \sum_{(i,j)\in T} x[i,j]+\sum_{(i,j)\in Q} x[i,j]$, so $\sum_{(i,j)\in T} x[i,j]+\sum_{(i,j)\in Q} x[i,j] \geq \sum_{i\in Row(T)} lr[i]$. Moreover $\sum_{(i,j)\in Q} x[i,j] \leq \sum_{j\in(Col(M)-Col(T))} uc[j]$, because the constraints on the columns of Q are satisfied. So, Equation 1 is satisfied.

Similarly, consider Q the rectangle containing the same columns as T and the rows that are not contained in T. Every feasible flow of $N(C)$ satisfies the constraints on the columns: $\sum_{(i,j)\in(T\cup Q)} x[i,j] \geq \sum_{i\in Col(T)} lc[i]$. We have $\sum_{(i,j)\in(T\cup Q)} x[i,j] = \sum_{(i,j)\in T} x[i,j]+\sum_{(i,j)\in Q} x[i,j]$, so $\sum_{(i,j)\in T} x[i,j]+\sum_{(i,j)\in Q} x[i,j] \geq \sum_{i\in Col(T)} lc[i,k]$. Moreover $\sum_{(i,j)\in Q} x[i,j] \leq \sum_{j\in(Row(M)-Row(T))} ur[j]$, because the constraints on the rows of Q are satisfied. So, Equation 2 is satisfied. $\odot$

A corollary of this theorem is close to a necessary condition of a theorem proposed for latin square by Ryser [13]:

Corollary 1 *If* $\forall i \in Row(M)$ $lr[i] = ur[i] = 1$, *and* $\forall j \in Col(M)$ $lc[j] = uc[j] = 1$, *then* $\sum_{(i,j)\in T} x[i,j] \geq p-(n-q)$

Thus, with only one cardinality (0,1)-matrix constraint we are able to take into account a property which is available for all $p \times q$ rectangles involved in the constraint. Instead of having an exponential number of cardinality constraints (because every row and column can be permuted) we have only one cardinality (0,1)-matrix constraint.

5.2 Introduction of cardinality variables

In a way similar as the one used for GCCs we propose to introduce cardinality variables in Cardinality (0,1)-Matrix constraint.

Definition 7 *Let $M = x[i,j]$ be a matrix of (0,1)-variables. A* **Cardinality (0,1)-Matrix constraint involving cardinality variables** *is a constraint C defined on M and rowCard and colCard in which*

- *every row i is associated with a cardinality variable $rowCard[i]$*
- *every column j is associated with a cardinality variable $colCard[j]$,and*

$T(C) = \{\ \tau$ *s.t.* τ *is a tuple on* $X(C)$
and $\forall i \in Row(M) : \sum_{j\in Col(M)} x[i,j] = rowCard[i]$
and $\forall j \in Col(M) : \sum_{i\in Row(M)} x[i,j] = colCard[j]\}$

It is denoted by card-(0,1)-$Matrix(M, rowCard, colCard)$.

The consistency of $C = card$-(0,1)-$matrix(M, lr, ur, lc, uc)$ is equivalent to the consistency of the constraint $card$-(0,1)-$matrix(M, rowCard, colCard)$ where $\forall i \in Row(M)$ $lr[i] = min(rowCard[i])$ and $ur[i] = max(rowCard[i])$ and $\forall j \in Col(M)$ $lc[i] = min(colCard[i])$ and $uc[i] = max(colCard[i])$. When the minimum or the maximum value of the domain of a cardinality variable is modified then C is modified and so the consistency of the constraint must be established again. Since the flow algorithms are incremental, a new feasible flow can be computed in $O(m)$.

