Gesellschaft für Informatik e.V. (GI) GI. (Hrsg.): INFORMATIK 2021,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 1107

A Survey of Constraint Transformation Methods

Sven Löffler, Ilja Becker, Franz Kroll, Petra Hofstedt[1]

Abstract: The solution performance of finite domain (FD) constraint problems can often be improved by either transforming particular constraints or sub-problems into other FD constraints like binary, table or regular membership constraints, or by transformation of the complete FD problem into an equivalent problem but of another domain, e.g. in a SAT problem. Specialized constraint solvers (like binary or SAT solvers) can outperform general constraint solvers for certain problems. However, this comes with high efforts for the transformation and/or other disadvantages such as a restricted set of constraints such specialized solvers can handle or limitations on the variables domains. In this paper we give an overview of CSP and constraint transformations and discuss applicabilibty and advantages and disadvantages of these approaches.

Keywords: Constraint Programming; CSP; Refinement; Optimization; Regular Membership Constraint; Regular CSPs; Table Constraint; SAT; Binary Constraint

1 Introduction

Constraint programming (CP) is a powerful method to model and solve NP-complete problems in a declarative way. Typical applications of CP are among others rostering, graph coloring, optimization, resource management, planning, scheduling and satisfiability (SAT) problems [Ma98].

Because the search space of constraint satisfaction problems (CSPs) and constraint satisfaction optimization problems (CSOPs or COPs) is immensely big and the solution process often needs an extremely large amount of time we are always interested in improving the solution process. In practice there are often various ways to describe a CSP and consequently the problem can be modeled by different combinations of constraints, which results in the differences in resolution speed and behavior. For example, there is the possibility to represent a CSP with constraints, which are of the same kind. Thus we have the possibility to use a solver, solver settings or search strategies which are optimized for the used constraints.

The rest of this paper is structured as follows. In Section 2, we introduce the necessary definitions of constraint programming. In Sections 3, 4, and 5 we give a survey of existing transformations of a general CSPs into binary, boolean, and table and regular CSPs. Furthermore, in each of these three sections we show examples of transformations and discuss advantages and disadvantages. Finally, in Section 6 we summarise and explain future steps in our researches.

[1] Brandenburg University of Technology Cottbus-Senftenberg, Germany
Sven.Loeffler@b-tu.de, Ilja.Becker@b-tu.de, Franz.Kroll@b-tu.de, Hofstedt@b-tu.de

2 Preliminaries

In this section we introduce basic definitions and concepts of constraint programming (CP) and present the relevant constraints, which are used in the rest of the paper. We consider *CSPs*, which are defined in the following way.

A *constraint satisfaction problem (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 constraints. At this, constraint $c_j = (X_j, R_j)$ is a relation R_j, which is defined over a set of variables $X_j \subseteq X$ [De03b; RBW06].

The *scope* of a constraint $c_j = (X_j, R_j)$ indicates the set of variables, which is covered by the constraint c_j: $scope(c_j) = X_j$ [De03b].

The relation R of a constraint $c = (X, R)$ represents a subset of the Cartesian product of the domain values $D_1 \times ... \times D_n$ of the corresponding variables $X = \{x_1, ..., x_n\}$. This can be expressed implicitely by a mathematical formula or (for FD constraints) explicitely by enumeration of the allowed tuples of domain values. In the following, we will use the explicit representation of constraints, i.e. by sets T of tuples of an ordered set of variables X. A CSP can only have a finite number of solutions because the number n of variables and the domain sizes are finite. Thus, it is possible to finitely enumerate all solutions of a CSP

Finally, we introduce two definitions of constraints relevant in this paper. Let a CSP $P = (X, D, C)$ and a subset X' of variables X of the CSP P be given.

For an ordered subset of variables $X' = \{x_1, \ldots, x_n\} \subseteq X$, a positive $table$ constraint $table(X', T)$ restricts any solution of the CSP to be compatible to the one of the given domain tuples of T. (For a negative constraint $table_n(X', T)$ a solution must be incompatible to all the given tuples in T.)

We use the notation of a $regular$ constraint as a synonym for $regular\ membership$ respectively $regular\ language\ membership$ constraint. The $regular$ constraint and its propagation [HPB04; Pe01; Pe04] is based on deterministic finite automatons (DFAs) [HU79]. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA, let $X' = \{x_1, \ldots, x_n\} \subseteq X$ be an ordered 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 constraint $regular(X', M)$ defines the allowed domain value tupes as:

$$\{(w_1, \ldots, w_n) \mid \forall i \in \{1, \ldots, n\}, w_i \in D_i,\ (w_1 w_2 \ldots w_n) \in L(M)\}\text{[HK06]}$$

So, the concatenation of the values w_i of the variables $x_i, \forall i \in \{1, ..., n\}$ must be accepted by the automaton M.

