The RegularGcc Matrix Constraint

Ronald de Haan[12], Nina Narodytska[23], and Toby Walsh[23]
Ronald.de_Haan@mailbox.tu-dresden.de,
{nina.narodytska,toby.walsh}@nicta.com.au

[1] Technische Universität Dresden
[2] NICTA, Australia
[3] University of New South Wales

Abstract. We study propagation of the RegularGcc global constraint. This ensures that each row of a matrix of decision variables satisfies a Regular constraint, and each column satisfies a Gcc constraint. On the negative side, we prove that propagation is NP-hard even under some strong restrictions (e.g. just 3 values, just 4 states in the automaton, or just 5 columns to the matrix). On the positive side, we identify two cases where propagation is fixed parameter tractable. In addition, we show how to improve propagation over a simple decomposition into separate Regular and Gcc constraints by identifying some necessary but insufficient conditions for a solution. We enforce these conditions with some additional weighted row automata. Experimental results demonstrate the potential of these methods on some standard benchmark problems.

1 Introduction

Global constraints can be used to model and reason about commonly found substructures. Many such models contain matrices of decision variables [1–3]. Matrix constraints are global constraints that apply to such matrices [4]. For example, the RegularGcc matrix constraint can be used to model rostering problems. It ensures each row of the matrix satisfies a Regular constraint (representing the shift rules) and each column satisfies a Gcc constraint (representing the required capacities for each shift). We prove here that propagating the RegularGcc constraint is costly, even under very severe restrictions. Therefore, as in [5], we look for partial methods that only enforce a limited level of consistency. These methods are based on necessary conditions that improve propagation over the decomposition into separate Regular constraints on the rows and separate Gcc constraints on the columns. These necessary conditions depend on extracting several string properties from the rows. We enforce these necessary conditions by constraining the rows with additional automaton constraints. Unfortunately, when the number of columns increases, these automata increase in size quite drastically. By using weighted automata, we show that we can limit the increase in size. Finally, we show that this approach can be used in a more general setting where we have a matrix with multicostRegular and Gcc constraints.

2 Intractable cases

We first prove that propagating the RegularGcc matrix constraint is intractable even under strong conditions. More precisely, we show that enforcing bound consistency (BC) is NP-hard. This justifies why we later look for partial propagation methods based on some necessary (but not sufficient) conditions.

Theorem 1. *Enforcing BC on* RegularGcc *is* NP*-hard, already for* Regular *constraints given by a DFA of size* 4*,* Gcc *constraints specifying only an upper bound on the number of occurrences of one particular value, and just* 3 *values.*

Proof: Reduction from 3-SAT. Let $\varphi = \gamma_1 \wedge \cdots \wedge \gamma_C$ be a Boolean formula in CNF on propositional variables $p_1, \ldots, p_R$. We construct an $R \times C$ matrix $\mathcal{M}$ of decision variables taking their values from $\{-1, 0, 1\}$, where each row $1 \leq r \leq R$ corresponds to a propositional variable p_r and each column $1 \leq c \leq C$ corresponds to a clause γ_c.

To initialize the domain of variables in the matrix, we do the following for each clause $\gamma_i = l_1^i \vee l_2^i \vee l_3^i$. We set $\mathcal{M}_{r,i} = 0$ for all propositions p_r not occurring in γ_i. For $j \in \{1, 2, 3\}$ we set $\mathcal{M}_{i,k} \in \{0, 1\}$ if $l_j^i = p_k$ and we set $\mathcal{M}_{i,k} \in \{0, -1\}$ if $l_j^i = \neg p_k$.

On each column we put the Gcc constraint that states that the value 0 occurs at most $R - 1$ times. On each row we put the Regular constraint that states that besides 0's either only 1's or only -1's occur.

We show that this instance of RegularGcc has a solution iff φ is satisfiable.

($\Rightarrow$) We create a satisfying assignment I for φ as follows. For each p_r, if in row r occurs at least one 1, we let $I(p_r) = \top$, otherwise we let $I(p_r) = \bot$ (the choice of $I(p_r)$ when only 0's occur in row r is arbitrary). Since in each column c there occur only $R - 1$ many 0's, we know that there exists some p_i for which $\mathcal{M}_{i,c} \neq 0$ and thus $I(l_i^c) = \top$. Therefore $I \models \gamma_c$.

($\Leftarrow$) Let I be an assignment satisfying φ. We can instantiate $\mathcal{M}$ as follows. For each clause $\gamma_c = l_1^c \vee l_2^c \vee l_3^c$, for $j \in \{1, 2, 3\}$ we do the following. If $l_j^c = p_k$ and $I(p_k) = \top$, we let $\mathcal{M}_{k,c} = 1$. If $l_j^c = \neg p_k$ and $I(p_k) = \bot$, we let $\mathcal{M}_{k,c} = -1$. Otherwise, we let $\mathcal{M}_{k,c} = 0$. Since I is functional each Regular constraint on the rows is satisfied. Also, since at least one literal is satisfied in each clause, each column contains at least one value that is not 0, so the Gcc constraints are satisfied.

Automaton $\mathcal{A}$ in Fig. 1 witnesses that this Regular constraint can be enforced by a DFA of size 4. Note that this proof also works for any other restriction on Regular constraints that can enforce that, for two given values, in any word at most one of these values occurs. □

In fact, since we only bound the number of one particular value in the Gcc constraint, the above proof also works for the RegularAmong constraint.

A common type of Regular constraint in a RegularGcc matrix constraint is a Stretch constraint. This constrains the length of any stretch of values (e.g. there are at most 3 night shifts in a row) and the possible transitions (e.g. a night

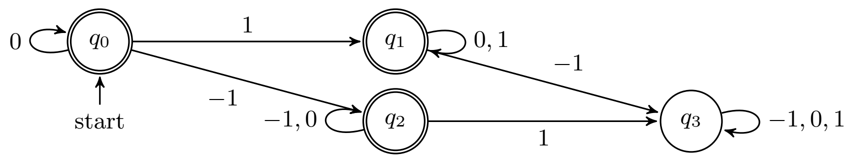

Fig. 1. Automaton $\mathcal{A}$.

shift can only be followed by a day off). Unfortunately, even this special case is intractable to propagate.

Theorem 2. *Enforcing BC on* StretchGcc *is* NP-*hard, already for just* 3 *values.*

Proof: Reduction from the Exact Cover problem. We are given $F = \{S_1, \ldots, S_n\}$ with $\bigcup_i S_i = U$. We ask if there is some subset $C \subseteq F$ with $\bigcup_{c \in C} c = U$ and $c \cap c' = \emptyset$ for all distinct $c, c' \in C$. W.l.o.g. we assume U contains the integers 1 to $|U|$.