Arc consistency can be established for the variables of M, because the problem remains the same for these variables. For the cardinality variables we have similar properties as for cardVar-GCCs:

Property 4 *Let $C = gcc(X, V, card)$ be a cardinality (0,1)-Matrix constraint involving cardinality variables. Then, we have:*
$\sum_{i \in Row(M)} rowCard[i] = \sum_{j \in Col(M)} colCard[j]$

Bound consistency of a sum constraint involving n variables can be established in $O(n)$. As for cardVar-GCCs, we have the property:

Property 5 *Let C be a cardinality (0,1)-matrix constraint involving cardinality variables, $ud(N(C) - \{s,t\})$ be the undirected version of the network of C in which the node s and t have been removed. Then for every connected component CC of $ud(N(C) - \{s,t\})$ we have:*
$\sum_{i \in Row(CC))} rowCard[i] = \sum_{j \in Col(CC)} colCard[j]$,
where $Row(CC)$ denotes the rows of M belonging to CC and $Col(CC)$ denotes the columns of M belonging to CC.

proof: All the connected components are disjoint by definition, thus the problem is equivalent to a disjunction of cardinality (0,1)-matrix constraint, each of them corresponding to a connected component. Then, Property 1 can be independently applied on each constraint. $\odot$

The filtering algorithm associated with cardinality variables is defined by Property 4 and by Property 5. Its complexity is $O(|Row(M)| + |Col(M)|)$ for all the sum constraints that can be defined and $O(m)$ for the search for connected components, where m is the number of edges of $ud(N(C) - \{s,t\})$.

6 Filtering Algorithm for the Cardinality Matrix Constraint

A cardinality matrix constraint is modeled by a cardVar-GCC on every row, a cardVar-GCC on every column, a constraint between the sum of cardinality variables, and a cardinality (0,1)-matrix constraint per symbol:

Definition 8 *Let $C = card\text{-}Matrix(M, V, rowCard, colCard)$ be a cardinality matrix constraint involving n rows, m columns and s symbols. Then,*

- *for every row i we define $Cr_i = gcc(vars(i, *, M), V, vars(i, *, rowCard))$*
- *for every column j we define $Cc_j = gcc(vars(*, j, M), V, vars(j, *, colCard)$*
- *for every value $a_k \in V$ we define the cardinality (0,1)-matrix*

$Cm_k = card\text{-}(0,1)\text{-}matrix(B_k, vars(*, k, rowCard), vars(*, k, colCard))$

- *for every value $a_k \in V$ and for every variables $x[i,j], i = 1..n, j = 1..m$*

we define the (0,1)-variable $b[i,j,k]$ and the constraint $b[i,j,k] = 1 \Leftrightarrow a_k \in D(x[i,j])$. We will denote by B_k all the (0,1)-variables defined from a_k,and by Cb_k the set of constraints defined from a_k.

• *we define the constraints:* $Cgr : \sum_{i=1..n} \sum_{k=1..s} rowCard[i,k] = nm$ *and* $Cgc : \sum_{j=1..m} \sum_{k=1..s} colCard[j,k] = nm$
Given

$$X_Q = M \ \cup \ rowCard \ \cup \ colCard \ \cup (\bigcup_{k=1..s} B_k)$$
$$\mathcal{D}(X_Q) \text{ the set of domains of the variables } X_Q$$

$$C_Q = \bigcup_{i=1..n} Cr_i \cup \bigcup_{j=1..m} Cc_i \cup \bigcup_{k=1..s} Cb_k \cup \bigcup_{k=1..s} Cm_k \cup Cgr \cup Cgc$$

The constraint network $\mathcal{Q} = (X_Q, \mathcal{D}(X_Q), C_Q)$ *is called* **the constraint network associated with a card-matrix constraint**.

Proposition 3 *Given* $C = card\text{-}matrix(M, V, rowCard, colCard)$*, and* $\Pi = (M, \mathcal{D}(M), \{C\})$ *be a constraint network and* $\mathcal{Q}$ *the constraint network associated with* C *then,* Π *is satisfiable iff* $\mathcal{Q}$ *is satisfiable.*

proof: When the M variables of Π are instantiated, the Cb constraint s of $\mathcal{Q}$ instantiated the (0,1)-variables of $\mathcal{Q}$ and since a solution satisfied the cardinality constraint for all the symbols then a solution of Π is a solution of $\mathcal{Q}$. Conversely, a solution of $\mathcal{Q}$ is obviously a solution of Π because the M and the cardinality variables of $\mathcal{Q}$ are the variables of Π, and the constraints of Π are satisfied by any solution of $\mathcal{Q}$.$\odot$

So, a card-matrix constraint can be filtered by applying arc consistency to the constraint network associated with it.