Based on the definition of the table and the regular constraint, we define a *table CSP* respectively a *regular CSP* as a CSP which contains only table or regular constraints.

Furthermore, there are *binary CSPs* which contain only constraints covering two or less variables and *boolean CSPs* (*SAT problems*) which contain only boolean variables and logical clauses of the form $(\bigvee_j (\neg) x_{ij})$ as constraints.

3 Transformations into Binary CSPs

Two common possibilities to transform FD-CSPs into binary CSPs are the dual [DP89] and the hidden transformations [Pe60; RPD90; SF94]. The goal of these transformations is to receive binary CSPs, i.e CSPs to which algorithms like path consistency (PC-1 und PC-2 in [De03a]) can be applied. Establishing path consistency is possible in polynomial time, but depending on the given CSP much transformation time is needed.

3.1 Dual Transformation

For the dual transformation of a CSP $P = (X, D, C)$ into a dual CSP $P^{dual} = (X^d, D^d, C^d)$ for each constraint $c_i = (X_i, T_i) \in C$ of the original CSP P a new dual variable $x_i^d \in X^d$ is created. The domain of each such variable contains all allowed tuples T_i of the corresponding constraint c_i. For each pair of original constraints c_i and $c_j \in C$, which cover at least one shared variable, a new binary dual constraint $c_{i,j}^d = (\{x_i^d, x_j^d\}, R_{i,j}^d) \in C^d$ is created. The relation $R_{i,j}^d$ defines all allowed tuples, which satisfy c_i and c_j.

It follows a formal definition of the dual transformation and an example, which illustrates the transformation of a CSP into a dual CSP.

Definition 1 (Dual transformation)**.** *The dual transformation of a CSP $P = (X, D, C)$ is defined by $P^{dual} = (X^d, D^d, C^d)$, with:*

$X^d = \{x_1^d, ..., x_m^d\}$ is a set of dual variables, where each variable x_i^d represents a constraint $c_i \in C$ of P.

$D^d = \{D_1^d, ..., D_m^d\}$, with $D_i^d = T_i$ is the set of domains for the dual variables. For every dual variable $x_i^d \in X^d$ holds $D_i^d = T_i$, where T_i is the list of admissible tuples of $c_i = (X_i, R_i)$.

C^d is a set of dual constraints over the dual variables. For each pair of constraints $c_i, c_j \in C$ with $c_i \neq c_j$ and $scope(c_i) \cap scope(c_j) = X_{i,j} \neq \emptyset$ of P a dual constraint $c_{i,j}^d = (\{x_i^d, x_j^d\}, T_{i,j}^d) \in C^d$ is created. Each tuple $(a_i, b_j) \in T_{i,j}^{dual}$ contains a tuple $a_i \in T_i$ and a tuple $b_j \in T_j$, which have equal projections to their shared variables [Ba02].

Example 1 (Dual transformation)**.** Given is the CSP P_1 $P_1 = (X, D, C)$ with

$$
\begin{aligned}
X &= \{x_1, x_2, x_3, x_4\} \\
D &= \{D_1 = \{0,1\}, D_2 = \{0,1\}, D_3 = \{0,1,2\}, D_4 = \{0,1,2\}\} \\
C &= \{c_1 = (x_1 \neq x_2),\ c_2 = (x_1 < x_3),\ c_3 = count(\{x_1, x_2, x_3, x_4\}, 2, 1)\}
\end{aligned}
$$

The *count*-constraint c_3 demands that the value 1 is assigned to 2 of the variables $x_1, \ldots, x_4$.

When tranforming P_1 into a dual CSP, for each constraint $c_1, c_2, c_3 \in C$ a dual variable $X^d = \{x_1^d, x_2^d, x_3^d\}$ is created. The allowed tuples of c_1, c_2 and c_3 are enumerated for the domains of x_1^d, x_2^d, and x_3^d. For example $D_1^d = \{(0,1), (1,0)\}$.

