Global Matrix Constraints*

George Katsirelos[1], Nina Narodytska[2], Claude-Guy Quimper[3], and Toby Walsh[2]

[1] LRI, Université Paris Sud 11, Paris, France, email: gkatsi@gmail.com
[2] NICTA and University of NSW, Sydney, Australia, email:
{nina.narodytska,toby.walsh}@nicta.com.au
[3] Université Laval. Québec, Canada, email: claude-guy.quimper@ift.ulaval.ca

Abstract. We study the propagation of constraints that apply to a whole matrix of decision variables. We identify several cases where propagation is fixed parameter tractable, as well as other cases where propagation is intractable. We find that the number of rows (or columns) in the matrix as a useful parameter in describing the complexity of propagation of such matrix constraints.

1 Introduction

Matrices of decision variables are a common pattern in many constraint models [1, 2]. So far, research has largely focused on the role of such matrices in modelling. Here, we launch a research programme to study their propagation. We give both general results, that apply to families of global constraints, and specific results, that apply to some specialized global constraints seen in a number of common matrix models.

Formal background: we will use the theory of parameterized complexity [3, 4]. A problem is *fixed-parameter tractable* (FPT) if it can be solved in $O(f(k)n^c)$ time where f is *any* computable function, k is a parameter, c is a constant, and n is the input size. Above FPT, there is a hierarchy of fixed-parameter *intractable* problem classes: $\text{FPT} \subseteq W[1] \subseteq W[2] \subseteq \ldots \subseteq \text{XP}$. The class $W[t]$ is characterized by the depth t of unbounded fan-in gates in a Boolean circuit specifying the problem. $W[1]$-hard problems are believed to be intractable as the halting problem for non-deterministic Turing machines is $W[1]$-complete in the length of the accepting computation.

2 Global matrix constraints

A *matrix constraint* is a global constraint posted on a matrix of decision variables. For example, we can eliminate all row symmetry in a matrix model by posting a LEXCHAIN matrix constraint to ensure that the rows are lexicographically ordered [5]. As a second example, with row and column symmetry, we can post a DOUBLELEX matrix constraint to ensure that the rows and columns are lexicographically ordered [6]. As a third example, we can use a REGULARSUM matrix constraint in shift scheduling problems to ensure that each row satisfies a REGULAR constraint (describing shift rules) and that

* Supported by ANR UNLOC project, ANR 08-BLAN-0289-01 and the Australian Government's Department of Broadband, Communications and the Digital Economy and the ARC.

each column has a given sum (representing the number of workers on duty). As a fourth and final example, we can model many rostering, sports scheduling and timetabling problems with the cardinality matrix constraint [7]; this is equivalent to global cardinality constraints on the individual rows and columns.

Whilst matrix constraints are used in many constraint models, our understanding of how to propagate them is more patchy. Typically, our understanding fits one of three cases. In the first case, we know that propagation is intractable. For example, enforcing domain consistency on DOUBLELEX is NP-hard [10]. In the second case, we know that propagation is tractable and we have either a specialized propagator or a decomposition that does not hinder propagation. For example, an efficient and complete propagation algorithms exists for the LEXCHAIN matrix constraint [5]. In the third case, we have incomplete propagation methods and it is not yet clear under what conditions propagation is tractable, or our methods are complete. For example, incomplete propagation methods have been proposed for the cardinality matrix constraint [7]. One of our contributions is to reduce the uncertainty here by proving that propagating the cardinality matrix constraint is intractable even under some strong restrictions. Another of our contributions is to identify conditions (like a bounded number of rows or columns) under which propagation of various matrix constraints becomes tractable.