7 Experiments

In order to perform comparisons with other approaches for which there are results reported in the literature we performed our empirical analysis for the particular case of the cardinality matrix constraint in which each value has to be assigned at most once in each row and column (alldiff matrix constraint). We used hard Latin square instances. (The benchmark instances are available from: http://mat.gsia.cmu.edu/COLOR02 or from gomes@cs.cornell.edu.)

A new strategy to select the next variable and the value to branch on for Latin square problems was proposed in [4]. This strategy clearly outperforms all the previous ones that have been tested. It consists of selecting the variable with the minimum domain size and then select the value which occurs the fewest times in the domains of the variables of the rows and the columns of the selected variable. We will denote it by dom-lessO. This strategy is a kind of minimum conflict strategy. We have improved this strategy by breaking the tie of variables. When two variables have the same size of domain we select the one for which the number of instantiated variables of the row and of the column is maximum. We tested several combinations (like the minimum number of already instantiated variables), and it appears that our variant is the most robust one. Breaking ties is interesting, but the ways we break the ties seem almost equivalent. We will denote our new strategy by dom-maxB-lessO.

	2alldiff-AC dom-lessO		3alldiff-AC dom-lessO		2alldiff-GAC dom-lessO		alldiff-matrix dom-lessO	
	time	#fails	time	#fails	time	#fails	time	#fails
qwh.order30.holes316		> 50,000		> 50,000	0.33	10	0.33	3
qwh.order30.holes320		> 50,000		> 50,000	1.16	1334	0.34	22
qwh.order50.holes2000		> 50,000	1.45	230	4.6	0	5.8	0
qwh.order60.holes1440		> 50,000		> 50,000		> 50,000		> 50,000
qwh.order60.holes1620		> 50,000		> 50,000		> 50,000	66.9	24,604
qwh.order60.holes1692		> 50,000		> 50,000	15.96	7,084	7.57	7,917
qwh.order60.holes1728		> 50,000		> 50,000		> 50,000	3.16	14
qwh.order60.holes1764		> 50,000		> 50,000	3.4	277	3.68	150
qwh.order60.holes1800		> 50,000		> 50,000	3.9	554	3.4	3
qwh.order70.holes2450		> 50,000		> 50,000	5.77	24	6.5	1
qwh.order70.holes2940		> 50,000		> 50,000	9.7	398	10.8	74
qwh.order70.holes3430		> 50,000		> 50,000	14.4	0	17	0

	2alldiff-GAC dom-lessO		2alldiff-GAC dom-maxB-lessO		alldiff-matrix dom-lessO		alldiff-matrix dom-maxB-lessO	
	time	#fails	time	#fails	time	#fails	time	#fails
qwh.order30.holes316	0.33	10	0.62	476	0.33	3	0.37	44
qwh.order30.holes320	1.16	1334	0.33	21	0.34	22	0.35	32
qwh.order50.holes2000	4.6	0	4.57	1	5.8	0	5.7	1
qwh.order60.holes1440		> 50,000		> 50,000		> 50,000	2.32	18
qwh.order60.holes1620		> 50,000		> 50,000	66.9	24,604	6.54	1,439
qwh.order60.holes1692	15.96	7,084	2.75	54	7.57	7,917	3.15	47
qwh.order60.holes1728		> 50,000	2.7	4	3.16	14	3.16	9
qwh.order60.holes1764	3.4	277	2.82	1	3.68	150	3.28	12
qwh.order60.holes1800	3.9	554	15.28	1,369	3.4	3	4.0	261
qwh.order70.holes2450	5.77	24	5.7	1	6.5	1	6.6	35
qwh.order70.holes2940	9.7	398	9.5	145	10.8	74	11.1	130
qwh.order70.holes3430	14.4	0	14.2	0	17	0	17.2	2

We present two sets of results. The first one is a comparison of our method with the approach of [4], the most competitive CP based strategy. We will see that our approach, using the alldiff matrix constraint, outperforms the approach reported in [4]. The second one is a comparison of the branching strategies when the alldiff matrix constraint is used.

