The Regularization of CSPs for Rostering, Planning and Resource Management Problems

Sven Löffler, Ke Liu, Petra Hofstedt

The Regularization of CSPs For Rostering, Planning and Resource Management Problems

Sven Löffler, Ke Liu, and Petra Hofstedt

Brandenburg University of Technology Cottbus-Senftenberg, Germany
Sven.Loeffler@b-tu.de

Abstract. This paper presents a new approach to solve rostering, planning and resource management problems. This is achieved by transforming several kinds of finite domain constraints of a given constraint satisfaction problem (CSP) into a set of regular membership constraints; and then these regular membership constraints are combined together to a more specific regular membership constraint.
The purpose of this approach is to improve the speed of CSPs resolution and to remove undesirable redundant constraints (constraints which slow down the resolution speed) by replacing part of or all constraints of a CSP with a set of regular membership constraints followed by the combination of multiple regular membership constraints into a new, more precise regular membership constraint.
A concise rostering example has demonstrated that our approach enables a significant improvement of the performance of the CSP resolution due to the pruning of the search tree.

Keywords: Constraint Programming, CSP, Refinement, Planning, Resource Management, Scheduling.

1 Introduction

Constraint programming is a powerful method to model and solve NP-complete problems in a declarative way. Typical problems in constraint programming are planning, scheduling, resource management, graph coloring and satisfiability (SAT) problems [10].

Mostly, a CSP in practical can be described in various ways; and consequently, the problem can be modeled by different combinations of constraints, which results in the diversity of resolution speed and behavior. This phenomenon mainly caused by different propagators used by different constraints. Hence, the diversity of models and constraints for a given CSP offers us an opportunity to improve the problem solving by using another model in which a certain type of constraints can be replaced by a faster alternative or combined together to form more specific constraints.

This paper presents a way to model planning, scheduling and resource management problems using a regular CSP and discusses the use of the transformation from a CSP to a regular CSP. Because of the size of such planning, scheduling and resource

management problems solving them is mostly very time consuming. The goal is to improve the resolution speed of CSPs, remove redundancy, strengthen propagation, and avoid unnecessary backtracks (or failed backtracks). We reach this goal by replacing some or all constraints of a CSP with regular membership constraints followed by the combination of such created multiple regular membership constraints into a new, more precise regular membership constraint. The regular membership constraint (in the following: regular constraint) and its propagation algorithm [12, 11, 7] provide the basis of this approach. In [9] was shown that this approach can improve the solving speed of a CSP by reducing the number of backtracks and fails. In this paper we focus more on the question how we can transform special constraints (especially the count constraint) into regular constraints.

This paper is structured as follows: In Sect. 2 the basics of constraint satisfaction problems (CSPs) are explained and the notion of a regular CSP is presented. In Sect. 3 we show that every CSP can be transformed into a regular CSP, theoretically, and we present a selection of effective constraint transformations for practical use. Sect. 4 is dedicated to a discussion of the advantages and disadvantages and shows an example of this new approach. Finally, we give an outlook on the directions of future research in Sect. 5.

2 Basic Principles of Constraint Programming

This paper will consider constraint satisfaction problems (CSPs) which are defined as follows.

Definition 1: CSP. [2] A constraint satisfaction problem (CSP) is defined as a 3-tuple $P = (X, D, C)$ with $X = \{x_1, x_2, ..., x_n\}$ is a set of variables, $D = \{D_1, D_2, ..., D_n\}$ is a set of finite domains where D_i is the domain of x_i, $C = \{c_1, c_2, ..., c_m\}$ is a set of primitive or global constraints on the subsets of X.

We distinguish primitive and global constraints. Primitive constraints are simple relations like $x_1 = x_2$, $x_1 \neq x_2$ or $x_1 + x_2 < x_3$. Global constraints [15] are more complex and mostly employed with efficient dedicated propagation algorithms to solve a problem faster than counterpart primitive constraints. Examples of global constraints are the *allDifferent*, *globalCardinality*, *cumulative*, *count*, *sum* and *regular* constraints, described in [14].