3 Some general theory

Theory can provide some general insights into the propagation of matrix constraints. To illustrate this, we consider *row-wise decomposable* matrix constraints. This is the class of matrix constraints that are logically equivalent to a set of constraints posted either on individual rows or on pairs of rows. For example, the LEXCHAIN matrix constraint is row-wise decomposable. On the other hand, the matrix constraint which ensures that a matrix contains an even number of non-zero entries is not row-wise decomposable. In general, it is not tractable to propagate a row-wise decomposable matrix constraint. When is it polynomial? One condition is when there is a small and bounded number of columns. For example, this holds for matrix models of `prob033` (word design for DNA computing) and `prob036` (fixed length error correcting codes) in CSPLib [11]. This condition holds in matrix models where the columns represent a fixed resource (e.g. distribution centres or production lines).

Observation 1 *Enforcing domain consistency on a row-wise decomposable constraint is fixed-parameter tractable in the number of columns.*

Proof: Suppose we have n rows, m columns and domains of maximum size d. We can channel each row into a single variable with at most d^m values. The row-wise decomposable constraint can then be replaced by a tree of constraints on these variables. Enforcing domain consistency on the decomposition ensures domain consistency on the original problem. Enforcing domain consistency on the channeling constraints takes $O(nd^{2m})$ time, and on the each of at most $O(n^2)$ constraints on the introduced variables in $O(d^{2m})$ time. Hence, the total time complexity is $O(n^2d^{2m})$. □

Note that this result holds even when the row-wise decomposition is intractable to propagate. The proof only assumes that it is polynomial to check whether an assignment satisfies the row-wise decomposition. Unfortunately, it is difficult to identify other

tractable cases without making assumptions about the precise semantics of the matrix constraints. In the next section, we make such assumptions by focusing on some matrix constraints that have appeared in the literature.

4 Some special cases

4.1 Cardinality matrix constraint

Our first case study is a strong negative result. We prove that propagating the cardinality matrix constraint is NP-hard even in a very restricted setting. A special case of the cardinality matrix constraint is the ALLDIFFERENT matrix constraint in which values are not repeated along any row or column [7]. This can be used to model quasigroup and related problems in fiber optical routing. It is already known that enforcing domain consistency on this constraint is NP-hard [8]. Our contribution here is to show that the constraint remains NP-hard even in the case of just 2 rows.

Theorem 1 *Enforcing domain consistency on a* $2 \times m$ ALLDIFFERENT *matrix constraint is* NP*-hard.*

Proof: We will describe first a simple gadget that we use in the rest of the reduction, inspired by a similar gadget in [9]. This gadget is a 2×3 submatrix (Table 1(a)). There exists a distinguished value i and two "copies" of i, c_1 and c_2. This gadget ensures that the values c_1 and c_2 have to be used in the top row if i is used in the top row anywhere outside the gadget, otherwise there exists a solution to the submatrix that uses neither of c_1 and c_2 but uses i. In this sense, the gadget "destroys" the value i and creates two copies. Note that, aside from i, c_1 and c_2, the other values are unique to the gadget

$\{i, p\}$	$\{p, c_1\}$	$\{t_2, c_2\}$
$\{p, t_1\}$	$\{q\}$	$\{t_1, t_2\}$

(a)

p	c_1	c_2
t_1	q	t_2

(b)

i	p	t_2
p	q	t_1

(c)

Table 1. 2×3 submatrix gadget

(i.e., do not appear in any other part of the construction). If i is used elsewhere, the top left cell has to take p, so the values c_1 and c_2 are necessarily used (Table 1(b)). If i is not used elsewhere, the top left cell can consume i and leave c_1 and c_2 unused (Table 1(c)).

We reduce a 3-SAT formula ϕ on N variables and M clauses to a matrix with 2 rows as follows. Intuitively, we use a cell in the first row for each variable and one for each column. We represent an assignment by assigning i for true and $-i$ for false to X_{1i}. We then use the gadget above to construct unique copies $c_{1,j}, c_{2,j}, c_{-3,j}$ of the values $1, 2$ and -3 if the j^{th} clause is $(-1, -2, 3)$ and its cell has domain $\{c_{1,j}, c_{2,j}, c_{-3,j}\}$. An assignment that violates it assigns $X_{11} = 1$, $X_{12} = 2$ and $X_{13} = -3$ and therefore consumes also the copies of these values, leaving no option for the cell of clause j.