The "2alldiff-GAC" method is the classical model using 2 alldiff constraints associated with the filtering algorithm establishing arc consistency. The "3alldiff-AC" method is the model in which 3 alldiff constraints have been used but the global constraints are not used, and "2alldiff-AC" method uses only 2 alldiff constraints. This latter method has been used by [4]. All the experiments have been performed on a Pentium IV M, 2Mhz running under Windows XP Professional, and ILOG Solver 6.0. The code is available upon request from the authors. Thus, these experiments are reproducible.

These results clearly show that:

- difficult instances cannot be solved without efficient filtering algorithms
- the alldiff-matrix clearly outperforms 2alldiff models
- the branching strategy we propose is better than the previous ones.

Several instances remain open for a CP approach: qwh.order40.holes528, qwh.order40.holes544,qwh.order40.holes560, qwh.order33.holes.381.bal, qwh.order50.holrd825.bal. The instance qwh.order35.holes405 is solved with our approach in 9,900 s and 6,322,742 backtracks.

8 Conclusion

We present the Cardinality Matrix Constraint to efficiently model cardinality matrix problems. We also propose a simple filtering algorithm of low cost to reduce the ranges of the cardinality variables. The cardinality (0,1)-matrix constraint is a particular case of the transportation problem, a well-studied network flow problem, and it provides a good representation to capture the interactions between rows, columns, and symbols. We report results for the Alldiff Matrix constraint, a particular case of the Cardinality Matrix Constraint. Our results show that the Alldiff Matrix constraint clearly outperforms standard formulations of Alldiff Matrix problems.

References

1. R. Ahuja, T. Magnanti, and J. Orlin. *Network Flows.* Prentice Hall, 1993.
2. C. Berge. *Graphe et Hypergraphes.* Dunod, Paris, 1970.
3. M. Dincbas, H. Simonis, and P. Van Hentenryck. Solving the car-sequencing problem in constraint logic programming. In *ECAI'88, proceedings of the European Conference on Artificial Intelligence*, pages 290–295, 1988.
4. I. Dotu, A. del Val, and M. Cebrian. Redundant modeling for the quasigroup completion problem. In *Proceedings of CP'03*, pages 288–302, 2003.
5. K. Easton, G. Nemhauser, and M. Trick. Sports scheduling. In J. Leung, editor, *Handbook of Scheduling: Models, Algorithms and Performance Analysis.* CRC Press, 2004.
6. L. Ford and D. Fulkerson. *Flows in Networks.* Princeton University Press, 1962.
7. D. Gale. A theorem on flows in networks. *Pacific J. Math*, 7:1073–1082, 1957.
8. C. Gomes and J.-C. Regin. The alldiff matrix. Technical report, Intelligent Information Institute - Cornell University, 2003.
9. I. Katriel and S. Thiel. Fast bound consistency for the global cardinality constraint. In *Proceedings CP'03*, pages 437–451, Kinsale, Ireland, 2003.
10. W. Kocjan and P. Kreuger. Filtering methods for symmetric cardinality constraints˙ In *First International Conference, CPAIOR 2004*, pages 200–208, Nice, France, 2004.
11. M. Milano (ed.). *Constraint and Integer Programming: Toward a Unified Methodology.* Kluwer, 2003.
12. J.-C. Régin. Generalized arc consistency for global cardinality constraint. In *Proceedings AAAI-96*, pages 209–215, Portland, Oregon, 1996.
13. H. Ryser. A combinatorial theorem with application to latin rectangles. *Proc. Amec. Math. Soc.*, 2:550–552, 1951.
14. H. Ryser. Combinatorial properties of matrices of zeros and ones. *Canad. J. Math*, 9:371–377, 1957.
15. R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal of Computing*, 1:146–160, 1972.
16. R. Tarjan. *Data Structures and Network Algorithms.* CBMS-NSF Regional Conference Series in Applied Mathematics, 1983.