The Power of Regular Constraints in CSPs

Sven Löffler[1], Ke Liu[1] and Petra Hofstedt[1]

Abstract: This paper discusses the use of the regular membership constraint as a replacement for other (global) constraints. The goal is to replace some or all constraints of a constraint satisfaction problem (CSP) with regular constraints and to combine them into a new regular constraint to remove redundancy and to improve the solution speed of CSPs. By means of a rostering problem as an example it is shown that our approach allows a significant improvement of the solution performance due to a reduction of the size of the search tree.

Keywords: CSP, Global Constraints, Regular Constraint, Optimization, Refinement, DFA

1 Introduction

Constraint programming is a powerful method to model and solve NP-complete problems in a declarative way. Typical and the most and with biggest success researched problems in constraint programming are rostering, graph coloring and satisfiability (SAT) problems [Ma98].

A perfect declarative program would be solving several descriptions of a problem with the same speed. Like we will show, this is not the case in general constraint problems. This paper introduces a way to make a constraint satisfaction problem (CSP) problem more independent from the description by the use of regular constraints.

In general, for real problems exist many ways to model them by constraints. Different constraint programs which describe the same real problem need different execution resp. solution time. One reason is that there are different constraints which can describe the same restrictions but use different propagation algorithms. Therefore, a replacement of constraints by constraints of another type which can solve a certain problem faster or which can be combined to more specific constraints can be a promising approach.

This paper discusses the use of the regular membership constraint in place of other (global and set of local) constraints. The goal is it to replace some or all constraints of a constraint satisfaction problem (CSP) with regular constraints and to combine some or all of these regular constraints to a new regular constraint to remove redundancy, unnecessary backtrack

[1] Brandenburg University of Technology Cottbus - Senftenberg, Programming Languages and Compiler Construction, Konrad-Wachsmann-Allee 5, D-03044 Cottbus, {Sven.Loeffler, Ke.Liu, Petra.Hofstedt}@b-tu.de

doi:10.18420/in2017_57

and to improve the solution speed of CSPs. The regular constraint and its propagation algorithm [Pe01, Pa08, HPvB04] provide the basis of this approach.

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

2 Basic principles of constraint programming

Constraint programming (CP) is a programming paradigm which developed from the logical programming since the middle of the 1980s. CP is qualified for declarative description and effective solution of big problems for planning and scheduling, configuration and optimization and for dealing with tasks for which only incomplete information is available. In contrast to the imperative programming it is not necessary to define the algorithmic steps which lead to a solution, instead, in CP we describe the problem and its solutions in a declarative way.

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

CSP [De03] A constraint satisfaction problem (CSP) is defined as a 3-tuple $P = (X, D, C)$ with
$X = \{x_1, x_2, \dots, x_n\}$ is a set of variables,
$D = \{D_1, D_2, \dots, D_n\}$ is a set of finite domains where D_i is the domain of x_i and
$C = \{c_1, c_2, \dots, c_m\}$ is a set of primitive or global constraints containing between one and all variables in X.

Primitive constraints are simple relations like $x_1 = x_2$, $x_1 \neq x_2$ or $x_1 + x_2 < x_3$. In contrast to this, global constraints [vHK06] are more complex and mostly base on efficient algorithms to solve a problem faster as with equivalent primitive constraints. Examples of global constraints are the *allDifferent* constraint, the *global cardinality constraint*, *cumulative*, *count*, *sum* and *regluar*.

Global constraint [RBW06] A global constraint is a condition which describes a relationship between a non-fixed number of variables.

Solving a CSP means finding assignments to $X = \{x_1, x_2, \dots, x_n\}$ such that the value of x_i is in D_i, $\forall i \in \{1, 2, \dots, n\}$, while all the constraints are satisfied. To find a solution of a CSP, consistency enforcement methods and search (mostly depth-first search) are used. A consistency enforcement method is an algorithm which tries to remove elements from the variables domains which are not part of solutions of the CSP.

While often it is not possible to reduce the domains such that only values remain, which are part of at least one solution *(global consistency)*, typically we remove values which violate single constraints and reach *hyper arc consistency* (or *local consistency*).