More formally, in our construction we have 1 column for each variable. The first row in column i has domain $\{-i, i\}$. After that, we have $2N$ blocks of columns, 1 for each literal. The $2i^{th}$ block uses the gadget to create $2^{\lceil \log d_{-i} \rceil}$ copies of the value i, where d_{-i} is the degree of the literal $-i$ (the number of negative occurences of the variable i). We copy i into c_{i_1}, c_{i_2}, then c_{i_1} into c_{i_3}, c_{i_4} and so on (Table 2). Note that $c_{i,1}, \ldots$

are unique for each instance of the gadget. The $(2i+1)^{th}$ block generates d_i copies of the value $-i$. For each clause j that contains the literal $-i$ (resp i), there exists a value $c_{i,j}$ (resp $c_{-i,j}$) that must be used iff $X_{1i} = -i$ (resp $X_{1i} = i$). The total number of columns for these copies is $O(N \log M)$. Finally, we have a block of M columns, one for each clause, so the cell in the first row of the k^{th} column has domain $\{-x \mid x \in c_k\}$. For example, if the clause is $(1 \vee -2 \vee 3)$, the domain of the first row is $\{-1, 2, -3\}$. The second row has disjoint singleton domains anywhere outside instances of the gadet.

$\{i, p\}$	$\{p, c_1\}$	$\{t_2, c_2\}$	$\{c_1, p'\}$	$\{p', c_3\}$	$\{t_4, c_4\}$
$\{p, t_1\}$	$\{q\}$	$\{t_1, t_2\}$	$\{p', t_3\}$	$\{q'\}$	$\{t_3, t_4\}$

Table 2. The connection between two gadgets to copy i to c_1 and c_2; c_1 to c_3 and c_4

We now show that this instance of the ALLDIFFERENT matrix constraint is satisfiable iff the original SAT instance is satisfiable. An assignment to the variables of ϕ is encoded by assigning $X_{1i} = i$ iff the variable i is true. Suppose there exists an assignment that satisfies ϕ. We create the assignment to the first N columns as described above and create the value copies in the gadgets by simple propagation. Consider a column that corresponds to the clause j. Since the clause is satisfied, there exists a literal that is true, say i. Hence, the value $-i$ is not used. As the value $c_{-i,j}$ is not used, we can assign it to the first row. Therefore, the entire matrix is filled and satisfies all constraints. Conversely, consider a solution of the ALLDIFFERENT matrix constraint. We construct an assignment to ϕ from the first N columns as described. For each clause column, we have assigned a value $c_{-i,j}$ to the first row, so value i is used. Since this maps to the variable i being true and the clause contains the literal i, the clause is satisfied. □

4.2 REGULAR^2 matrix constraint

Our next case study is a little more positive (and surprising). With a REGULAR^2 matrix constraint, we have REGULAR constraints along each row and column of a matrix. Such a matrix constraint is used in models for the nonogram problem (`prob012` in CSPLib). We consider the *restricted* REGULAR^2 matrix constraint in which variables are just 0/1 and the same REGULAR constraint applies to each row, and the same (but perhaps different) REGULAR constraint to every column. We first prove that propagating the restricted REGULAR^2 matrix constraint is NP-hard. Note we cannot appeal to the result in [12] for nonograms as this reduction uses a different REGULAR constraint for each row and column.