Definition 2: Global constraint. [14] A global constraint is a restriction which describes a relationship between a non-fixed number of variables.

Furthermore, we define a solution of a CSP, local consistency and global consistency.

Definition 3: Solution. A value valuation $\sigma = d_1 \times \cdots \times d_n$, $\forall i \in \{1, ..., n\} : d_i \in D_i$ of the variables $X = \{x_1, x_2, ..., x_n\}$ in which the value d_i is assigned to the variable x_i, is a solution of P if all constraints C are satisfied.

Definition 4: Local consistency (Hyper-arc consistency). [1] A constraint $c \in C$

where $c \subseteq D_1 \times \cdots \times D_k$ is locally consistent if $\forall i \in \{1, \ldots, k\}$, $\forall d \in D_i$,$\exists (d_1, \ldots, d_{i-1}, d_{i+1}, \ldots, d_k) \in D_1 \times \cdots \times D_{i-1} \times D_{i+1} \times \cdots \times D_k$ such that $(d_1, \ldots, d_{i-1}, d, d_{i+1}, \ldots, d_k)$ satisfies c.

Definition 5: Global consistency. [2] Let $P = (X, D, C)$ be a CSP with the variables $X = \{x_1, \ldots, x_n\}$, the domains $D = (D_1, \ldots, D_n)$ and the set of constraints $C = \{c_1, \ldots, c_k\}$ over X. The CSP P is globally consistent, if $\forall i \in \{1, \ldots, n\}$, $\forall d_i \in D_i : \exists d_1 \in D_1, \ldots, \exists d_{i-1} \in D_{i-1}, \exists d_{i+1} \in D_{i+1}, \ldots, \exists d_n \in D_n$ such that the valuation σ with $\sigma(x_k) = d_k$, $k \in \{1, \ldots, n\}$ is a solution for P, i.e. satisfies the conjunction of all constraints in C.

To find a solution of a CSP, a certain level of consistency-enforcing algorithm is interleaved with backtrack search. The consistency-enforcing algorithm (e.g., local or global consistency) removes the illegal values that cannot occur in any solutions of the CSP in the domain of the variables.

It is important to differentiate between local consistency and global consistency in this paper. Local consistency guarantees that each value of a variable in the scope of the constraint is at least part of one solution of this constraint. In contrast, global consistency implies that each value of the variable of the CSP can be extended to at least one solution of the entire CSP.

Therefore, global consistency is much strong enforcement of consistency. In particular, the search interleaved with global consistency is backtrack free.

We introduce regular CSPs as CSPs (as defined in Definition 1), where all constraints must be regular membership constraints.

Definition 6: Regular CSP. A regular constraint satisfaction problem (RCSP) is defined as a 3-tuple $P = (X, D, C)$ with $X = \{x_1, x_2, \ldots, x_n\}$ is a set of variables, $D = \{D_1, D_2, \ldots, D_n\}$ is a set of finite domains where D_i is the domain of x_i $\forall i \in \{1, 2, \ldots, n\}$ and $C = \{c_1, c_2, \ldots, c_m\}$ is a set of regular constraints on the subsets of X.

The only difference between a CSP and an RCSP is that a CSP allows all kinds of constraints while an RCSP only allows regular constraints. Because every regular constraint is hyper arc consistent, an RCSP with only one (regular) constraint is globally consistent. This means no backtracking will be necessary when solving such a problem.

3 Transformation From a CSP Into a RCSP

In this section, we briefly show that for every CSP exists at least one equivalent RCSP (theoretically), and then we present a selection of effective constraint transformations for the practical use.