Hyper arc consistency [Ap03] A constraint $c \in C$, $c \subseteq D_1 \times \ldots \times D_k$ is called hyper arc consistent if for every $i \in \{1, \ldots, k\}$ and $\forall a \in D_i$:

$$\exists (a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_k) \in D_1 \times \ldots \times D_{i-1} \times D_{i+1} \times \ldots \times D_k : \\ (a_1, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_k) \in c$$

Intuitively, a constraint c is hyper arc consistent if every domain value of each variable is part of at least one solution of c. Unfortunately, mostly it is not possible to conclude from the hyper arc consistency of all single constraints in a CSP to the global consistency of the whole CSP.

Global consistency [De03] Let be $P = (X, D, C)$ 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 consistant, 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 constraints of C in conjunction.

Finding a valid assignment, i. e. solution, to a constraint satisfaction problem is usually accomplished by combining or nesting backtracking search with consistency enforcement. To do so, a variable is assigned one of the values from its domain in every node of the search tree. Due to time-complexity issues, the used consistency methods are rarely complete [De03].

2.1 A general problem in (CSP) modelling

When describing real problems using CSPs, there are typically many possible ways of interpretation and modeling supported by various constraints. The resulting CSPs, thus, can have very different solution speed and behavior. As an example consider the following problem.

Problem 1: Given n, $n \in \mathbb{N}$, $n > 8$, find 100.000 permutations of the numbers from 1 to n.

This sounds like an easy problem which can be modeled very easily by the following CSP.

CSP 1.1: Let $P = (X, D, C)$ be a constraint satisfaction problem (CSP) with:
$X = \{x_1, x_2, \ldots, x_n\}$

$D = \{D_1, D_2, \dots, D_n \mid D_i = \{1, 2, \dots, n\},\ i \in \{1, 2, \dots, n\}\}$
$C = \{\mathit{allDifferent}(x_1, x_2, \dots, x_n)\}$

The *allDifferent* constraint (see e. g. [vHK06]) is a global constraint ensuring that all variables are pairwise different. CSP 1.1 obviously solves our problem but there are other modelings, too. Take a look on a second CSP describing our problem.

CSP 1.2: Let $P = (X, D, C)$ be a constraint satisfaction problem (CSP) with:

$X = \{x_1, x_2, \dots, x_n\}$
$D = \{D_1, D_2, \dots, D_n \mid D_i = \{1, 2, \dots, n\},\ i \in \{1, 2, \dots, n\}\}$
$C = \{x_i \neq x_j \mid i, j \in \{1, 2, \dots, n\},\ i < j\}$

While both modelings, i. e. CSP 1.1 and CSP 1.2 declaratively are equivalent, their propagation mechanisms differ. CSP 1.1 considers the problem from a global perspective using a global constraint and a corresponding propagation mechanism [Ré94, Ló03] over all variables. CSP 1.2 considers a conjunction of $\frac{n*(n-1)}{2}$ disequality constraints. Thus, CSP 1.1 mainly relies on consistency enforcement mechanisms while CSP 1.2 uses local consistency enforcement nested with a huge amount of search. Often it is assumed that global constraints result in a faster solution process. However, this is not necessarily the case. For our problem, a small test example shows that for smaller problems up to $n \leq 45$ CSP 1.2 is faster than CSP 1.1.

Whether the solution process for a certain modeling is faster than for a semantically equivalent CSP description, depends on many different things, like the kind of constraints, the number of variables, variable orderings, domain sizes, search strategy, propagation engine and much more. If only one of these points changes, it may result significantly in the speed of the solution process. Finding the best combination of search strategy, propagation engine, used constraints, variable order and so on is one of the biggest problems in CSPs and it is not yet completely solved or even researched. Until someone might find an algorithm to detect best combinations of settings for general problems, it is useful to consider settings which are better in average or in special areas of applications.

2.2 Regular CSPs

In this section, we introduce regular constraints as a means of modeling constraint problems. First, we briefly recall the notion of a DFA [La12].