Theorem 2 *Enforcing domain consistency on a restricted* REGULAR^2 *matrix constraint is NP-hard.*

Proof: Reduction from 1in3-SAT with n Boolean variables on m positive clauses. We construct a $n+1$ by $2m+1$ Boolean matrix. The first row represents a truth assigment, In particular, the $i+1$th entry in the first row is set to 1 iff x_i is true. Each subsequent pair of rows represents a clause. In particular, the $i+1$th entry in the $2k-1$th row is

set to 1 iff x_i is in the kth clause. The first column in the matrix has 0, then alternates 0 with 1. For a row that starts with 0, the row automaton accepts any value. For a row that starts with 1, the row automaton accepts only those rows in which 2 entries are set to 1. The column automaton records the value of the first variable, and accepts either the first column, or any column in which the $2k$th entry is set to 1 iff both the 1st and $2k-1$th are set to 1. This restricted REGULAR^2 matrix constraint has a solution iff the 1in3-SAT problem has a model. In fact, by using additional leading rows and columns, we can give a reduction that uses the same automaton for both the rows *and* the columns. □

Naively, we can find a support for the mn variables in a m by n restricted REGULAR^2 matrix constraint in $O(d^{mn})$ time where d is the domain size. Surprisingly, assuming without loss of generality $m \geq n$, we can reduce this to $O(mn\frac{r}{q}(qr)^{2n+2m-\frac{m}{n}})$ where q and r are the number of states in the row and column automata respectively. This is a significant reduction as we move the mn term from an exponent to a coefficient.

Theorem 3 *A support for a m by n restricted* REGULAR^2 *matrix constraint can be found in $O(mn\frac{r}{q}(qr)^{2m+2n-\frac{m}{n}})$ tiem where q and r are the number of states in the row and column automata and $m \geq n$.*

Proof: We unfold each automaton m and n times. We then create a $m+1 \times n+1$ matrix K of constrained variables. The domain of of $K[i,j]$ is a pair giving all possible states of the row automaton at the $i-1$th step, and the column automaton at the $j-1$th step. The domains on the first row contains q pairs since the column automaton is in its initial state. The domains on the first column contains r pairs since the row automaton is in its initial state. All other domains have qr pairs. We have transition constraints between $K[i,j]$ and $K[i+1,j]$, and between $K[i,j]$ and $K[i,j+1]$ permitting only valid transitions. We then find a support by a divide and conquer method that recursively tries to instantiate the middle row and middle column, leaving four $\frac{m}{2} \times \frac{n}{2}$ subproblems to solve in order to validate the instantiation. When instantiating the middle row and middle column, we have q choices of values for the variable on the first row, r choices of values for the variable on the first column, and qr choices of values for the $m+n-1$ variables that are neither on the first row nor the first column. Combined, there is a total of $(qr)^{m+n}$ possible choices. Suppose the run time complexity to solve a $m \times n$ matrix is $g(m,n)$. Then $g(m,n) = (qr)^{n+m}4g(\frac{m}{2},\frac{n}{2})$ for $n > 1$ and $m > 1$ and $g(m,1) = mr^2$ as the base case. To solve this recurrence, let $n = 2^k$ and $m = z2^k$ for some value $z \geq 1$ and $h(k) = g(m,n)$. Then $h(k) = 4(qr)^{(z+1)2^k}h(k-1)$. That is, $h(k) = 4^k(qr)^{(z+1)2^{k+1}-z-1}zr^2$. This means $g(m,n) = n^2(qr)^{2m+2n-\frac{m}{n}-1}\frac{m}{n}r^2 = mn\frac{r}{q}(qr)^{2m+2n-\frac{m}{n}}$. □

We next identify conditions under which propagation of the REGULAR^2 is fixed parameter tractable. We first show that if we simply bound the number of rows, the constraint remains $W[1]$-hard to propagate, even when we restrict the column automata to encode a linear inequality with unit coefficients on 0/1/-1 variables. We denote this special case by REGULARSUM. As mentioned before, this can be used to model rostering and related problems.