We construct a $|F| \times |U|$ matrix $\mathcal{M}$, of decision variables taking their value in $\{-1, 0, 1\}$. For each row $1 \leq r \leq |F|$ and each value $1 \leq i \leq |U|$ we do the following. If $i \in S_r$, we let $\mathcal{M}_{r,i} \in \{0, 1\}$. If $i \notin S_r$, we let $\mathcal{M}_{r,i} \in \{-1, 0\}$.

On each column we put the Gcc constraint that states that the value 1 occurs exactly once. On each row we put the Stretch constraint stating that each stretch of 0's must have a length of at least $|U|$.

We show that this instance of StretchGcc has a solution iff there exists an exact cover.

($\Rightarrow$) Take a solution for our instance. We let C be the set of all U_r for which row r in the solution contains only -1's and 1's. Obviously $C \subseteq F$. In order to show that $\bigcup_{c \in C} = U$, it suffices to show that $U \subseteq \bigcup_{c \in C}$. Take an arbitrary $i \in U$. Since our solution contains at least one 1 in each column, we know there is some $c \in C$ such that $i \in c$. We also show that all distinct $c, c' \in C$ are disjoint. Take arbitrary $c, c' \in C$ such that $c \neq c'$. Assume $j \in c \cap c'$. This means that column j in the solution would contain two 1's, which contradicts the Gcc constraints on the columns.

($\Leftarrow$) Let $C \subseteq F$ be an exact cover. We fill $\mathcal{M}$ as follows. For each row $1 \leq r \leq |F|$, we do the following. If $S_r \in C$, fill the row with -1's and 1's (this can be done only in one way). Otherwise, fill row r with only 0's. Obviously, the Stretch constraints on the rows are satisfied. Also, since C is an exact cover, we know that for each $1 \leq i \leq |U|$ there is exactly one row r such that $\mathcal{M}_{r,i} = 1$. Thus the Gcc constraints on the columns are satisfied. $\Box$

This proof also works for the Regular constraint accepting only words that contain either only 0's or only -1's and 1's, and therefore for any restriction on Regular constraints that can enforce permitted (or forbidden) words of length two (such as the meta constraint Slide).

Corollary 1. *Enforcing BC on* SlideGcc *is* NP-*hard, already for* Slide *constraints based on constraints of arity* 2 *and just* 3 *values.*

Another common type of REGULAR constraint in a REGULARGCC matrix constraint is a SEQUENCE constraint. This limits the number of values of a particular type that occur in each sequence (e.g. at most 3 shifts in every 7 can be night shifts). This case is unfortunately also intractable to propagate. We prove that both enforcing domain consistency (DC) and enforcing bound consistency (BC) are NP-hard even if the matrix has just a few columns.

Theorem 3. *Enforcing DC on* SEQUENCEGCC *is* NP*-hard, already for just* 5 *columns.*

Proof: Reduction from the 3D Matching problem. The proof is inspired by [6]. Given are three pair-wise disjoint sets W, Z, Y of equal size q and a set $M \subseteq W \times Z \times Y$, $|M| = m$. The question is if there exists $M' \subseteq M$ such that $|M'| = q$ and no two different elements of M' agree in any coordinate.

Assume $M = \{s_1, \ldots, s_m\}$. We create a $m \times 5$ matrix $\mathcal{M}$ of decision variables taking their value in $\{0, t, w_1, \ldots, w_q, z_1, \ldots, z_q, y_1, \ldots, y_q\}$. For each $(w_i, z_i, y_i) = s_i$ we let $\mathcal{M}_{i,1} \in \{0, w_i\}$, $\mathcal{M}_{i,2} \in \{0, t\}$, $\mathcal{M}_{i,3} \in \{0, z_i\}$, $\mathcal{M}_{i,4} \in \{0, t\}$, and $\mathcal{M}_{i,5} \in \{0, y_i\}$.

We constrain each row i with the constraint SEQUENCE$(\mathcal{M}_i, 1, 2, 2, \{0\})$, stating that in each sequence of length 2, at least one 0 occurs. On columns 1 (resp. 3 and 5) we put the GCC constraint stating that each value in W (resp. Z and Y) occurs at least once, and that at least $m - q$ many 0's occur. On columns 2 and 4 we put the GCC constraint stating that at least q many t's occur, and at least $m - q$ many 0's.

We show that this instance of SEQUENCEGCC has a solution iff there exists a 3D matching.

($\Rightarrow$) Take a solution for SEQUENCEGCC. We know column 2 contains exactly $m - q$ many t's, and q many 0's. For each occurrence of a t in column 2 at row i, columns 1 and 3 contain a 0 at row i (by the SEQUENCE constraint). Then, by the GCC constraint, for all rows j where column 2 contains a 0, columns 1 and 3 contain a non-0 at row j, and thus (by SEQUENCE) column 4 contains a 0 at row j. By a similar argument, we know that in the remaining rows column 4 contains t's. Continuing this argument for column 5, we know that in the solution there are q many rows taking values $(w_i, 0, z_i, 0, y_i)$ and $m - q$ rows taking values $(0, t, 0, t, 0)$. By the GCC constraints, we know that each value $w \in W$ occurs exactly once, as well as each value $z \in Z$ and each $y \in Y$. Since the possible values were chosen by taking elements from M, we know that $M' = \{s_i \mid \mathcal{M}_i \neq (0, t, 0, t, 0)\}$ is a 3D matching.

($\Leftarrow$) Let $M' \subseteq M$ be a 3D matching. We can fill $\mathcal{M}$ as follows. For each $(w_i, z_i, y_i) = s_i \in M'$, we let $\mathcal{M}_i = (w_i, 0, z_i, 0, y_i)$. For each $s_i \in M \backslash M'$ we let $\mathcal{M}_i = (0, t, 0, t, 0)$. Obviously each row satisfies the SEQUENCE constraint. Since $|M'| = q$ and each value $w \in W$ occurs exactly once in the first coordinate of M' (and similarly for values $z \in Z$ and the second coordinate, and $y \in Y$ and the third coordinate), we have that each column satisfies the corresponding GCC constraint. □

Note that the SEQUENCE constraints on all rows are the same, but the GCC constraints on the columns differ.

Theorem 4. *Enforcing BC on* SequenceGcc *is* NP*-hard, already for just* 5 *columns.*

Proof: The proof is similar to the proof of Theorem 3, and also inspired by [6].

Let cw_i (resp. cz_i, cy_i) be the number of occurrences of the value w_i (resp. z_i, y_i) in M. For each value w_i (resp. z_i, y_i) we create $cw_i - 1$ (resp. $cz_i - 1$, $cy_i - 1$) clones of it. We now define the total order on values as $U = [-w_q^1, \ldots, -w_q^{cw_q-1}, \ldots, -w_1^1, \ldots, -w_1^{cw_1-1}, -z_q^1, \ldots, -z_q^{cz_q-1}, \ldots, -z_1^1, \ldots, -z_1^{cz_1-1}, -y_q^1, \ldots, -y_q^{cy_q-1}, \ldots, -y_1^1, \ldots, -y_1^{cy_1-1}, 0, t, y_1, \ldots, y_q, z_1, \ldots, z_q, w_1, \ldots, w_q]$.