DFA A deterministic finite automaton (DFA) is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$, where

- Q is a finite set of states,
- Σ is the finite input alphabet,
- δ is a transformation function $Q \times \Sigma \rightarrow Q$,

- $q_0 \in Q$ is the initial state,
- $F \subseteq Q$ is the set of final or accepting states.

A word $w = (w_1, w_2, \ldots, w_n)$ is accepted by M, i. e. $w \in L(M)$, if the corresponding DFA M with the input w stoppes in a final state $f \in F$.

Regular membership constraint [vHK06] Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA, let $X = \{x_1, \ldots, x_n\}$ be a set of variables with domains $D = \{D_1, D_2, \ldots, D_n\}$, $\forall i \in \{1, \ldots, n\} : D_i \subseteq \Sigma$. The regular language membership constraint (regular constraint) is defined as:

$$regular(X, M) = \{(w_1, \ldots, w_n) \mid \forall i \in \{1, \ldots, n\} : \ w_i \in D_i, \ (w_1, w_2, \ldots, w_n) \in L(M)\}$$

In the following, we use the notion of regular CSPs:

Regular CSP A regular constraint satisfaction problem (regular CSP) is defined as a 3-tuple $P = (X, D, C)$, where
$X = \{x_1, x_2, \ldots, x_n\}$ 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 and
$C = \{c_1, c_2, \ldots, c_m\}$ is a set of regular constraints over variables from X.

3 Transformation from a general CSP into a regular CSP

In this section we briefly show that for every CSP there exist an equivalent regular CSP and we enumerate a selection of constraints for which effective transformations exist.

3.1 A theoretical consideration

The solution space S, i. e. the number of solutions, of every CSP P is finite by definition. For P, there is a finite number n of variables $X = \{x_1, \ldots, x_n\}$ with finite domains $D = \{D_1, \ldots, D_n\}$. A CSP P over n variables but with the only constraint *true* has a solution space S of size $\Pi_{i=1}^{n} D_i$. In general, constraints can restrict domains but they can't add values to a domain. Thus, the solution space S' of a CSP P', i. e. P but with the constraint set $C' = \{c_1, \ldots, c_m\}$ is smaller or equal to the solution space of the original CSP P: $S' \subseteq S$.

A finite solution space can also be described by a finite language L over the finite alphabet $\Sigma = \bigcup_{\forall i \in \{1,\ldots,n\}} D_i$. Every word w of L has length n. A word $w = (w_1, w_2, \ldots, w_n)$ is an element of L iff there exists a solution $s = \{x_1 = w_1, \ldots, x_n = w_n\}$ for P.

Since L is finite, there exists a DFA M for L (Myhill-Nerode theorem [HU79]) and, therefore, for the solutions of the CSP P. It follows that for every CSP $P = (X, D, C)$ there exists an equivalent regular CSP $P' = (X, D, C')$ where $C' = \{regular(X, M)\}$.

3.2 Transformations

As discussed previously, for every general CSP $P = (X, D, C)$, there exists a regular CSP $P' = (X, D, C')$. Obviously P' can be generated if all solutions of P are known because, given the language L, we can easily create an appropriate DFA A. In practice, however, this is not useful because we intend to use the regular CSP P' to find one or all solutions of P faster.

For this reason, we are interested in direct transformations of (global) constraints into regular constraints. For some constraints corresponding efficient transformations are known (e. g. for *stretch* see [Pa08]), for others we defined such, among them *count*, the *global cardinality constraint*, and *table* constraints.

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' and show cases in which this can be very useful and improve the solution speed.

4.1 Advantages and disadvantages

Let's start with the obvious disadvantages: We need a transformation for every constraint (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.

What are the advantages on the other hand? The solution process of a CSP consists of depth-first search (DFS) nested with consistency enforcement. The regular constraint has hyper arc consistency over its variables. Thus, transforming constraints with a lower consistency level into regular constraints, can (while potentially increasing the effort for consistency enforcement) reduce the number of fails and backtracks in DFS in the solving process. Furthermore, an important advantage is that several constraints can be combined into one new constraint with hyper arc consistency by using the automata intersection.