Theorem 4 *Enforcing domain consistency on the* REGULARSUM *matrix constraint is $W[1]$-hard when the parameter is the number of rows even with just 3 values.*

Proof: We reduce from weighted 3-SAT. Let ϕ be a 3-SAT formula with N variables and M clauses, which we try to satisfy with $k > 0$ variables set to true. We write i for the true literal of the i^{th} variable, $-i$ for its false literal, c_j for the j^{th} clause. $var(i) \in c_j$ is true if i or $-i$ is in c_j. $f(j)$ is the number of false literals in c_j.

We construct an instance of REGULARSUM with $\lceil \log N \rceil + N + M + 1$ columns and k rows on the alphabet $\{-1, 0, 1\}$. The automaton has three sets of states, besides the initial state q_0. The first set encodes a choice of a true variable by treating the first $\lceil \log N \rceil$ symbols of each row as the binary representation of a variable index. We denote these states by the term $c(i, S)$, $0 \leq i \leq \lceil \log N \rceil$ where S is a set of variable indices. In this notation, $q_0 \equiv c(0, [1, N])$. The next N columns visit states denoted by the term $v(i, j)$ with $i \in [1, N], j \in [1, N + 1]$. The sums in these columns ensure that each variable is chosen at most once. The final $M + 1$ columns visit additional states which we denote by the pair $\langle i, j \rangle, i \in [1, N], j \in [1, M]$. The sums in these columns ensure that each clause is satisfied. The automaton's transition function is as follows:

$$
\begin{aligned}
\delta(c(i, S), b) &= c(i + 1, \{x \in S | (i + 1)^{th} \text{ bit of } x \text{ is } b\}) && \text{for } b \in \{0, 1\} \\
\delta(c(\lceil \log N \rceil, \{j\}), 0) &= v(j, 1) && \\
\delta(v(i, j), 0) &= v(i, j + 1) && \text{if } i \neq j \\
\delta(v(i, i), -1) &= v(i, i + 1) && \\
\delta(v(i, N + 1), 0) &= \langle i, 1 \rangle && \\
\delta(\langle i, j \rangle, 0) &= \langle i, j + 1 \rangle && \text{if } var(i) \notin c_j \\
\delta(\langle i, j \rangle, 1) &= \langle i, j + 1 \rangle && \text{if } i \in c_j \\
\delta(\langle i, j \rangle, -1) &= \langle i, j + 1 \rangle && \text{if } -i \in c_j
\end{aligned}
$$

All other transitions lead to a rejecting absorbing state. All states $\langle i, M + 1 \rangle$ are accepting. All column sums are of the form $\sum_{i=1}^{k} X_{ij} \geq P_j$. We set $P_j = 0$ for the first $\lceil \log N \rceil + 1$ columns, $P_j = -1$ for the next $N + 1$ columns and $P_j = 1 - f(j - \lceil \log N \rceil - N - 2)$ for the rest.

In this reduction, each row corresponds to one variable that is assigned true and each column after the first $\lceil \log N \rceil + N + 2$ to one clause. The first set of states encode the variables with index whose binary encoding matches the current prefix and the bit that will be fixed next. The second set of states encode the variable that is chosen for that row in the first element and the current column in the second element. The third set of states encode the variable that is chosen for that row in the first element of the tuple and the current clause in the second. Thus, the automaton allows the choice of any variable in the transition away from q_0 but fixes it in the rest of the row.

The sums in columns $[\lceil \log N \rceil + 1, \lceil \log N \rceil + N + 2]$ ensure that no variable is chosen in two rows. If that happens, there will be two -1 entries in one column for a sum of -2, violating the sum constraint of that column. The sums in the last M columns ensure that either one of the clause's positive literals is true or that not all of its negative literals are false (i.e., the corresponding variable set to true). This is based on the well known encoding of clauses as pseudo-Boolean constraints. For example, the clause $(a \vee b \vee -c)$ is encoded as $a + b - c \geq 0$, while $(e \vee -f \vee -g)$ is encoded as $e - f - g \geq -1$. The only additional insight here is that a variable that is not set to true does not contribute to this sum. Thus the constraint over a column is projected to those variables that are true.