Each pair $c_i, c_j \in C$ with at least one shared variable yields a new binary dual constraint $c_{i,j}^d$. For example we get $c_{1,2}^{dual} = (\{x_1^{dual}, x_2^{dual}\}, T_{1,2}^{dual})$, where tuple list $T_{1,2}^d$ represents the projection of the allowed tuples of c_1 and c_2, such that the shared variable x_1 has the same value: $T_{1,2}^d = \{((0,1),(0,1)),\ ((0,1),(0,2)),\ ((1,0),(1,2))\}$. It follows an excerpt of the resulting dual CSP $P^{dual} = (X^d, D^d, C^d)$ with

$$
\begin{aligned}
X^d &= \{x_1^d, x_2^d, x_3^d\} \\
D^d &= \{D_1^d = \{(0,1),\ (1,0)\}, \\
&\quad\ D_2^d = \{(0,1),\ (0,2),\ (1,2)\}, \\
&\quad\ D_3^d = \{(0,0,1,1),\ (0,1,0,1),\ (0,1,2,1),\ (0,1,1,0), ..., \}\} \\
C^d &= \{c_{1,2}^d = (\{x_1^d, x_2^d\}, \{((0,1),(0,1)),\ ((0,1),(0,2)),\ ((1,0),(1,2))\}), \\
&\quad\ c_{1,3}^d = (\{x_1^d, x_3^d\}, \{((0,1),(0,1,0,1)),\ ((0,1),(0,1,2,1)), \\
&\qquad\qquad ((0,1),(0,1,1,0)), ..., ((1,0),(1,0,1,2)), ...\}) \\
&\quad\ c_{2,3}^d = (\{x_2^d, x_3^d\}, \{((0,1),(0,0,1,1)),\ ((0,1),(0,1,1,0)), \\
&\qquad\qquad ((0,1),(0,1,1,2)), ((0,2),(0,1,2,1)), ...\})\}
\end{aligned}
$$

3.2 Hidden Transformation

Regarding the hidden transformation of a CSP $P = (X, D, C)$ into a hidden CSP $P^{hidden} = (X \cup X^h, D \cup D^h, C^h)$, the hidden variables X^h and their domains are created analogously to the dual variables in the above transformation. The hidden constraints C^h are binary constraints, each between an original variable $x_i \in X$ and a hidden variable $x_j^h \in X^h$ such that the constraint $c_j \in C$, now represented by the hidden variable x_j^h, contains the variable x_i, i.e. $x_i \in scope(c_j)$. In this way, a bipartite graph with the two sets of nodes X and X^h is created. The hidden constraints C^h guarantee that every variable is only assigned values, which fulfill the initial constraints, in which this variable involved.

Definition 2 (Hidden transformation)**.** *Given a CSP $P = (X, D, C)$, the hidden transformation yields a CSP $P^{hidden} = (X^h \cup X, D^h \cup D, C^h)$, where X^h and D^h are created as X^d and D^d by the dual transformation. C^h is the set of binary hidden constraints, where each connects a hidden variable $x^h \in X^h$ with a variable $x \in X$ of the initial CSP P.*

For every hidden variable $x_j^h \in X^h$ and for every initial variable $x_i \in X$, which are connected via the initial constraint $c_j = (X_j, T_j)$ ($x_i \in scope(c_j)$), a hidden constraint $c_{i,j}^h \in C^h$ exists in C^h. Any such hidden constraint $c_{i,j}^h = (\{x_i, x_j^h\}, R_{i,j})$ specifies the admissible tuples $t \in T_j$ when a value a is assigned to the variable x_i ($R_{i,j} := t[x_i] = a$) [Ba02].

Example 2 (Hidden transformation)**.** Given is again the CSP P_1, which is supposed to be transformed into a hidden CSP. For each of the three constraints c_1, c_2 and c_3 a hidden variable $X^h = \{x_1^h, x_2^h, x_3^h\}$ and domain $D^h = \{D_1^h, D_2^h, D_3^h\}$ is created as by the dual transformation.

A new, binary hidden constraint $c_{i,j}^h$ must be generated for every pair of variables $x_i \in X$ and $x_j^h \in X^h$ with $x_i \in scope(c_j)$. We get the following constraints: $c_{1,1}^h = (\{x_1, x_1^h\}, T_{1,1}^h), c_{2,1}^h = (\{x_2, x_1^h\}, T_{2,1}^h)$, $c_{1,2}^h = (\{x_1, x_2^h\}, T_{1,2}^h)$, $c_{3,2}^h = (\{x_3, x_2^h\}, T_{3,2}^h)$, $c_{1,3}^h = (\{x_1, x_3^h\}, T_{1,3}^h)$, $c_{2,3}^h = (\{x_2, x_3^h\}, T_{2,3}^h)$, $c_{3,3}^h = (\{x_3, x_3^h\}, T_{3,3}^h)$ and $c_{4,3}^h = (\{x_4, x_3^h\}, T_{4,3}^h)$.