3.1 Theoretical Considerations

According to Definition 1 a CSP P has a fixed number n of variables $X = \{x_1, \ldots, x_n\}$ and their respective domains $D = \{D_1, \ldots, D_n\}$ are finite, it follows that the potential

solution space of the CSP P is limited by l (1).

$$l = \prod_{i=1}^{n} |D_i| \quad (1)$$

If the number of solutions is finite, then these can be enumerated by a regular language; besides, for every regular language exists at least one automaton which describes this language (Myhill-Nerode theorem [8]). It follows that for every CSP P exists at least one regular CSP P_{reg} which is declaratively equivalent to P and contains only one regular constraint.

Such a RCSP $P_{reg} = (X, D, C_{reg})$ can obviously be generated if all solutions of P are known, because, given the language L, we can easily create an appropriate deterministic finite automaton (DFA) M. In practice, however, this is not useful because we intend to use the RCSP P_{reg} to find one, all or the best solutions of P faster than with the original CSP.

For this reason, we consider the direct transformation from multiple types of constraints to regular constraints. For several global constraints (see 3.2), we defined corresponding efficient transformations, among them count, global cardinality constraint, and table constraints.

3.2 Transformations From Special Global Constraints Into Regular Constraints

Nevertheless, there is a gap between theory and practice because finding all solutions for the fast transformation from CSP to RCSP can hardly be used in practice. Thus, we seek an efficient transformation method for global constraints. In the following, we are going to present the transformations for several global constraints defined and implemented in [13].

The Count Constraint. The count constraint is defined as $count(X', occ, v)$, where $X' = \{x_1, x_2, ..., x_n\}$ is a set of variables ($X' \subseteq X$, X is the set of Variables in the CSP) and $occ \in X$ with $D_{occ} = \{occ_{min}, ..., occ_{max}\}$ is a variable to denote admissible numbers of occurrences of the value $v \in N$ in X'.

In order to transform a count constraint into a regular constraint, we take into consideration the following two different cases:

Case 1: The *occ* variable is not used in other constraints.

In this case, the *occ* variable can be implicitly represented in the DFA. We create a DFA $M = (Q, \Sigma, \delta, q_0, F)$ with

- $Q = \{q_0, q_1, ..., q_{occ_{max}}\}$,
- $\sum = \bigcup_{i=1}^{n} D_i$
- q_0 is the initial state,
- $F = \{q_i \mid i \in D_{occ}\}$
- $\delta(q_i, v) = q_{i+1} \mid \forall i \in \{0, ..., occ_{max-1}\}$
- $\delta(q_i, v_2) = q_i \mid \forall i \in \{0, ..., occ_{max}\}, \forall v_2 \in \Sigma \backslash v$.

The regular constraint *regular(X', M)* is equivalent to the original count constraint, and can be used to replace the count constraint.

a) Case 1 b) Case 2

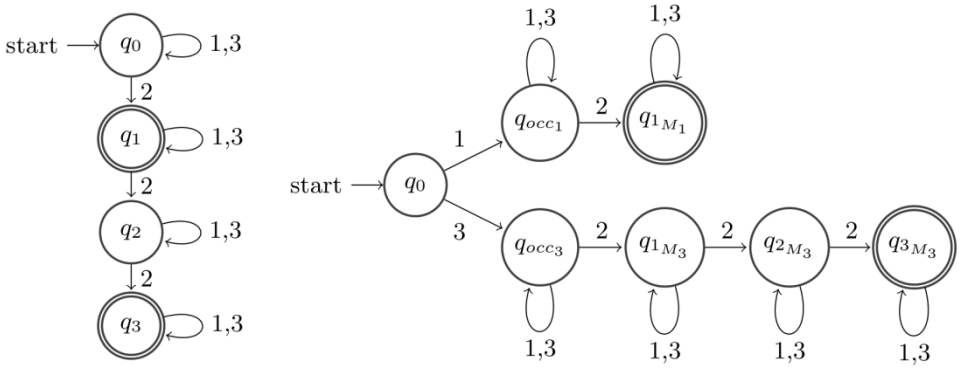

Fig. 1. Automaton representations of the *count* constraint from example 1 (both cases).