We show that the instance ϕ is satisfiable iff the REGULARSUM matrix constraint has a solution. Suppose ϕ is satisfiable. Then from a satisfying instance, we construct an assignment to REGULARSUM by matching a true variable with each row (in any manner) and encoding its id in the first $\lceil \log N \rceil$ columns. For the rest of the columns, if row i is matched to variable j, we assign values to the rows such that the automaton visits states $\langle j, 1\rangle, \ldots, \langle j, M\rangle$. Now consider clause j. Suppose j is satisfied by a positive literal i. Then there exists a row r such that the corresponding automaton is at state $\langle i, j\rangle$ before parsing the contents of column $\lceil \log N \rceil + N + 2 + j$. Since this state has a unique outgoing transition, the value of X_{rj} is 1. The sum of all the other rows is at most $-f(j)$, thus the sum is greater or equal to $1 - f(j)$. Suppose now that j is satisfied by a negative literal $-i$. Then, none of the automata in any row visits state $\langle i, j\rangle$. Thus, the sum of all rows is at least $-(f(j) - 1) = 1 - f(j)$.

Suppose the REGULARSUM matrix constraint has a solution. The corresponding assignment of ϕ is constructed by setting to true all the variables i such that $\langle i, j\rangle$ is visited by at least one automaton, and the rest to false. Assume some clause c_j in ϕ is not satisfied. Then, all $f(j)$ of its negative literals are false, contributing $-f(j)$ to the sum. None of c_j's true literals are true, so these do not contribute a positive term and the rest of the variables each contribute a negative term to the sum. Thus the sum of column j is less than $1 - f(j)$, and the column constraint is violated, a contradiction. □

When we also consider the size of the row automata, the REGULAR^2 matrix constraint is fixed parameter tractable. Hence, if we have both a small number of employees and simple shift rules, reflected in an automaton with a small number of states, propagation is tractable.

Observation 2 *Enforcing domain consistency on the* REGULAR^2 *matrix constraint is fixed parameter tractable in $k = n(\log Q)$ where Q is the maximum number of states in any row automaton and n the number of rows.*

Proof: Let $R_1, \ldots, R_n$ be the automata in each row with corresponding transition functions δ_i, and $R_{C_1}, \ldots, R_{c_m}$ be the automata in each column with corresponding transition functions δ_{C_i}. To simplify notation we assume all column automata and all row automata are the same, but this restriction is easily lifted. We construct an automaton $R_\wedge$ with states that are represented by the tuple $\langle c, i, q_C, q_1, \ldots, q_n\rangle$, where c is the current column, i is the current row, q_C is the current state of the column automaton and $q_1 \ldots q_n$ are the current states of each of the n row automata. The initial state is $\langle 1, 1, q_{C_0}, q_0, \ldots, q_0\rangle$, where q_{C_0} and q_0 are the initial states of the column and row automata, respectively. The transitions are as follows: $\delta(\langle c, i, q_C, q_{r_1}, \ldots, q_{r_i}, \ldots, q_{r_n}\rangle, b) = \langle c, i+1, \delta_C(q_C, b), q_{r_1}, \ldots, \delta(q_{r_i}, b), \ldots, q_{r_n}\rangle$ if $i < n$ and $\delta(\langle c, n, q_C, q_{r_1}, \ldots, q_{r_n}\rangle, b) = \langle c+1, 1, q_{C_0}, q_{r_1}, \ldots, \delta_n(q_{r_n}, b)\rangle$ if $\delta_C(q_C, b)$ is accepting. All other transitions lead to an absorbing rejecting state. A state $\langle m+1, 1, q_C, q_{r_1}, \ldots, q_{r_n}\rangle$ is accepting if all the q_{r_i} are accepting.