The lists of tuples $T_{i,j}^h$ result as given above, e.g. $T_{1,1}^h = \{(0, (0, 1)),\ (1, (1, 0))\}$ associates with the first tuple the value 0 for x_1 with the valuation $(0, 1)$ for constraint c_1 (and 1 for x_1 with $(1, 0)$ for c_1 with the second tuple). This results in a new hidden CSP $P^{hidden} = (X \cup X^h, D \cup D^h, C^h)$ with:

$$
\begin{array}{lcl}
X \cup X^h & = & \{x_1, x_2, x_3, x_4, x_1^h, x_2^h, x_3^h\} \\
D \cup D^h & = & \{D_1 = \{0, 1\},\ D_2 = \{0, 1\},\ D_3 = D_4 = \{0, 1, 2\}, \\
 & & D_1^h = \{(0, 1),\ (1, 0)\},\ D_2^h = \{(0, 1),\ (0, 2),\ (1, 2)\}, \\
 & & D_3^d = \{(0, 0, 1, 1),\ (0, 1, 0, 1),\ (0, 1, 2, 1),\ (0, 1, 1, 0), ..., \}\} \\
C^h & = & \{c_{1,1}^h = (\{x_1, x_1^h\}, T_{1,1}^h\}),\ c_{2,1}^h = (\{x_2, x_1^h\}, T_{2,1}^h\}), \\
 & & c_{1,2}^h = (\{x_1, x_2^h\}, T_{1,2}^h\}),\ c_{3,2}^h = (\{x_3, x_2^h\}, T_{3,2}^h\}), ...\}
\end{array}
$$

where $T_{1,1}^h = \{(0, (0, 1)),\ (1, (1, 0))\}, T_{2,1}^h = \{(0, (1, 0)),\ (1, (0, 1))\}$,
$T_{1,2}^h = \{(0, (0, 1)),\ (0, (0, 2)),\ (1, (1, 2))\}, T_{3,2}^h = \{(1, (0, 1)),\ (2, (0, 2)),\ (2, (1, 2))\}$, ...

Advantages and Disadvantages of Binary Transformations. Both approaches have the *advantage*, that they allow the use of algorithms which require binary constraints, e.g. path consistency algorithms (PC-1 and PC-2 in [De03a]). On the other side both approaches have the *disadvantages*, that they are only reasonable when the whole CSP is transformed and they need to list all solutions of all constraints, which is mostly extremely time consuming.

The hidden transformation has the advantage over the dual transformation that the original variables remain. This allows a direct read off a solution, no variable transformation from hidden variables is necessary. The advantage of the dual transformation is that it needs less variables and constraints than the hidden transformation, in most cases.

4 Transformations into SAT Problems

SAT problems are capable of representing every other FD-CSP. This requires the transformation of all non-boolean variables into boolean ones, as well as a translation of all constraints based upon these new variables. The main advantage of this approach is the ability to leverage the power of modern SAT solvers. A variety of algorithms for the transformation of FD-CSPs into boolean CSPs is desribed in [Ga07; Pe15a; Wa00]. Following we describe some exemplary selected methods.

4.1 Direct Encoding

The direct encoding approach creates a new variable for each possible variable assignment. Depending on whether an original variable is assigned a certain value, the corresponding value variable is $True$ (1) or not. For each illegal assignment of values to the variables X_i according to a constraint $c_i = (X_i, R_i)$ a clause over the previously created boolean values is generated, that rules out this assignment. Additional constraints are introduced that ensure, that only one value assignment representing variable per original variable can be true at a time.

Definition 3 (Direct encoding)**.** *Given a CSP $P = (X, D, C)$ we define its direct encoding to be $P^{de} = (X^{de}, D^{de}, C^{de})$ where:*

$X^{de} = \{x^{de}_{i,j} \mid i \in \{1, ..., |X|\}, j \in \{1, ..., |D_i|\}\}$, is a set of boolean variables where each variable $x^{de}_{i,j}$ represents, whether the original variable x_i is assigned the j-th value of the domain D_i or not ($x_i = d_j \leftrightarrow x^{de}_{i,j} = 1$).

$D^{de} = \{D^{de}_{i,j} = \{0, 1\} \mid i \in \{1, ..., |X|\}, j \in \{1, ..., |D_i|\}\}$ is the set of the boolean domains of the newly created variables.

C^{de} is the set of constraints over the newly created boolean variables X^{de}. For each constraint $c = (X, T) \in C$ of the original CSP P, with $X = \{x_1, ..., x_n\}$, a constraint $c^{de} \in C^{de}$ can be given in boolean form, that is equivalent to c. For each viable tuple $t_l = (v_{l,1}, ..., v_{l,n}) \notin T$ of the constraint c a clause $\neg x^{de}_{1,v_{l,1}} \vee \neg x^{de}_{2,v_{l,2}} \vee ... \vee \neg x^{de}_{n,v_{l,n}}$ is generated.