Example 1. Given the count constraint *count({x_1, x_2, x_3, x_4, x_5}, occ, 2)* with $D_i = \{1, 2, 3\}\ \forall i \in \{1, 2, ..., 5\}$ and $D_{occ} = \{1, 3\}$. The legal assignments for variables of count constraint must contain value 2 with 1 or 3 occurrences. Our transformation yields the automaton *M*:

$M = (\{q_0, q_1, q_2, q_3\}, \{1, 2, 3\}, \delta, q_0, \{q_1, q_3\})$ with
$\delta = \{\ ((q_0, 1) \rightarrow q_0), ((q_0, 2) \rightarrow q_1), ((q_0, 3) \rightarrow q_0),$
$((q_1, 1) \rightarrow q_1), ((q_1, 2) \rightarrow q_2), ((q_1, 3) \rightarrow q_1),$
$((q_2, 1) \rightarrow q_2), ((q_2, 2) \rightarrow q_3), ((q_2, 3) \rightarrow q_2),$
$((q_3, 1) \rightarrow q_3), ((q_3, 3) \rightarrow q_3)\}.$

A graphical presentation of *M* is depicted in Fig. 1a.

Case 2: The *occ* variable is used by other constraints.

In this case, we perform as follows: First, we create an automaton *M* with set of states $Q = \{q_0\} \cup (q_{occ_t} \mid \forall t \in D_{occ})$. We let q_0 be the initial state, all other states are labelled with occ_t, $t \in D_{occ}$. In the next step, we create $|occ|$ successor automata M_t, $t \in D_{occ}$ as described for case 1 with variables X_0 , value *v* and *occ = t* as input. We combine these automata M_t with the automaton *M* such that the initial state of each M_t is replaced with the state q_{occ_t} from *M*. The original count constraint can replace by the regular constraint *regular({occ, x_1, ..., $x_{|X'|}$}, M)* with the resulting DFA *M*. A graphical presentation of *M* for the previous example is depicted in Fig. 1b.

The *global cardinality constraint.* The *global cardinality constraint* can be interpreted as several count-constraints for which an efficient transform into regular constraints is given above.

The *stretch* constraint. A good description of the *stretch* constraint and its transformation into a regular constraint is given in [11].

The *table*-constraint. The table constraint is defined as *table(X', A, b)*, where $X' = \{x_1, x_2, ..., x_n\} \subseteq X$ is a set of variables, and the Matrix $A^{m \times n}$ list all tuples extensionally. If boolean variable

b is true, any allowed assingment of *X'* for the table constraint must be limited among the rows of matrix *A*, otherwise not.

Without loss of generality, we only sketch the automaton construction for *b* is true. For each table constraint with *b* is false an equivalent table constraint with *b* is true can be found.

The idea here is to create *m* DFAs M_j, $j \in \{1, ..., m\}$ which only accept the words $w_j = A_{j,1}A_{j,2}...A_{j,n}$. We build *M* as the union of these *m* automata M_j . This DFA *M* can be used as a replacement of the *table* constraint with *regular(X', M)*.

We have presented examples of possible replacements of constraints with regular constraints. However, there exist more, e.g., cumulative constraint, which, however, is beyond the scope of this paper. Please note that a regular constraint created directly from another constraint might be slower than the original constraint. Thus, this transformation is, fairly often, useful if it is possible to combine several regular constraints into one new regular constraint. This can be done by the intersection of automata in which several regular constraints are combined together into one new regular constraint so that fewer backtracks and failures will be encountered. The next section will show more details about this.

4 The Use of Regular CSPs

In this section, we discuss advantages and disadvantages of the transformation of a general CSP *P* into an equivalent regular CSP P_{reg}, and use a rostering problem as an example.

4.1 Advantages and Disadvantages

There are some obvious disadvantages: a transformation for every constraint is necessary (not trivial for some global constraints) and it may be time-consuming to transform a constraint to a regular constraint. Furthermore, the newly created regular constraints might even reduce the performance of the solver.