We create the matrix $\mathcal{M}$ in a similar fashion as in the proof for Theorem 3, with the difference that for each $(w_i, z_i, y_i) = s_i$ we let $\mathcal{M}_{i,1} \in [-w_i^1, \ldots, -w_i^{cw_i-1}, \ldots, w_i]$, $\mathcal{M}_{i,3} \in [-z_i^1, \ldots, -z_i^{cz_i-1}, \ldots, z_i]$, $\mathcal{M}_{i,5} \in [-y_i^1, \ldots, -y_i^{cy_i-1}, \ldots, y_i]$, and $\mathcal{M}_{i,2} \in [0, t]$ and $\mathcal{M}_{i,4} \in [0, t]$.

We adapt the constraint on rows to Sequence$(\mathcal{M}_i, 1, 2, 2, [-w_q^1, 0])$, stating that in each sequence of length 2, at least one value in $[-w_q^1, 0]$ occurs. On columns 1 (resp. 3 and 5) we replace the Gcc constraint with one stating that each value in $\{-w_q^1, \ldots, -w_q^{cw_q-1}, \ldots, -w_1^1, \ldots, -w_1^{cw_1-1}\} \cup W$ (resp. $\{-z_q^1, \ldots, -z_q^{cz_q-1}, \ldots, -z_1^1, \ldots, -z_1^{cz_1-1}\} \cup Z$ and $\{-y_q^1, \ldots, -y_q^{cy_q-1}, \ldots, -y_1^1, \ldots, -y_1^{cy_1-1}\} \cup Y$) occurs at least once. We do not change the Gcc constraints on columns 2 and 4.

We show that this instance of SequenceGcc has a solution iff there exists a 3D matching.

($\Rightarrow$) By reasoning similarly to the proof of Theorem 3 (replacing '0' with 'a value in $[-w_q^1, 0]$' when reasoning about columns 1, 3 and 5), we know that in the solution there are q many rows taking values $(w_i, 0, z_i, 0, y_i)$, possibly containing clones, and $m - q$ rows taking values (n, t, n', t, n'') where n, n', n'' are either 0 or some clone w_i', z_i', or y_i' (respectively).

Furthermore, by the Gcc constraint on the odd columns, we know that each value in $\{-w_q^1, \ldots, -w_q^{cw_q-1}, w_q\}$ must occur exactly once. Since these values occur only in the domains of $\mathcal{M}_{i,1}$ for the cw_q many $s_i \in M$ that contain w_q, we know that each of these $\mathcal{M}_{i,1}$ must take a value in $\{-w_q^1, \ldots, -w_q^{cw_q-1}, w_q\}$.

Then, by the Gcc constraint on the odd columns, we know that each value in $\{-w_{q-1}^1, \ldots, -w_{q-1}^{cw_{q-1}-1}, w_{q-1}\}$ must occur exactly once. These values occur only in the domains of $\mathcal{M}_{i,1}$ for the $s_i \in M$ that contain w_{q-1} or w_q. However, the $\mathcal{M}_{i,1}$ for the $s_i \in M$ that contain w_q must take values in $\{-w_q^1, \ldots, -w_q^{cw_q-1}, w_q\}$. Therefore, the $\mathcal{M}_{i,1}$ for the $s_i \in M$ that contain w_{q-1} must take a value in $\{-w_{q-1}^1, \ldots, -w_{q-1}^{cw_{q-1}-1}, w_{q-1}\}$.

Repeating this argument recursively until reaching the value t, we can restrict the effective domain of the odd positions of $\mathcal{M}_i$ for $s_i = (w_i, z_i, y_i) \in M$ to $\mathcal{M}_{i,1} \in \{-w_i^1, \ldots, -w_i^{cw_i-1}, w_i\}$, $\mathcal{M}_{i,3} \in \{-z_i^1, \ldots, -z_i^{cz_i-1}, z_i\}$, and $\mathcal{M}_{i,5} \in \{-y_i^1, \ldots, -y_i^{cy_i-1}, y_i\}$.

Therefore, we know that each row $\mathcal{M}_i$ for $s_i = (w_i, z_i, y_i) \in M$ either has the values $(w_i, 0, z_i, 0, y_i)$ or the values (w_i', t, z_i', t, y_i') for some clones w_i', z_i', y_i' of

w_i, z_i, y_i. Now, by the Gcc constraints, we know that each value $w \in W$ occurs exactly once, as well as each value $z \in Z$ and each $y \in Y$. Since the possible values were chosen by taking elements from M, we know that $M' = \{s_i \mid \mathcal{M}_i = (w, 0, z, 0, y), w \in W, z \in Z, y \in Y\}$ is a 3D matching.

($\Leftarrow$) Let $M' \subseteq M$ be a 3D matching. We can fill $\mathcal{M}$ as follows. For each $(w_i, z_i, y_i) = s_i \in M'$, we let $\mathcal{M}_i = (w_i, 0, z_i, 0, y_i)$. For each $(w_i, z_i, y_i) = s_i \in M \backslash M'$ we let $\mathcal{M}_i = (w'_i, t, z'_i, t, y'_i)$ for some clones w'_i, z'_i, y'_i of w_i, z_i, y_i that have not been used before in the process of filling $\mathcal{M}$. We know there are enough different clones for this procedure. It is easy to verify that this instantiation of $\mathcal{M}$ satisfies all the constraints. □

3 Fixed parameter tractable cases

We have seen that propagating the RegularGcc matrix constraint is NP-hard even under the strong restriction that either the number of values or the number of columns is bounded. However, if we consider Regular^2 and we bound the number of columns and the number of states in the row and column automata we at last have a case in which propagation is polynomial.

Theorem 5. *Enforcing DC on* Regular^2 *is fixed parameter tractable in* $k = C \cdot |Q| \cdot (\log |Q'|)$*, where* C *is the number of columns,* $|Q|$ *is the size of the row automata, and* $|Q'|$ *is the size of the column automata.*

Proof: We assume w.l.o.g. that all row constraints are the same, and all column constraints are the same. We can encode the matrix constraint on a $R \times C$ matrix $\mathcal{M}$ in a single DFA on the matrix stretched out to a single sequence of variables $\mathcal{M}_{1,1}, \dots, \mathcal{M}_{1,C}, \dots, \mathcal{M}_{R,1}, \dots, \mathcal{M}_{R,C}$. The state set of the automaton is $Q \times Q'^{|C|}$. In each state, the automaton keeps track of the current state $q' \in Q'$ for each column c, as well as the current state $q \in Q$ in the current row. The size of the automaton is $\mathcal{O}(|Q| \cdot 2^{C(\log |Q|)})$. Enforcing DC on a Regular constraint takes time polynomial in the size of the automaton, so our algorithm runs in fixed parameter tractable time. □

We also get tractability if we bound the number of rows and the size of the automata.