To ensure that the variables $X^{de}_i = \{x^{de}_{i,j} \mid \forall j \in \{1, ..., |D_i|\}\}$ represent the possible variable assignments for x_i, one needs to ensure that exactly one variable in X^{de}_i becomes true (1). To that for each variable x_i so-called $addLeastOne$ clauses $\bigvee_{v \in D_i} x^{de}_{i,v}$, as well as $atMostOne$ clauses $\neg x^{de}_{i,v_1} \vee \neg x^{de}_{i,v_2}, \forall v_1, v_2 \in D_i$ with $i \neq j$ are added [Pr09].

Example 3 (Direct encoding)**.** The CSP P_1 is transformed into a boolean CSP using direct encoding. For each variable x_1, x_2, x_3 and x_4 and for each of its domain values a new boolean variable must be created.

For each invalid tuple $t_i \notin T_i$ of each constraint $c_i, i \in \{1,2,3\}$ a clause is generated that excludes this tuple. From the invalid tuples $(0,0)$ and $(1,1)$ for c_1 therefore follow the clauses $c_{1,1} = \neg x^{de}_{1,0} \vee \neg x^{de}_{2,0}$ and $c_{1,2} = \neg x^{de}_{1,1} \vee \neg x^{de}_{2,1}$. The other constraints are transformed analogously.

Finally the $addLeastOne$ and $atMostOne$ constraints for the variables $x_1, \ldots, x_4$ need to be created. E.g. for the original variable $x_1 \in X$ the resulting constraints are $addLeastOne(x_1) = (x^{de}_{1,0} \vee x^{de}_{1,1})$ and $atMostOne(x_1) = \{(\neg x^{de}_{1,0} \vee \neg x^{de}_{1,1})\}$.

The direct encoding of CSP P_1 leads to the boolean CSP $P^{de} = (X^{de}, D^{de}, C^{de})$ where:

$$
\begin{aligned}
X^{de} &= \{x^{de}_{1,0}, x^{de}_{1,1}, x^{de}_{2,0}, x^{de}_{2,1}, x^{de}_{3,0}, x^{de}_{3,1}, x^{de}_{3,2}, x^{de}_{4,0}, x^{de}_{4,1}, x^{de}_{4,2}\} \\
D^{de} &= \{D^{de}_{1,0} = D^{de}_{1,1} = D^{de}_{2,0} = \ldots = D^{de}_{4,0} = D^{de}_{4,1} = D^{de}_{4,2} = \{0,1\}\} \\
C^{de} &= \{c_{1,1} = (\neg x^{de}_{1,0} \vee \neg x^{de}_{2,0}),\ c_{1,2} = (\neg x^{de}_{1,1} \vee \neg x^{de}_{2,1}) \\
&\quad c_{2,1} = (\neg x^{de}_{1,0} \vee \neg x^{de}_{3,0}),\ c_{2,2} = (\neg x^{de}_{1,1} \vee \neg x^{de}_{3,0}),\ c_{2,3} = (\neg x^{de}_{1,1} \vee \neg x^{de}_{3,1}), \\
&\quad c_{3,1} = (\neg x^{de}_{1,0} \vee \neg x^{de}_{2,0} \vee \neg x^{de}_{3,0} \vee \neg x^{de}_{4,0}), \ldots\}, \\
&\quad addLeastOne: \\
&\qquad \cup \{(x^{de}_{1,0} \vee x^{de}_{1,1}), (x^{de}_{2,0} \vee x^{de}_{2,1}), (x^{de}_{3,0} \vee x^{de}_{3,1} \vee x^{de}_{3,2}), (x^{de}_{4,0} \vee x^{de}_{4,1} \vee x^{de}_{4,2})\} \\
&\quad atMostOne: \\
&\qquad \cup \{(\neg x^{de}_{1,0} \vee \neg x^{de}_{1,1}), (\neg x^{de}_{2,0} \vee \neg x^{de}_{2,1}), (\neg x^{de}_{3,0} \vee \neg x^{de}_{3,1}), (\neg x^{de}_{3,0} \vee \neg x^{de}_{3,2}), \\
&\qquad\quad (\neg x^{de}_{3,0} \vee \neg x^{de}_{3,3}), (\neg x^{de}_{3,1} \vee \neg x^{de}_{3,2}), (\neg x^{de}_{3,1} \vee \neg x^{de}_{3,3}), (\neg x^{de}_{3,2} \vee \neg x^{de}_{3,3}), \ldots\}
\end{aligned}
$$

4.2 Logarithmic Encoding

The idea behind logarithmic encoding is to encode the index of an assigned value for a variable in a binary representation with boolean variables. For each variable $x_i \in X$ of the original CSP P, exactly $r = \lceil log_2 |D_i| \rceil$ variables $x^{le}_{i,r}, ..., x^{le}_{i,1}$ are created, where the assignment for the variables $x^{le}_{i,r}, ..., x^{le}_{i,1}$ represents the index of a value in the domain D_i of the original variable x_i in binary form.