To illustrate the advantages we briefly repeat how a final domain solver finds solutions. The solver uses depth-first search (DFS) nested with consistency enforcement. In contrast to some other constraints (e.g. sum or cumulative constraint), the regular constraint enforces hyper arc consistency (local consistency) over its variables. Therefore, transforming constraints with a lower consistency level into regular constraints, can (while potentially in-/decreasing the complexity for consistency enforcement) reduce the number of fails and backtracks in DFS in the solution process.

Furthermore, an important advantage is that several constraints can be combined into one new constraint by means of automata intersection. This reduces the number of overlapping constraints and, if we reach a single (hyper-arc consistent) constraint, then the whole CSP has global consistency and no backtracking is necessary any more. Similarly, this idea will often use to substitute pairwise inequality constraints with one allDifferent [13, 3] constraint.

A great advantage is that the transformation into regular constraints allows combining constraints of originally different types into a new constraint with joint

propagation function. This happens by generating the automata, building their intersection (create product automata), and minimizing the resulting DFAs. This, furthermore, leads to a removal of redundancy and, in general, to a reduction of fails and backtracking steps in the depth-first search process.

This preprocessing algorithm happens before other algorithms like common parallelization algorithms will be used. This means that the performance improvements of both approaches (regularization and parallelization) can be used simultaneously.

4.1 A Rostering Example

In this section, an example is shown to demonstrate the power of regular constraints. We use the same example as [9] but the results are summarized. The study [9] states that regular constraints can improve the performance of CSPs; however, the regular constraints were created directly by the developer requiring domain knowledge about DFAs and regular languages. In this paper we added some automatic transformations by the solver so that the designer of the CSP can model the problem by normal constraints without additional knowledge about regular languages.

CSP 1. Consider a rostering problem, CSP $P = (X, D, C)$, with $X = \{x_1, x_2, ..., x_n \mid n \bmod 7 = 0\}$, $D = \{D_1, D_2, ..., D_n\}$, where $i \in \{1, ..., n\}: D_i = \{0,1,2,3\}$ and $C = \{C_{shiftRequirements}, C_{shiftRepetitions}, C_{shiftOrder}, C_{equalDays}\}$.

Based on the constraints *count*, *stretch*, *table* and *arithm* which are defined in [13, 3] the constraints C can be modeled as follows:

- $C_{shiftRequirements}$[1] $= \{count(X_i, occ, j) \; \forall i \in \{1, 2, ..., 7\}, j \in \{0, 1, 2, 3\}\}$ where $X_i = \{x_i, x_{7+i}, x_{14+i}, ...\} \subset X$ and $occ = \{\left\lfloor \frac{n}{7*4} \right\rfloor, \left\lceil \frac{n}{7*4} \right\rceil\}\}$
- $C_{shiftRepetitions} = stretch(X, \{2, 2, 2, 2\}, \{4, 4, 4, 3\}, \{0, 1, 2, 3\})$
- $C_{shiftOrder} = \{table(x_i, x_{i+1}, \begin{pmatrix} 3 & 1 \\ 3 & 2 \\ 2 & 1 \end{pmatrix}, false) \; \forall i \in \{1, 2, ..., n - 1\}\}$
- $C_{equalDays} = \{arithm(x_i, \text{“=“}, x_{i+1}) \; \forall i \in \{6, 13, ..., n - 1\}\}$

We consider n $(n \in N, n \bmod 7 = 0)$ days, i.e. several weeks. A variable x_i, $i \in \{1, 2, ..., n\}$ represents the shift of a person A_1 at day i, where we have four possible shifts: 0, 1, 2, and 3 represent a day off, an early shift, a late shift, and a night shift respectively.