As an (well-known) example, consider Fig. 1. On the left-hand side (a) we see a constraint network over the variables $X = \{a, b, c\}$ with domains $D = \{D_a, D_b, D_c\}$, where $D_a = D_b = D_c = \{1, 2\}$, and $C = \{a \neq b, b \neq c, a \neq c\}$. The right-hand side (b) gives a semantically equivalent constraint network, but with another constraint representation: $C' = \{allDifferent(a, b, c)\}$.

A solver using search nested with consistency enforcement will find out, that no solution exists in both cases. However, in case (a) the propagation method of the solver uses local

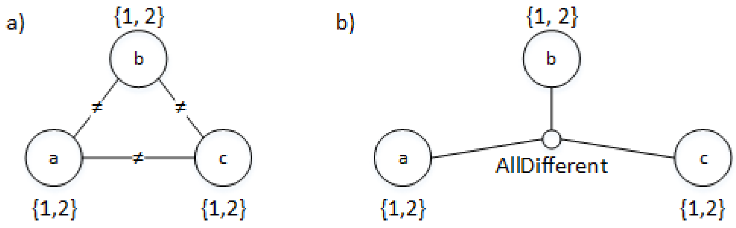

Fig. 1: The use of the allDifferent constraint

consistency over the three disequality constraints and, thus, needs backtracking search to finally reduce the solution space to the empty set. In contrast to this, in the second case (b), the *allDifferent* constraint is propagated with global consistency enforcement and directly yields the empty solution space without search. Notice, that the constraints (in both cases) have hyper arc consistency but hyper arc consistency leads only sure to global consistency if the constraints are not overlapping.

A transformation of the constraint network of Fig. 1 case (a) yields a representation similar to case (b): In a first step, every disequality constraint is transformed into a regular constraint, i. e. C is transformed into C'', where
$C'' = \{regular(\{a,b,c\}, M_1),\ regular(\{a,b,c\}, M_2),\ regular(\{a,b,c\}, M_3)\}$ and with DFAs M_1, M_2, and M_3 (for $a \neq b$, $b \neq c$, and $a \neq c$ resp.) given Fig. 2.

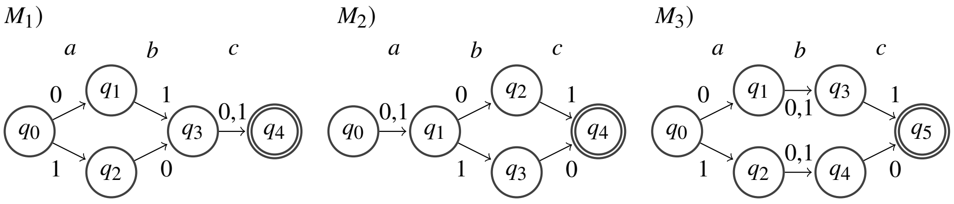

Fig. 2: Equivalent DFAs for the disequality constraints from Fig. 1 (a)

In a second step, an intersection of the automata M_1, M_2, and M_3 is built and minimized which yields one new automaton M_{final} and a corresponding regular constraint. In our example, the language M_{final} accepts is the empty set. Thus, propagation of the newly created regular constraint on M_{final} is sufficient to find out about the empty solution space and search is not needed (as in case (b), Fig. 1).

One big advantage of the transformation of general CSPs into regular constraints (in contrast to using the original constraints) is that this method allows the combination of constraints of originally different type into a one new constraint with joint propagation function. This is possible by generating the automata, building their intersection, and minimizing the resulting DFA as described above. This, furthermore, leads to a removal of redundancy and, in average, to a reduction of fails and backtracking steps in the depth-first search process.