Theorem 6. *Enforcing DC on* RegularGcc *is fixed parameter tractable in* $k = r(\log Q)$ *where* r *is the number of rows and* Q *the maximum number of states in any row automaton.*

Proof: This follows directly from Observation 2 in [4], and the fact that Gcc over a sequence with fixed size can be encoded in a DFA with polynomially many states. □

On the other hand, just bounding the number of rows is not enough to give tractability.

Theorem 7. *Enforcing BC on* RegularGcc *is W[2]-hard in* $k = R$ *the number of rows, even with just* 2 *values.*

Proof: This proof is similar to the proof of Theorem 3 in [4]. We reduce from p-Hitting-Set. Let $\mathcal{H} = (V, E)$ a hypergraph, where $V = \{v_1, \dots, v_{|V|}\}$ and $E = \{e_1, \dots, e_{|E|}\}$. We ask if there is a hitting set $S \subseteq V$ in $\mathcal{H}$ of cardinality k.

We construct an instance $\mathcal{M}$ of RegularGcc with $|V| + |E|$ columns and k rows on the alphabet $\{0, 1\}$. The Regular constraint accepts $|V|$ different words $w^1, \dots, w^{|V|}$ of length $|V| + |E|$. For any word w^v, the vth value is 1, and the remainder of the first $|V|$ values is 0. Also, for any word w^v and any $1 \leq j \leq |E|$, the jth value of w^v is 1 if $v \in e_j$, and is 0 otherwise. The Gcc constraints we put on the columns are as follows. In the first $|V|$ columns we require exactly one 1. In the remaining columns, we require at least one 1.

In this reduction, each row corresponds to one vertex that is being chosen for inclusion in the hitting set, and each column after the first $|V|$ to one hyperedge.

The Gcc constraints on the first $|V|$ columns ensure that no vertex is chosen in two rows. The constraints on the last $|E|$ columns ensure that each hyperedge contains a vertex chosen for the hitting set. If there is a hyperedge all vertices of which are not included in the hitting set, the column corresponding to this hyperedge will contain 0's only, violating the Gcc constraint of that column.

We show that there exists a hitting set S in $\mathcal{H}$ of cardinality k iff the RegularGcc matrix constraint has a solution.

($\Rightarrow$) Assume there exists a hitting set S in $\mathcal{H}$ of size k. We construct an assignment to RegularGcc by matching one vertex $v \in S$ with each row (in any manner). If row i is matched to vertex v, we assign word w^v to row i. The fact that S contains k different vertices v ensures that k different words w^v are used. Also, since S is a hitting set, we know that for each hyperedge e_j there is at least one $v \in e_j \cap S$, and so each of the last $|E|$ columns contains at least one 1. Thus the Gcc constraints are satisfied.

($\Leftarrow$) Suppose the RegularGcc matrix constraint has a solution. We construct a hitting set S by taking all v such that w^v is a row in the solution. By the Gcc constraints we know that the solution contains k different words w^v, so S is of size k. Now, to derive the contrary, assume there exists a hyperedge $e_j \in E$ such that $S \cap e_j = \emptyset$. Then the column corresponding to e_j (column $|V| + j$) contains only 0's. This violates Gcc on this column, which is a contradiction. □

Note that the Gcc constraints in the above proof can be expressed with Regular constraints of bounded size as well, which gives us the following corollary.

Corollary 2. *Enforcing BC on the* Regular^2 *matrix constraint is W[2]-hard in k the number of rows, even with just* 2 *values.*

Another special case that is intractable is when we repace the Gcc constraint on the columns with a simpler sum constraint.

Theorem 8. *Enforcing BC on the* RegularSum *matrix constraint is W[2]-hard in k the number of rows, even with just* 3 *values.*

Proof: (Sketch) The proof is similar to the proof of Theorem 7 and the proof of Theorem 3 in [4]. We reduce from p-Hitting-Set and construct a matrix constraint as in the proof of Theorem 7, with the following differences. The first $|V|$ columns we fill with -1's instead of 1's. We can then replace the Gcc constraints on these columns with the Sum constraint requiring a sum of at least -1. The Gcc constraints on the last $|E|$ columns can be replaced with the Sum constraint requiring a sum of at least 1. The arguments in the proof of Theorem 7 now hold for this instance of RegularSum. $\square$

Note that this result is strictly stronger than the W[1]-hardness proof of enforcing BC on RegularSum in [4].

4 Some necessary conditions

Motivated by these rather negative complexity results, we investigate how to improve propagation over a simple decomposition into separate Regular and Gcc constraints by means of deriving necessary conditions based on string properties. In fact, we will show how to extend the method of [5] to the (decomposed) setting of multicostRegular constraints on the rows and Gcc constraints on the columns. This method is based on a double counting argument. Using automata constraints we extract several string properties from the rows. For these string properties, we derive lower and upper bounds based on the Gcc constraints on the columns. This allows us to derive necessary constraints relating the bounds to the corresponding string properties.

We start with some preliminary definitions needed for our exposition. The multicostRegular global constraint [7] is defined as follows. Given a sequence $X = (x_1, x_2, \dots, x_n)$ of finite domain decision variables and a deterministic finite automaton $\mathcal{A} = (Q, V, \Delta, s, F)$, the constraint Regular$(X, \mathcal{A})$ holds iff X is a word of length n over V accepted by DFA $\mathcal{A}$. Given a vector $Z = (z^0, \dots, z^R)$ of bounded variables and $c = (c^r_{q,v})^{r \in [0 \dots R]}_{q \in Q, v \in V}$ a family of assignment cost matrices, multicostRegular$(X, Z, \mathcal{A}, c)$ holds iff Regular$(X, \mathcal{A})$ holds and for an accepting run $q_0 q_1 \dots q_n$ of $\mathcal{A}$ on the instantiation $(v_0, \dots, v_n)$ of X we have that $\sum_{0 \leq i < n} c^r_{q_i, v_{i+1}} = z^r$ for all $0 \leq r \leq R$.