As typical for many rostering problems, we just consider the plan of one person A_1 and assume, that the plan for further staff is received by rotating A_1s plan by e.g. a week. For example given a shift plan $sol_{A_1} = (v_1, v_2, ..., v_7, v_8, ..., v_{14}, v_{15}, ..., v_n)$ (as a solution of the CSP P) for A_1, the plan for a person A_2 would be $sol_{A_2} = (v_8, ..., v_{14}, v_{15}, ..., v_n, v_1, v_2, ..., v_7)$. The constraints C are explained in the following:

- The $C_{shiftRequirements}$ constraints guarantee that for each day for each shift are exactly as many staff persons as needed. For example there must be between

[1] The use of *global cardinality constraints* would be an alternative here, but, in performance measurements yields worse results in our experiments.

3 and 4 employees each Monday in morning shift.

- The $C_{shiftRepetitions}$ constraints guarantee that the same shift in consecutive days is limited by a lower and an upper bound. This is necessary because in some countries (i.e. Germany) ergonomic knowledge must be respected [4].
- The $C_{shiftOrder}$ constraints restrict the order of shifts. We consider a forward rotation of shifts which guarantees conformance with the regulations on rest periods in the German labor time law [5, §5].
- $C_{equalDays}$ are constraints which guarantee that on Saturday a person always has the same shift as on the following Sunday, which is recommended in [4].

Remark: Of course, for real rostering problems, further restrictions and recommendations must be considered.

Evaluation. A series of tests with different values for *n* (28, 35, 42, 49, 56) days and four different search strategies, a combination of the variable selectors *Smallest* and *FirstFail* and the value selectors *IntDomainMin* and *IntDomainMedian* was investigated. We applied variable and value selection strategies as defined by Charles Prud'homme for the Choco 4 solver (for details see [13]).

We compare the solution behavior of the original version of the CSP *P* and its regular version CSP P_{reg}, as a result of the intersection of the automatons created with the presented transformations.

In the regular version P_{reg} all constraints have been transformed as described in the previous sections and then all automata - except the $C_{shiftRequirements}$ - were intersected and minimized to a new constraint $c_{regular}$ as described before. We omitted the intersection of the automatons which represents the $C_{shiftRequirements}$ constraints with the other automata because this requires more time as available. The presented transformations in our example have a time requirement in double-digit micro seconds, which is negligible in comparison to the needed solution time.

Table 1 shows the average improvements of the constructed regular CSP P_{reg} in comparison to the original CSP *P* over all four search strategies and for the respective values of *n*. For example, for *n = 28* days the regular approach was in average 4.546 times faster to find the first solution and 2.413 times faster to find all solutions in comparison to the original approach. For *n = 42* and *n = 49* there exist no solutions. The regular approach was *3.950* respectively more than *2.780* times faster than the original approach to come to this conclusion. Because problems can need a lot of time, we limited the solution time. The original approach for *n = 49* doesn't find a solution in this time limit but the time limit was *2.780* times over the time which the regular approach needed. So the regular approach was at least *2.780* times faster as the original approach.

Table 1. Statsitics for n = 28, …, 56.

Criteria \n	28	35	42	49	56
1st solution	4.546	5.774	-	-	>6.278
All solutions	2.413	6.146	-	-	-
No solution	-	-	3.950	>2.780	-

For *n = 56* not all solutions were found in the time limit, but the first solution was found more than six times faster in average and within the limited time more than nine times so many solutions were found. Furthermore table 2 shows the reduction factor of the number of nodes (fails resp. backtracks) of the original approach divided by the number of nodes (fails resp. backtracks) of the regular approach. You can see that the full search tree of the regular approach has at least two times less nodes, fails and backtracks as the original approach which is also an indication for the faster solution speed. For the reason of the time limit the whole search tree was not created for *n = 56* so that it cannot be said how much smaller the search tree for this RCSP is.

Table 2. Statsitics for n = 28, …, 56.