4.2 A rostering example

In this section, a more complex example is shown to demonstrate the power of regular constraints. Consider a rostering CSP $P = (X, D, C)$, where

$X = \{x_1, x_2, \ldots, x_n \mid n \ mod \ 7 = 0\}$
$D = \{D_1, D_2, \ldots, D_n \mid \forall i \in \{1, \ldots, n\} : \ D_i = \{0, 1, 2, 3\}\}$
$C = \{C_{shiftRequirements}, C_{shiftRepetitions}, C_{shiftOrder}, C_{equalDays}\}$

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

As typical for many staff 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 $sol_{A_1} = (v_1, v_2, \ldots, v_7, v_8 \ldots, v_{14}, v_{15}, \ldots, v_n)$ (as a solution of the CSP P) for A_1, the plan for a person A_2 would be $sol_{A_2} = (v_8, \ldots, v_{14}, v_{15}, \ldots, v_n, v_1, v_2, \ldots, v_7)$, and for a certain person A_3 it might be $sol_{A_3} = (v_{15}, \ldots, v_n, v_1, v_2, \ldots, v_7, v_8, \ldots, v_{14})$.

The constraints C are explained in the following:

$C_{shiftRequirements}$ contains the core constraints which describes how many shifts of each kind (i. e. from 0, 1, 2, 3) are necessary for every day (i. e. $1, 2, \ldots, 7$) in a week. Often these values vary over the week, for example, less staff is needed on weekends (days 6 and 7). To express such restrictions, the *count* constraint is useful. Let's for our example assume that we wish a nearly equal staffing per day. This yields the following *count* constraints:

$\forall i \in \{1, 2, \ldots, 7\}, j \in \{0, 1, 2, 3\} : count(X_i, occ, j)$
where $X_i = \{x_i, x_{7+i}, x_{14+i}, \ldots\} \subset X$ and $occ = \{\lfloor \frac{n}{7*4} \rfloor, \lceil \frac{n}{7*4} \rceil\}$

Remark: The use of *global cardinality constraints* would be an alternative here, but, yield in performance measurements worse results.

$C_{shiftRepetitions}$ contains constraints to limit the minimum and maximum number of the same shift in consecutive days. In Germany, ergonomic knowledge must be respected. This means for example that the minimum number of days in a row with the same shift should be two [Bu13]. As well, the maximum number of repetitions is limited. In our example, we allow a maximum of four repetitions, except that on night shifts, where we limit it on three.

The $stretch(X, min, max, values, \ldots)$ constraint is perfect to model these requirements [Pa08] as follows:

$stretch(X, \{2, 2, 2, 2\}, \{4, 4, 4, 3\}, \{0, 1, 2, 3\}, \ldots)$

$C_{shiftOrder}$ restricts the order of shifts. We require that only later shifts or days off can place after an earlier shift, this guarantees conformance with the regulations on rest periods in the German working time law [Ar94, §5]. Here, we use *table* constraints, which can express allowed or disallowed relations by tuples. The following *table* constraint says that for each two successive days x_i and x_{i+1} after a night shift there must not follow an early or a late shift (i. e. tuples (3, 1) and (3, 2) are forbidden) and that a late must not be followed by an early shift (i. e. tuple (2, 1) is forbidden).

$$\forall i \in \{1, 2, \ldots, n-1\} : table(x_i, x_{i+1}, \begin{pmatrix} 3 & 1 \\ 3 & 2 \\ 2 & 1 \end{pmatrix}, false)$$

$C_{equalDays}$ are constraints which guarantee that on Saturday a person always has the same shift as on the following Sunday, which is recommended [Bu13]. Here, we use *arithm* constraints (arithmetic equality).

$$\forall i \in \{6, 13, \ldots, n-1\} : arithm(x_i, \text{“ = “}, x_{i+1})$$

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

4.2.1 Evaluation

A series of tests with different values for n (28, 35, 42, 49, 56) and four different search strategies (see Table 1) was investigated. We applied variable and value selection strategies as defined by Charles Prud'homme for the Choco 4 solver (for details see [Ch17, PFL16]).