For any two DFAs $\mathcal{A}_1 = (Q_1, V, \Delta_1, s_1, F_1)$ and $\mathcal{A}_2 = (Q_2, V, \Delta_2, s_2, F_2)$, with corresponding c^1 and c^2 cost matrices over resources $\mathcal{R} = \{r^0, \dots, r^R\}$, we define the product automaton $\mathcal{A}_1 \times \mathcal{A}_2 = (Q_1 \times Q_2, V, \Delta, (s_1, s_2), F_1 \times F_2)$ and product cost matrix $c = c^1 \times c^2$ as follows.

$$\Delta((q_1, q_2), v) = (\Delta_1(q_1, v), \Delta_2(q_2, v))$$

$$c^r_{(q_1,q_2),v} = c^{1,r}_{q_1,v} + c^{2,r}_{q_2,v} \quad \text{for } 0 \leq r \leq R$$

In other words, when taking the product of two weighted automata, we take the usual cross product of the underlying automata, and add the cost matrices.

We show how to extract relevant string properties using multicostRegular constraints on the rows. In the following, we let $v \in V$ denote a value that the decision variables can take, we let $\hat{v}, \hat{w}_i \subseteq V$ (for indices $i \in \mathbb{N}$) denote a subset of these values, we let $\neg\hat{v}$ denote $V \backslash \hat{v}$, and we let Z be a set of bounded variables representing the calculated weights. We also define the concatenation $\hat{w}_1 \cdot \ldots \cdot \hat{w}_m$ of several $\hat{w}_i$ as the set $\{w_1 \cdots w_m \in V^m \mid w_i \in \hat{w}_i, 1 \leq i \leq m\}$.

To extract the number of uninterrupted stretches of elements from $\hat{v}$ in X using a resource variable $z^r \in Z$, we can use the weighted DFA $\mathcal{A}_1^{\hat{v}}$ (Figure 2), where transitions are marked with the symbol and the cost c^r they consume. For any word X, we have that $\textsc{multicostRegular}(X, Z, \mathcal{A}_1^{\hat{v}}, c)$ holds for z^r the number of stretches of symbols in $\hat{v}$ that occur in X.

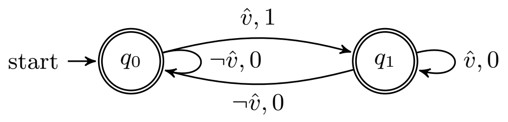

Fig. 2. Automaton $\mathcal{A}_1^{\hat{v}}$. Transitions are marked with cost c^r.

To extract whether a word $\overline{w} \in \overline{\hat{w}}$ occurs in X starting at position k using a resource variable z_k^r, we can use the weighted DFA $\mathcal{A}_2^{k,\overline{\hat{w}}}$ (Figure 3) with parameter $k \in \mathbb{N}$, where transitions are marked with the symbol and the cost c^r they consume. For any word X, we have that $\textsc{multicostRegular}(X, Z, \mathcal{A}_2^{k,\overline{\hat{w}}}, c)$ sets z_k^r to true if and only if some word $\overline{w} \in \overline{\hat{w}}$ occurs in X starting at position k. To extract the total number of occurrences of words $\overline{w} \in \overline{\hat{w}}$ (starting any position) in X, we take the sum of the values of the variables z_k^r (for $1 \leq k \leq n$) that represent whether a suitable word $\overline{w}$ occurs in X starting at position k.

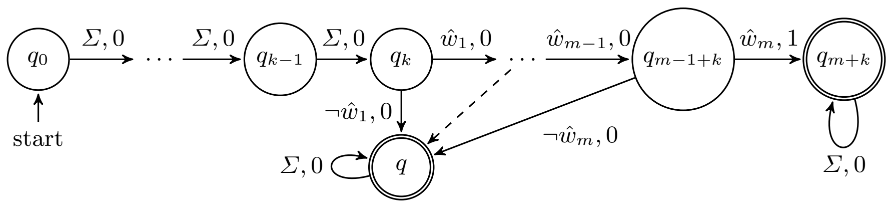

Fig. 3. Automaton $\mathcal{A}_2^{k,\overline{\hat{w}}}$. Transitions are marked with cost c^r.

To extract the minimum and maximum length of stretches, we can simulate counters using weights. Let $\mathcal{A}$ be a DFA annotated with counters $d = (d_1, \ldots, d_m)$, taking their values from $\{0, \ldots, n-1\}$. We can construct a DFA $\mathcal{A}'$ of size less than or equal to $n^m \cdot |\mathcal{A}|$, together with a cost matrix c for resources $r^1, \ldots, r^m$ such that for any word w there exists an accepting run for w on $\mathcal{A}$ where the counters have final values $(v_1, \ldots, v_m)$ if and only if there exists an accepting run for w on $\mathcal{A}'$ for $(z^1, \ldots, z^m) = (v_1, \ldots, v_m)$. This can be done straightforwardly by choosing $Q \times \{0, \ldots, n-1\}^m$ as state set for $\mathcal{A}'$, and choosing transitions Δ corresponding to the update formulae for the counters. Now c can be chosen to mimic the changes in counter values over transitions. The automaton $\mathcal{A}'$ can possibly be reduced in size by removing unreachable states or minimizing it using other methods.

We can transform any given automaton $\mathcal{A}$ to extract the minimum and maximum length of a stretch of symbols from $\hat{v}$ occurring in $\mathcal{A}$ on X as follows.

We annotate $\mathcal{A}$ with counters that represent *stretchminlen*$(\hat{v}, n)$ and *stretchmaxlen*$(\hat{v}, n)$, as described in [5]. Then we transform this annotated automaton, as described above, into an automaton $\mathcal{A}'$ with resource variables $z^{\hat{v}}_{min}$ and $z^{\hat{v}}_{max}$ whose values (respectively) represent the minimum and maximum length of stretches of symbols in $\hat{v}$ occurring in X.

The above automata, extracting the different string properties from rows, can be combined with each other and with other automata by using the product operation. By defining zero cost matrices for all resources not used explicitly in a given automaton, we can extract several different string properties simultaneously with one weighted product automaton.

In this more general decomposed setting with multicostRegular constraints on the rows, a tractable option for propagation is the algorithm based on a Lagrangian relaxation of the Resource Constrained Shortest Path Problem (RCSPP) from [7]. Using weighted automata to extract string properties has several advantages. The size of the automata is relatively low. The automata used to extract the number of stretch occurrences are even of constant size. A weighted automaton used to extract a string property is never larger than the unfolding of an (unweighted) automaton annotated with counters used to extract the same string property. Also, the use of weighted automata allows us to express several other constraints with small automata. For instance, Gcc constraints on the rows can be expressed by a weighted automaton with a single state. In fact, Gcc constraints can be expressed using additional weights on other automaton constraints already posed on the rows.