Definition 4 (Logarithmic encoding)**.** *Given a CSP $P = (X, D, C)$ we define its logarithmic encoding to be $P^{le} = (X^{le}, D^{le}, C^{le})$ where:*

$X^{le} = \{x^{le}_{i,j} \mid i \in \{1, ..., |X|\}, j \in \{1, .., r\}\}$ with $r = \lceil log_2 |D_i| \rceil$ is a set of boolean variables where each series of variables $x^{le}_{i,r}, ..., x^{le}_{i,1}$ represents, that the original variable x_i is assigned the k-th value of the domain D_i. The assignments $b_{i,r} ... b_{i,1}$ of the variables $x^{le}_{i,r}, ..., x_{i,1}$ correspond to the binary representation of k.

$D^{le} = \{D_{i,j} = \{0,1\} \mid i \in \{1, ..., |X|\}, j \in \{1, ..., \lceil log_2 |D_i| \rceil\}\}$ is the set of the boolean domains of the newly created variables.

C^{le} is the set of constraints over the newly created boolean variables X^{le}. For each original constraint $c = (X, T) \in C$ with $X = \{x_1, ..., x_n\}$, a constraint $c^{le} \in C^{le}$ can be given in

boolean form that is equivalent to c. For each tuple $t_l = (v_{l,1}, ..., v_{l,n}) \notin T$, that violates a constraint c, the following clause is created:

$$c_l^{le} = \vee_{i \in \{1,...,n\}, j \in \{\lceil log_2 |D_i| \rceil, ..., 1\}} \begin{cases} \neg x_{i,j}^{le} & b_{i,j} = 1 \\ x_{i,j}^{le} & , otherwise \end{cases}$$

The values $b_{i,r}...b_{i,1}$ encode the binary representation of the index of the value $v_{l,i}$ in D_i.

Additionally one needs to ensure that for each domain whos size is not divisible by two, the binary representations of indices that do not correspond to values in the domain are invalid. For this each of these indices can be represented by a one-dimensional, negative tuple and be transformed into a clause accordingly [Ga07; IM94; Wa00].

Example 4 (Logarithmic encoding)**.** Following the CSP P_1 will be transformed into a boolean CSP using logarithmic encoding. For the variables x_1 and x_2, whose domain sizes are two, it suffices to create new variables $x_{1,0}^{le}$ and $x_{2,0}^{le}$ respectively. For the variables x_3 and x_4 two variables $x_{3,1}^{le}$ and $x_{3,0}^{le}$, and $x_{4,1}^{le}$ and $x_{4,0}^{le}$ respectively, are created accordingly to their domain size of three. The domains of all created variables equal $\{0, 1\}$. Because the original variables x_3 and x_4 cannot be assigned to value 3, it is necessary to add the two negative tuple clauses $c_{3,3}^n = \neg x_{3,0}^{le} \vee \neg x_{3,1}^{le}$, $c_{4,3}^n = \neg x_{4,0}^{le} \vee \neg x_{4,1}^{le}$.

For each of the three constraints c_1, c_2 and c_3 a boolean constraint based on their respective negative tuple lists is created. For c_1 we get the negative tuple list $T_1' = \{(0, 0),\ (1, 1)\}$, from which the constraints $c_{1,1}^{le} = x_{1,0}^{le} \vee x_{2,0}^{le}$ and $c_{1,2}^{le} = \neg x_{1,0}^{le} \vee \neg x_{2,0}^{le}$ are derived. The other constraints can be obtained analogously. Following is an excerpt of the transformed CSP $P^{le} = (X^{le}, D^{le}, C^{le})$ with:

$$\begin{aligned} X^{le} &= \{x_{1,0}^{le}, x_{2,0}^{le}, x_{3,1}^{le}, x_{3,0}^{le}, x_{4,1}^{le}, x_{4,0}^{le}\} \\ D^{le} &= \{D_{1,0}^{le} = D_{2,0}^{le} = D_{3,1}^{le} = D_{3,0}^{le} = D_{4,1}^{le} = D_{4,0}^{le} = \{0, 1\}\} \\ C^{le} &= \{(c_{3,3}^n = \neg x_{3,0}^{le} \vee \neg x_{3,1}^{le}), (c_{4,3}^n = \neg x_{4,0}^{le} \vee \neg x_{4,1}^{le}), \\ &\quad (c_{1,1}^{le} = x_{1,0}^{le} \vee x_{2,0}^{le}), (c_{1,2}^{le} = \neg x_{1,0}^{le} \vee \neg x_{2,0}^{le}) \\ &\quad (c_{2,1}^{le} = x_{1,0}^{le} \vee x_{3,1}^{le} \vee x_{3,0}^{le}),\ (c_{2,2}^{le} = \neg x_{1,0}^{le} \vee x_{3,1}^{le} \vee \neg x_{3,0}^{le}), ...\}, \end{aligned}$$

4.3 Support Encoding

The support encoding approach utilizes the same mechanism as direct encoding for transforming variables and domains, but models the original constraints with clauses that represent the valid tuples. Further more, support encoding is only applicable on binary CSPs. After the definition follows an example that shows the application of the support encoding on the dual CSP P^{dual} of P_1.

Definition 5 (Support encoding)**.** *Given a CSP $P = (X, D, C)$ we define its support encoding to be $P^{se} = (X^{se}, D^{se}, C^{se})$ where X^{se} and D^{se} are created like X^{de} and D^{de} in the direct encoding. C^{se} is the set of constraints over the newly created boolean variables X^{se}.*

For each pair of variables $x_i, x_j \in X$ with $i \neq j$ of each original constraint $c = (X, T)$, $X = \{x_1, ..., x_n\}$, $(x_i, x_j \in scope(c) = X)$ and each value $v \in D_i$ a new clause $\neg x_{i,v} \vee \bigvee_{w \in A} x_{j,w}$ is created. The set $A \subseteq D_j$ contains all values $v_k \in D_j$ which x_j can be instantiated to, such that at least one assignment $x_i = v$ and $x_j = v_k$ exists and satisfies c.

For each original variable $x_i \in X$ also the $addLeastOne$ $\bigvee_{v \in D_i} x^{de}_{i,v}$ and $atMostOne$ clauses $\neg x^{de}_{i,v} \vee \neg x^{de}_{i,w}, \forall v, w \in D_i\ i \neq j$ must be added too [Ge02].

Example 5 (Support encoding)**.** Given is the binary CSP P^{dual}. First the variables X^d are transformed into boolean variables X^{se}. The resulting variables are $X^{se} = \{x^{se}_{1,1}, x^{se}_{1,2}, x^{se}_{2,1}, x^{se}_{2,2}, x^{se}_{2,3}, x^{se}_{3,1}, x^{se}_{3,2}, ...\}$, where each assignment of 1 to a variable $x^{se}_{i,j}$ represents, that the dual variables x^d_i gets assigned the j-th value of its domain. Thus, $x^{se}_{3,2}$ being assigned the value 1 represents the dual variable x^d_3 being assigned the tuple $(0, 1, 0, 1)$.

For each variable $x^d \in X^d$ of the dual problem the $addLeastOne$ and $atMostOne$ clauses are created. For the variable x^d_2 the resulting clauses are $x^{se}_{2,1} \vee x^{se}_{2,2} \vee x^{se}_{2,3}$, as well as $\neg x^{se}_{2,1} \vee \neg x^{se}_{2,2}$, $\neg x^{se}_{2,1} \vee \neg x^{se}_{2,3}$ and $\neg x^{se}_{2,2} \vee \neg x^{se}_{2,3}$. The clauses for x^d_1 and x^d_3 are created analogously.

Finally the constraints C^d need to be transformed. The original domains $D^d_i \in D^d$ are replaced by the domains $D'_i \in D'$, which only contain the indices of the domain values instead of the actual domain values ($D'_i = \{0, 1, ..., |D^d_i| - 1\}, \forall i \in \{1, ..., 3\}$. The constraint $c^d_{1,2}$ is transformed exemplary as follows. For each value $v \in D'_1$ of the dual value pair x^d_1, x^d_2 the clause $\neg x^{se}_{1,v} \vee \bigvee_{w \in A} x^{se}_{1,w}$ is created. The resulting clauses are $c_{1,1} = \neg x^{se}_{1,0} \vee x^{se}_{2,0} \vee x^{se}_{2,1}$ and $c_{1,2} = \neg x^{se}_{1,1} \vee x^{se}_{2,2}$. Since both directions need to be taken into account, we also need to generate clauses for the variable pair x^d_2, x^d_1. This gives the clauses $c_{1,3} = \neg x_{2,0} \vee x_{1,0}$, $c_{1,4} = \neg x_{2,1} \vee x_{1,0}$ and $c_{1,5} = \neg x_{2,2} \vee x_{1,1}$.