Criteria \n	28	35	42	49	56
Nodes	3.018	2.501	4.538	>2.385	-
Fails	3.043	3.636	4.538	>2.385	-
Backtracks	3.069	11.824	4.538	>2.385	-

5 Conclusion

We are going to give a brief summary of this paper, a conclusion, and we explain our next steps.

Summary and Conclusion. We presented a new approach for the modelling and optimization of general CSPs using the regular constraint.

It was explained how a set of selected global constraints can be transformed into a regular CSP. A regular constraint can be minimized by the use of automata intersection and minimization. By employing a rostering example we have demonstrated that this approach allows to reduce the size of the search tree and to significantly improve the solution speed. For practical problems the execution time for the transformation must be taken into consideration.

Our investigations support that our approach can be applied successfully when considering sub-problems of a potentially large CSP *P* and, thus, subsets of its variables *X*. Transforming only sub-problems (instead of *P* completely) is, thus, much faster and, still, leads to a reduction of the number of constraints (also the number of backtracks) and of redundancy which, altogether, improves the solution speed.

Future work. Future work will be finding more direct transformations for global constraints, finding an automated transformation algorithm, investigating promising variable orderings to optimize the size of the DFAs. Furthermore we would like to study variable and value orderings for the regular constraint, research on potential benefits of decomposition of automata, general (static) criteria to decide when to apply the approach as well as extracting promising application areas in general.

References

1 Apt, K.: Principles of Constraint Programming. Cambridge University Press,

New York, NY, USA (2003)
2 Dechter, R.: Constraint processing. Elsevier Morgan Kaufmann (2003)
3 Demassey, S.: Global Constraint Catalog. http://sofdem.github.io/gccat/ (2014-06-05), last visited 2017-10-24.
4 German Federal Institute for Occupational Safety and Health (BAuA) - Design of night and shift work. http://www.baua.de/ (2013), last visited 2017-05-10.
German working time law from 6. June 1994, (BGBl.IS.1170,1171), last updates by Art.3 para. 6 of the Law of 20 April, 2013 (BGBl.IS.868). http://www.gesetze-im-internet.de/arbzg (1994), last visited 2017-05-10.
5 Gottlob, G., Grohe, M., Musliu, N., Samer, M., Scarcello, F.: Hypertree decompositions: Structure, algorithms, and applications. In: International Workshop on Graph-Theoretic Concepts in Computer Science. pp. 1-15. No. 3787 in Lecture Notes in Computer Science, Springer (2005)
6 Hellsten, L., Pesant, G., Beek, P. v.: A Domain Consistency Algorithm for the Stretch Constraint, Principles and Practice of Constraint Programming - CP 2004. In: Wallace, M. (ed.) Lecture Notes in Computer Science, vol. 3258, pp. 290-304, Springer.
7 Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley (1979)
8 Löffler, S., Liu, K., Hofstedt, P., (2017). The Power of Regular Constraints in CSPs. In: Eibl, M., Gaedke, M. (ed.), Lecture Notes in Informatics (LNI), INFORMATIK 2017. Society for computer science, Bonn. (pp. 603-614).
9 Marriott, K.: Programming with Constraints - An Introduction, MIT Press (1998), Cambridge
10 Paltzer, N.: Regular Language Membership Constraint. Seminararbeit, Saarland University, Germany (2008)
11 Pesant, G.: A Filtering Algorithm for the Stretch Constraint, Principles and Practice of Constraint Programming - CP 2004. In: Walsh, T. (ed.) Lecture Notes in Computer Science, vol. 2239, pp. 183-195. Springer (2001)
12 Prud'homme, C., Fages, J.G., Lorca, X.: Choco Documentation. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S. (2016), http://www. Choco-solver.org/, last visited 2017-06-20
13 Rossi, F., Beek, P.v., Walsh, T.: Handbook of Constraint Programming. Elsevier, Amsterdam, First edn. (2006)
14 Van Hoeve, W. J., Katriel, I.: Global Constraints, Chapter 6. Elsevier, (2006)