Using the above string properties, we can derive necessary conditions that exploit the matrix structure. Consider the following CSP, similar to the one sketched in [5]. Given three positive integers R, K, and V, we have an $R \times K$ matrix $\mathcal{M}$ of decision variables with domain $\{0, 1, \ldots, V-1\}$, and a $V \times K$ matrix $\mathcal{M}^{\#}$ of cardinality variables with domain $\{0, 1, \ldots, R\}$. Each row r, for $0 \leq r < R$, of $\mathcal{M}$ is subject to a multicostRegular constraint. For simplicity, we assume that each row is subject to the same constraint. Each column k, for $0 \leq k < K$, of $\mathcal{M}$ is subject to a Gcc constraint that restricts the number of occurrences of the values according to column k of $\mathcal{M}^{\#}$. Let $\#^v_k$ denote the number of occurrences of value v, for $0 \leq v < V$, in column k of $\mathcal{M}$, that is, the cardinality variable in row v and column k of $\mathcal{M}^{\#}$. For any $\hat{v} \subseteq V$, we let $\#^{\hat{v}}_k$ denote $\sum_{v \in \hat{v}} (\#^v_k)$.

In order to constrain the number of occurrences of words, we define the bounds $lw_k(\overline{\hat{w}})$ and $uw_k(\overline{\hat{w}})$ on the number of occurrences of words in $\overline{\hat{w}}$ starting at column k, based on the Gcc constraints on the columns, as follows:

$$lw_k(\overline{\hat{w}}) = \max\left(\left(\sum_{j=0}^{|\overline{\hat{w}}|-1} \#^{\hat{w}_j}_{k+j}\right) - (|\overline{\hat{w}}| - 1) \cdot R, 0\right) \tag{1}$$

$$uw_k(\overline{\hat{w}}) = \min_{j=0}^{|\overline{\hat{w}}|-1} \left(\#^{\hat{w}_j}_{k+j}\right) \tag{2}$$

Note that definitions (1) and (2) are exactly the same as in [5]. The lower bound (1) is the worst-case intersection of all column value occurrences. The upper bound (2) is justified by the fact that a word cannot occur more often than its minimally occurring letter. We now get the following necessary conditions for each $0 \leq k < K$:

$$lw_k(\overline{\hat{w}}) \leq \sum_{r=0}^{R-1} z_{r,k}^{\overline{\hat{w}}} \qquad (3) \qquad\qquad uw_k(\overline{\hat{w}}) \geq \sum_{r=0}^{R-1} z_{r,k}^{\overline{\hat{w}}} \qquad (4)$$

where $z_{r,k}^{\overline{\hat{w}}}$ denotes the resource variable representing whether a word in $\overline{\hat{w}}$ occurs in row r starting at column k. Since we extracted the number of word occurrences for each starting position k, we can directly relate the bounds derived from the column constraints with the number of word occurrences per starting position. This results in constraints (3) and (4) potentially leading to more propagation than their counterparts in [5]. This is illustrated in Example 1. Note that the constraints from [5] on words occurring as a prefix or as a suffix correspond to particular cases of constraints (3) and (4).

Example 1. Consider the scenario concerning a partially instantiated 5×5 matrix in Figure 4, which could occur as a node in the search tree. Let $\overline{\hat{w}} = \{2\}\{2\}$. In this scenario $lw_k(\overline{\hat{w}})$ are variables. Also, $z_r^{\overline{\hat{w}}}$ is a variable such that $z_r^{\overline{\hat{w}}} = \sum_{k=0}^{K-1} z_{r,k}^{\overline{\hat{w}}}$. In this scenario, the bounds of the variables $z_{r,k}^{\overline{\hat{w}}}$ can be automatically derived by the row automata. By using equation (3), we can directly detect unsatisfiability in this case, since $lw_1(\overline{\hat{w}}) \in [3,5]$ and thus $lw_1(\overline{\hat{w}}) \not\leq (\sum_{r=0}^{R-1} z_{r,1}^{\overline{\hat{w}}}) \in [0,2]$. Consider the counterpart of equation (3) from [5]: $\sum_{k=0}^{K-|\overline{\hat{w}}|} lw_k(\overline{\hat{w}}) \leq \sum_{r=0}^{R-1} z_r^{\overline{\hat{w}}}$. Using this constraint, unsatisfiability cannot directly be detected in this particular case.

Fig. 4. Example search tree node.

$$\mathcal{M}: \begin{array}{||c|c|c|c|c||} \hline & & & & \\ \hline & & & & \\ \hline 1 & & 1 & & \\ \hline 1 & & 1 & & \\ \hline 1 & & 1 & & \\ \hline \end{array}$$

$lw_0(\overline{\hat{w}}) \in [0,2]$	$z_{0,1}^{\overline{\hat{w}}} \in [0,1]$	$z_0^{\overline{\hat{w}}} \in [0,1]$
$lw_1(\overline{\hat{w}}) \in [3,5]$	$z_{1,1}^{\overline{\hat{w}}} \in [0,1]$	$z_1^{\overline{\hat{w}}} \in [0,1]$
$lw_2(\overline{\hat{w}}) \in [0,2]$	$z_{2,1}^{\overline{\hat{w}}} = 0$	$z_2^{\overline{\hat{w}}} \in [0,1]$
$lw_3(\overline{\hat{w}}) \in [0,5]$	$z_{3,1}^{\overline{\hat{w}}} = 0$	$z_3^{\overline{\hat{w}}} \in [0,1]$
$lw_4(\overline{\hat{w}}) = 0$	$z_{4,1}^{\overline{\hat{w}}} = 0$	$z_4^{\overline{\hat{w}}} \in [0,1]$

Take note of the following case, where $\overline{\hat{w}} = \hat{v}$ for some $\hat{v} \subseteq V$. In this case, for each $0 \leq k < K$ the constraints (1) and (2) and the constraints (3) and (4) simplify to, respectively:

$$lw_k(\hat{v}) = uw_k(\hat{v}) = \#_k^{\hat{v}} \qquad (5) \qquad\qquad \#_k^{\hat{v}} = \sum_{r=0}^{R-1} z_{r,k}^{\hat{v}} \qquad (6)$$

In order to constrain the number of occurrences of stretches, we define the bounds ls_k^+ and us_k^+ (referring to the number of uninterrupted stretches of variables from $\hat{v}$ that start in column k) and the bounds ls_k^- and us_k^- (referring to the number of uninterrupted stretches of variables from $\hat{v}$ that end in column k), based on the GCC constrains as follows:

$$ls_k^+ = \max(0, \#_k^{\hat{v}} - \#_{k-1}^{\hat{v}}) \tag{7}$$
$$us_k^+ = \#_k^{\hat{v}} - \max(0, \#_{k-1}^{\hat{v}} + \#_k^{\hat{v}} - R) \tag{8}$$
$$ls_k^- = \max(0, \#_k^{\hat{v}} - \#_{k+1}^{\hat{v}}) \tag{9}$$
$$us_k^- = \#_k^{\hat{v}} - \max(0, \#_{k+1}^{\hat{v}} + \#_k^{\hat{v}} - R) \tag{10}$$