Name	Variable selector	Value selector
Search 1	“Smallest”	“IntDomainMin”
Search 2	“Smallest”	“IntDomainMedian”
Search 3	“FirstFail”	“IntDomainMin”
Search 4	“FirstFail”	“IntDomainMedian”

Tab. 1: The used search strategies

Tables 2 to 5 show the average results over all four search strategies and for the respective values of n. For example, Table 2 shows results for the staff rostering over four weeks, i. e. $n = 4 * 7 = 28$.

We compare the solution behaviour of the *original* version of the CSP P and its *regular* version, i. e. a CSP P' as a result of the transformation. In the *regular* version P' all constraints - except the $C_{shiftRequirements}$ - were transformed like described in the previous chapter (including intersection and minimization of the automaton) to a new constraint $c_{regular}$ as described before. We omitted the transformation of $C_{shiftRequirements}$ because that, in particular the intersection step, needs more time as we can save.

$n = 28$	t_1 in s	t_{44} in s	Nodes	Backtracks	Fails
Original	0.110	0.734	10501	20917	10415
Regular	0.024	0.304	3480	6873	3393
Improvement	4.546	2.413	3.018	3.043	3.069

Tab. 2: Statistics for $n = 4 * 7 = 28$

$n = 35$	t_1 in s	t_{100} in s	t_{10000} in s	t_{217339} in s	Nodes	Backtracks	Fails
Original	0.045	0.234	2.312	26.319	1262001	2089324	827324
Regular	0.008	0.073	0.630	4.283	504647	574618	69970
Improvement	5.774	3.210	3.670	6.146	2.501	3.636	11.824

Tab. 3: Statistics for $n = 5 * 7 = 35$

$n = 42$	t_{no} in s	Nodes	Backtracks	Fails
Original	5.337	199283	398568	199284
Regular	1.351	43911	87823	43912
Improvement	3.950	4.538	4.538	4.538

Tab. 4: Statistics for $n = 6 * 7 = 42$

By t_i we represent the time in seconds needed to compute i solutions of the corresponding CSP. Thus, t_1 is the time to find a first solution, t_{100} to find the first 100 solutions (depending on selection strategies). For values of i which are different from 1 or multiples of 100, i represents the maximum number of solutions. “Nodes” represents the number of nodes in the completely expanded search tree (i. e. for the last t_i in the resp. tables), “Backtracks” and “Fails” the numbers of backtracking steps and failing leafs, resp., during this search. The values for t_{no} represent time needed to detect that no solution exists.

It can be seen that the *regular* approach is in average always faster due to a lower number of nodes, fails, and backtracks to be investigated during search. Furthermore, and not to be seen here, not only the average values over all search strategies were better resp. times were faster, but this also holds for every single *regular* search wrt. its *original* version of the CSP.

Table 6 shows the number of solutions found in a restricted time of $300s$ for the four search strategies (cf. Table 1). Again, the *regular* approach is superior, while the scattering can be quite large. However, during our investigations it became obvious that in cases where higher numbers of solutions are requested, the *regular* approach is always better.

5 Conclusion

This section contains a summary of the paper, gives a conclusion and explains our next steps.

$n = 49$	t_{no} in s	Nodes	Backtracks	Fails
Original	> 600	> 26684769	> 53369496	> 26684758
Regular	215.807	11188915	22377832	11188916
Improvement	> 2.780	> 2.385	> 2.385	> 2.385

Tab. 5: Statistics for $n = 7 * 7 = 49$

$n = 56$ (300s)	Search 1	Search 2	Search 3	Search 4
Original	0	1140	25	1442
Regular	20512	1463	419	3886
Improvement	∞	1.283	16.76	2.695

Tab. 6: Number of solutions found in $300s$ for $n = 8 * 7 = 56$

5.1 Summary

In this paper we presented a new approach for the optimization of general CSPs using the regular constraint. It was shown that every CSP can be transformed into a regular CSP by transforming primitive and global constraints into regular constraints and using automata intersection and minimization. By means of a rostering example we demonstrated that this approach allows to reduce the size of the search tree and to significantly improve the solution speed.