Note that in this unfolded automaton, i and c simply encode some information about the current layer, so they do not contribute to the explosion of the number of states. Thus, the number of states of this automaton is $nmQ^nQ_C = nQ_Cm2^{n(\log |Q|)}$, where Q_C is the number of states of the column automaton. Therefore the constraint REGULAR$([X_{11}, \ldots, X_{n1}, X_{12}, \ldots, X_{nm}], R_\wedge)$ can be propagated in time $O(n|Q_C|2^{n(\log |Q|)}m)$, which is linear in the number of columns. □

This construction is also a witness for REGULAR^2 being in the class XP when the parameter is only the number of rows: the size of the input is $\Theta(Q + nm)$, thus its complexity is of the form $O(n^k)$, as required for XP. The $R_\wedge$ automaton can also be used in a weighted REGULAR constraint so that it also computes the optimal solution. For example, in rostering the objective may minimize the number of working shifts.

5 Conclusions

Global constraints that apply to whole matrix of decision variables are a common modelling pattern. We have studied the propagation of such matrix constraints. We have identified a number of cases where propagation is polynomial. For example the REGULARSUM matrix constraint is fixed parameter tractable when we bound the number of rows and states in the automaton. We have also identified other cases where propagation is intractable. For example, the cardinality matrix constraints is NP-hard to propagate even with just two rows. In future work, we intend to study other matrix constraints (like the SCALARPRODUCT and REGULARGCC matrix constraint). In addition we intend to explore the pruning and the cost of propagating matrix constraints in practice.

References

1. Flener, P., Frisch, A., Hnich, B., Kiziltan, Z., Miguel, I., Walsh, T.: Matrix modelling. In: Proc. of Formul'01 Workshop on Modelling and Problem Formulation, CP2001.
2. Flener, P., Frisch, A., Hnich, B., Kiziltan, Z., Miguel, I., Walsh, T.: Matrix modelling: Exploiting common patterns in constraint programming. In: Proc. of Int. Workshop on Reformulating Constraint Satisfaction Problems, CP2002.
3. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer (1999)
4. Bessiere, C., Hebrard, E., Hnich, B., Kiziltan, Z., Quimper, C.G., Walsh, T.: The parameterized complexity of global constraints. In: Pro. of 23rd National Conference on AI, AAAI (2008) 235–240
5. Carlsson, M., Beldiceanu, N.: Arc-consistency for a chain of lexicographic ordering constraints. Tech. report T2002-18, Swedish Institute of Computer Science (2002)
6. Flener, P., Frisch, A., Hnich, B., Kiziltan, Z., Miguel, I., Pearson, J., Walsh, T.: Breaking row and column symmetry in matrix models. In: 8th Int. Conf. on Principles and Practices of Constraint Programming (CP-2002), Springer (2002)
7. Régin, J.C., Gomes, C.: The cardinality matrix constraint. In: 10th Int. Conf. on Principles and Practice of Constraint Programming (CP 2004). (2004) 572–587
8. C. Colbourn. The complexity of completing partial latin squares. *Discrete Applied Mathematics*, 8(1):25–30, April 1984.
9. M. Kutz, K. Elbassioni, I. Katriel, and M. Mahajan. Simultaneous matchings: Hardness and approximation. *Journal of Computer and System Sciences*, 74(5):884–897, August 2008.
10. Katsirelos, G., Narodytska, N., Walsh, T.: Static constraints for breaking row and column symmetry. In: 16th Int. Conf. on Principles and Practices of Constraint Programming (CP-2010), Springer-Verlag (2010)
11. Gent, I., Walsh, T.: CSPLib: a benchmark library for constraints. In: 5th Int. Conf. on Principles and Practices of Constraint Programming (CP-99).
12. Nagao, T., Ueda, N.: NP-completeness results for nonogram via parsimonious reductions. Tech. Report TR96-0008, Dept. of CS, Tokyo Institute of Technology (1996)