Definitions (7) through (10) are exactly the same as in [5]. The lower bound (7) is the difference between the number of occurrences of values $\hat{v}$ in column k minus the number of occurrences of $\hat{v}$ in column $k-1$, if positive. If the total number of occurrences of values $\hat{v}$ on column k and on column $k-1$ are strictly greater than the number of rows R, then there must be at least $\#_{k-1}^{\hat{v}} + \#_k^{\hat{v}} - R$ stretches of values $\hat{v}$ that cover both columns. This minimum intersection gives us the upper bound (8). Bounds (9) and (10) are derived similarly. We now get the following necessary conditions:

$$\sum_{k=0}^{K-1} ls_k^+(\hat{v}) \leq \sum_{r=0}^{R-1} z_r^{\hat{v}} \tag{11}$$
$$\sum_{k=0}^{K-1} us_k^+(\hat{v}) \geq \sum_{r=0}^{R-1} z_r^{\hat{v}} \tag{12}$$
$$\sum_{k=0}^{K-1} ls_k^-(\hat{v}) \leq \sum_{r=0}^{R-1} z_r^{\hat{v}} \tag{13}$$
$$\sum_{k=0}^{K-1} us_k^-(\hat{v}) \geq \sum_{r=0}^{R-1} z_r^{\hat{v}} \tag{14}$$

where $z_r^{\hat{v}}$ denotes the variable corresponding to the resource that represents the number of uninterrupted sequences of symbols in $\hat{v}$ occurring in row r.

In order to constrain the minimum and maximum length of a stretch, using the minimum and maximum length ($z_{min}^{\hat{v}}$ and $z_{max}^{\hat{v}}$, respectively) of uninterrupted sequences of symbols in $\hat{v}$ occurring in any row, we get the following necessary conditions for each $0 \leq k < K$:

$$\#_k^{\hat{v}} \geq \sum_{j=\max(0,k-z_{min}^{\hat{v}}+1)}^{k} ls_j^+(\hat{v}) \tag{15}$$
$$\#_k^{\hat{v}} \geq \sum_{j=k}^{\min(K-1,k+z_{min}^{\hat{v}}-1)} ls_j^-(\hat{v}) \tag{16}$$

Constraints (15) and (16) are justified by the fact that stretches starting resp. ending at the considered columns j must overlap column k. Also, for each $0 \leq k < K - z_{max}^{\hat{v}}$ we get the necessary condition:

$$ls_k^+(\hat{v}) + \sum_{j=z_{min}^{\hat{v}}}^{z_{max}^{\hat{v}}} \#_{k+j}^{\hat{v}} - (z_{max}^{\hat{v}} - z_{min}^{\hat{v}} + 1) \cdot R \leq 0 \tag{17}$$

and for each $z_{max}^{\hat{v}} \leq k < K$ the necessary condition:

$$ls_k^-(\hat{v}) + \sum_{j=z_{min}^{\hat{v}}}^{z_{max}^{\hat{v}}} \#_{k-j}^{\hat{v}} - (z_{max}^{\hat{v}} - z_{min}^{\hat{v}} + 1) \cdot R \leq 0 \tag{18}$$

The justification behind constraint (17) is that for a stretch of values $\hat{v}$ beginning at column k, there must be a value not in $\hat{v}$ in some column j, for $k + z^{\hat{v}}_{min} \leq j \leq k + z^{\hat{v}}_{max}$. Constraint (18) is justified similarly.

5 Evaluation

To evaluate our method, we used NSPLib [8], a library of benchmark instances of the nurse scheduling problem (NSP). This is a particular rostering problem. For N the number of nurses, D the number of days in the scheduling horizon, and S the number of shifts, the objective is to construct a $N \times D$ matrix of values in the integer interval $[1, S]$, where value S represents the off-duty shift.

In instance files, there are hard coverage constraints and soft preference constraints. We only consider the hard coverage constraints. These give for each day d and shift s the lower bound on the number of nurses that must be assigned to shift s on day d. These constraints can be modelled by Gcc constraints on the columns. We considered instance files for $N \times 7$ rosters with $N \in \{25, 50, 75, 100\}$.

Case files provide hard constraints on the rows. For each shift s, there are lower and upper bounds on the number of occurrences of s in any row. There are also lower and upper bounds on the cumulative number of occurrences of the working shifts $1, \ldots, S-1$ in any row. These two types of constraints can be modelled by Gcc constraints on the rows. For each shift s, there are also lower and upper bounds on the length of any stretch of value s in any row. Finally, there are lower and upper bounds on the length of any stretch of working shifts $1, \ldots, S-1$ in any row. These two types of constraints can be modelled by stretch_path and stretch_path_partition constraints on the rows, respectively. By translating these row constraints to automata, we get that the NSP benchmark problems as described above correspond to the RegularGcc pattern studied in this paper.

In order to compare the effect of the necessary conditions in the settings of both weighted and unweighted automata, we implemented the row constraints (both for the constraints from the case files and for extracting string properties) using weighted finite automata as well as regular (unweighted) finite automata. For the setting of unweighted automata, we translated the case constraints specified for each shift and for the total set of working shifts as a single Regular constraint on each row (by taking the corresponding minimised product DFA). For each string property that we extract from the rows, we used automata annotated with counters (as described in [5]), unfolded into a DFA, expressed as a decomposition into ternary constraints [9] allowing us to extract the counter values. The methods used in [5] for automata annotated with counters are not implemented in the free major constraint programming libraries and solvers.

For the setting of weighted automata. We translated the case constraints for each shift and for the total set of working shifts as a single multicostRegular constraint on each row (by taking the corresponding product automaton). For each string property that we extract from the rows, we posed a single multicostRegular constraint defined by the corresponding weighted automaton as described in Section 4.

In order to compare the two settings fairly, we posed the constraints defined by automata in a similar pattern, i.e., we take the products of corresponding automata in the two settings. One advantage that the setting of weighted automata possesses, is that taking the product of particular automata results in a relatively small increase in the automaton size, not nearly as explosive as the size increase in the corresponding unweighted product automaton. In order to improve propagation, we were able to pose the weighted automata extracting the number of stretches of different shifts from the rows as the product of the corresponding automaton with (a copy of) the automata specifying the constraints on the number of shift occurrences from the case file. In the unweighted setting this is completely intractable, since the size of the product DFA corresponding to the automata annotated with counters gets too large.

In both settings, we implemented necessary constraints based on the following string properties:

– for each shift, lower and upper bounds on the number of its occurrences,
– for each shift, lower and upper bounds on the number and length of its stretches,
– each word of length at most 2 that consists of one single shift.

In the setting of weighted automata, the necessary constraints are derived as described in Section 4. In the setting of unweighted automata, the necessary constraints are derived as in [5].