When comparing the performance of regular CSPs in contrast to the original version of the problem, of course, also the time needed for the transformation must be taken into consideration. Here, our investigations support that our approach can be applied successfully when considering subproblems of a potentially large CSP P and, thus, subsets of its variables X. The reason is the exponential growth of the number of states of the automaton with the number of variables. Transforming only subproblems (instead of P completely) is, thus, much faster and, nevertheless, leads to a reduction of the number of constraints (also the number of backtracks) and of the redundancy which, altogether, improves the solution speed.

5.2 Future work

Future work includes finding good direct transformations for global constraints, investigating promising variable orderings to optimize the size of the DFAs and, similarly, variable and value orderings for the regular constraints, research on potential benefits from decomposition of automata, finding out about general (static) criteria to decide when to apply the approach as well as extracting promising application areas in general.

References

[Ap03] Apt, Krzysztof: Principles of Constraint Programming. Cambridge University Press, New York, NY, USA, 2003.

[Ar94] Arbeitszeitgesetz vom 6. Juni 1994 (BGBl. I S. 1170, 1171), zuletzt geändert durch Artikel 3 Absatz 6 des Gesetzes vom 20. April 2013 (BGBl. I S. 868). http://www.gesetze-im-internet.de/arbzg/BJNR117100994.html, 1994. last visited 2017-05-10.

[Bu13] Bundesanstalt für Arbeitsschutz und Arbeitsmedizin (BAuA) - Gestaltung von Nacht- und Schichtarbeit. http://www.baua.de/de/Informationen-fuer-die-Praxis/Handlungshilfen-und-Praxisbeispiele/Arbeitszeitgestaltung/Nacht-%20und%20Schichtarbeit.html, 2013. last visited 2017-05-10.

[Ch17] Choco Solver 4.0.1. http://www.choco-solver.org/, 2017. last visited 2017-05-10.

[De03] Dechter, Rina: Constraint processing. Elsevier Morgan Kaufmann, 2003.

[HPvB04] Hellsten, Lars; Pesant, Gilles; van Beek, Peter: A Domain Consistency Algorithm for the Stretch Constraint. In: Principles and Practice of Constraint Programming, 10th International Conference, CP 2004. pp. 290–304, 2004.

[HU79] Hopcroft, John E.; Ullman, Jeffrey D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley, 1979.

[La12] Lang, Hans Werner: Algorithmen in Java - Sortieren, Textsuche, Codierung, Kryptographie. Oldenbourg Verlag, München, 3rd edition, 2012.

[Ló03] López-Ortiz, Alejandro; Quimper, Claude-Guy; Tromp, John; van Beek, Peter: A Fast and Simple Algorithm for Bounds Consistency of the AllDifferent Constraint. In (Gottlob, Georg; Walsh, Toby, eds): IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence. Morgan Kaufmann, pp. 245–250, 2003.

[Ma98] Marriott, Kim: Programming with Constraints - An Introduction. MIT Press, Cambridge, 1998.

[Pa08] Paltzer, Niko: Regular Language Membership Constraint. Seminararbeit, Universität des Saarlandes, Deutschland, 2008.

[Pe01] Pesant, Gilles: A Filtering Algorithm for the Stretch Constraint. In (Walsh, Toby, ed.): Principles and Practice of Constraint Programming, 7th International Conference, CP 2001. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 183–195, 2001.

[PFL16] Prud'homme, Charles; Fages, Jean-Guillaume; Lorca, Xavier: Choco Documentation. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2016. http://www.choco-solver.org/.

[RBW06] Rossi, Francesca; Beek, Peter van; Walsh, Toby: Handbook of Constraint Programming. Elsevier, Amsterdam, First edition, 2006.

[Ré94] Régin, Jean-Charles: A Filtering Algorithm for Constraints of Difference in CSPs. In (Hayes-Roth, Barbara; Korf, Richard E., eds): Proceedings of the 12th National Conference on Artificial Intelligence, Volume 1. AAAI Press / The MIT Press, pp. 362–367, 1994.

[vHK06] van Hoeve, Willem-Jan; Katriel, Irit: Global Constraints. In [RBW06], First edition, 2006. Chapter 6.