The objective of our experiments is to measure the impact in runtime and backtracks for the different settings. The experiments were run under Choco 2.1.1 on a 2.27 GHz Intel Xeon with a 4GB RAM. All runs were allocated 3 CPU minutes. For each case and nurse count N, we used instances 1-270.

In the experiments we used a labelling procedure that selects variables with the smallest domain, with a row-wise order as tie-breaker, and selects the smallest value. We used a LexChain constraint for symmetry breaking. We used the implementation of the multicostRegular constraint available in Choco.

Table 1 summarises the running of the instances for the different settings (the setting of weighted automata with cross products (CWA) and without extra cross products (WA), and the setting of unweighted automata (UA)), for Cases 7 and 8. Each row first indicates the number of known instances of some satisfiability status for a given case and nurse count N, and then the performance of each setting to the first solution, namely the number of instances decided to be of that status without timing out, as well as the average runtime (in seconds) and the average number of backtracks for all instances on which none of the settings timed out. Numbers in boldface indicate best performance in a row.

The benchmark results in Table 1 show that WA and CWA were able to solve significantly more instances compared to the method using unweighted automata, both for satisfiable and unsatisfiable instances. Further, CWA improved the performance for most of the benchmarks in terms of number of backtracks and runtime, compared to WA. Notably, the UA method solved only 4 out of 156 known unsatisfiable instances while CWA and UA solved all of these benchmarks. This shows that using weighted automata together with necessary constraints leads to significantly more pruning than using unweighted automata with similar necessary constraints. For the majority of solved unsatisfiable instances, WA and CWA detected unsatisfiability at the root of the search tree. This is not visible in the table, because the shown runtimes and number of backtracks are based on instances solved by all methods. Note that these benchmarking results are not directly comparable to the results in [5], since these results were obtained under a different experimental set-up (e.g. a different search strategy was used).

Overall, the results indicate that the use of weighted automata to solve rostering problems shows potential. A combination of weighted automata and necessary constraints dramatically increase propagation compared to using unweighted automata. Our results on unsatisfiable instances suggest that such a combination can be very useful in finding optimum solutions for rostering problems. Another advantage of our approach is that it can be easily implemented in open-source constraint solvers.

Table 1. NSPLib benchmark results.

				WA			CWA			UA		
Case	N	Status	Known	#Inst	Time	#Bktk	#Inst	Time	#Bktk	#Inst	Time	#Bktk
7	25	sat	129	122	23.8	1866	**123**	21.9	**1509**	103	**21.1**	2400
		unsat	30	**30**	0	0	**30**	0	0	0	0	0
7	50	sat	60	58	**16.6**	**693**	**60**	19.5	708	34	20.0	1227
		unsat	31	**31**	**0.1**	**0**	**31**	0.3	**0**	1	0.2	**0**
7	75	sat	29	25	**22.0**	742	**27**	25.6	**737**	17	22.1	929
		unsat	38	**38**	0	0	**38**	0	0	0	0	0
7	100	sat	34	29	30.9	1733	**34**	**29.8**	**1437**	13	38.5	2196
		unsat	19	**19**	0.2	**0**	**19**	**0.2**	**0**	1	0.3	**0**
8	25	sat	138	131	11.5	776	**133**	10.6	**646**	114	**9.1**	1123
		unsat	6	**6**	0	0	**6**	0	0	0	0	0
8	50	sat	90	83	13.1	606	**88**	**9.4**	**294**	71	15.0	1512
		unsat	8	**8**	**0.1**	**0**	**8**	0.1	**0**	1	0.2	**0**
8	75	sat	61	58	13.3	412	**62**	**10.4**	**233**	45	12.6	505
		unsat	19	**19**	0	0	**19**	0	0	0	0	0
8	100	sat	65	60	17.9	308	**65**	**13.4**	**143**	45	16.3	439
		unsat	5	**5**	**0.1**	**0**	**5**	0.1	**0**	1	0.3	**0**

6 Conclusions

We studied the propagation of the RegularGcc matrix constraint. We showed that propagation is NP-hard, even under some strong restrictions, and also showed two cases in which propagation is fixed parameter tractable. Additionally, we showed how to improve propagation over a decomposition into separate Regular constraints on the rows and Gcc constraints on the columns by identifying some necessary but insufficient conditions. We showed how the use of weighted automata for the row constraints can be beneficial. Experimental results on nurse scheduling problems demonstrate the potential for this method.

References

1. Flener, P., Frisch, A.M., Kzlltan, B.H.Z., Miguel, I., Walsh, T.: Matrix modelling. In: Proceedings of the International Workshop on Modelling and Problem Formulation. (2001) 27–41
2. Flener, P., Frisch, A.M., Kzlltan, B.H.Z., Miguel, I., Walsh, T.: Matrix modelling: Exploiting common patterns in constraint programming. In: Proceedings of the International Workshop on Reformulating Constraint Satisfaction Problems. (2002) 27–41
3. Régin, J.C., Gomes, C.P.: The Cardinality Matrix Constraint. In Wallace, M., ed.: Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP'04). Volume 3258 of Lecture Notes in Computer Science., Springer (2004) 572–587
4. George Katsirelos, Nina Narodytska, C.G.Q., Walsh, T.: Global matrix constraints. In: Proceedings of the International Workshop on Constraint Modelling and Reformulation. (2011) 27–41
5. Beldiceanu, N., Carlsson, M., Flener, P., Pearson, J.: On matrices, automata, and double counting. In Lodi, A., Milano, M., Toth, P., eds.: Proceedings of the 7th International Conference on Integration of AI and OR Techniques in Constaint Programming for Combinatorial Optimization Problems (CPAIOR'10). Volume 6140 of Lecture Notes in Computer Science., Springer (2010) 10–24
6. Crama, Y., Spieksma, F.: Scheduling jobs of equal length: Complexity, facets and computational results. In: Integer Programming and Combinatorial Optimization, 4th International IPCO Conference. Volume 920., Springer 277–291
7. Menana, J., Demassey, S.: Sequencing and Counting with the multicost-regular Constraint. In van Hoeve, W.J., Hooker, J.N., eds.: Proceedings of the 6th International Conference on Integration of AI and OR Techniques in Constaint Programming for Combinatorial Optimization Problems (CPAIOR'09). Volume 5547 of Lecture Notes in Computer Science., Springer (2009) 178–192
8. Vanhoucke, M., Maenhout, B.: On the characterization and generation of nurse scheduling problem instances. European Journal of Operational Research **196**(2) (2009) 457–467
9. Quimper, C.G., Walsh, T.: Global grammar constraints. In Benhamou, F., ed.: Proceedings of the 12th International Conference on Principles and Practice of Constraint Programming (CP'06). Volume 4204., Springer (2006) 751–755