4.4 More SAT Encodings

Additionally to the previously discussed encoding mechanisms there are more known transformations, that cannot be discussed in detail here. Following we provide brief summaries as well as references.

Minimal Support Encoding. The minimal support encoding was defined in [Ar08] and transforms a CSP into a SAT problem. Generally, the approach follows the methods of support encoding, the difference being that for each constraint c over the variables $scope(c) = \{x_1, x_2\}$ the support clauses are only given for one variable, x_1 or x_2.

Regular Encoding. Another method to transform binary CSPs into boolean CSPs is regular encoding [AM04]. The main concept behind regular encoding is to replace each variable $x_i \in X$ of the original CSP with its domain being $D_i = \{0, ..., m\}$ with a unary vector of boolean variables $\overrightarrow{U_i} = [x^{re}_{i,1}, ..., x^{re}_{i,m}]$. Each vector $\overrightarrow{U_i}$ is restricted so that a variable $x^{re}_{i,j}$ can only become 1 if all variables $x^{re}_{i,k}$ with $k < j$ also become 1. Each of these variables $x^{re}_{i,j}$ is called a regular variable.

Therefore each vector U_i takes the form $[1, 1, ..., 1, *, *, *, ..., *, 0, 0]$, where 1 represents variables that are assigned the value 1, $*$ represents variables that weren't assigned a value yet, and 0 represents variables that were assigned 0. The number of 1s in $\overrightarrow{U_i}$ needs to be equal to $d_i \in D_i$, which corresponds to the variable x_i.

A survey of the previously discussed encodings and further methods can be found in Chapter 4 of "Bridging Constraint Satisfaction and Boolean Satisfiability" [Pe15b].

Advantages and Disadvantages of SAT Transformations. All introduced SAT transformations have the *advantage*, that they allow the use of very well researched and fast SAT solvers. Also all transformations have the *disadvantage*, that they need much time for the transformation. This follows from the fact that in all approaches either all allowed tuples or all disallowed tuples must be processed. Even though a lot of information about the original CSP instance is usually lost during the translation stage and a large set of propositional clauses is produced, often needing much time in the process, SAT solvers sometimes outperform conventional CSP solvers [Pe15b].

It follows a discussion of the differences of the introduced transformations. Comparing support encoding with minimal support encoding, the first needs more constraints than the latter, but reaches arc-consistency if unit propagation is enforced on the transformed CSP. Logarithmic encoding needs less boolean variables than the other approaches, which is effectively noticeable if the domain sizes of the original CSP are very big. On the other hand, logarithmic encoding needs many more literals in the clauses, which slows down the solving speed of the SAT solver. The direct encoding approach and support encoding are usually inefficient in practice, since they often produce very large sets of clauses [Pe15b]. Thus, sometimes it is promising to use regular encoding which can represent equalities and inequalities with less clauses than the previously mentioned methods.

On the other hand, direct and logarithmic encoding both need to handle every tuple which does not satisfy a constraint, while support encoding needs to handle every tuple which does satisfy a constraint. Thus, direct and logarithmic encoding are more promising if the number of satisfying tuples of the constraints is small, while support encoding looks promising if the number of satisfying tuples of the constraints is small.

In contrast to support encoding, minimal support encoding does not reach arc-consistency if unit propagation is enforced. On the other hand, it can reduce the number of clauses. In the

end, the decision which transformation is the best one for a CSP depends on the structure, properties and connections of the used constraints.

5 Transformations into table and regular CSPs

For the transformation of a CSP $P = (X, D, C)$ into a table CSP or regular CSP every admissible tuple of each constraint must be listed and transformed. However, an important difference to the previous approaches is that now a constraint or subset of constraints is substituted by one single constraint, which reaches *generalized arc consistency (GAC)*. This can strengthen the propagation without changing the underlying solver.

The transformation of CSPs into table CSPs is called **tabulation**. Let be T_j the tuple list, which contains the tuples which satisfy the constraint $c_j = (X_j, T_j) \in C$. The constraint $table(X_j, T_j)$ (cf. Sect. 2) is the tabulation of the constraint c_j. So you can transform all constraints in C to receive a table CSP [Ge07].

The transformation of CSPs into regular CSPs is called **regularization**. Analogously to tabulation, let be T_j the tuple list, which contains the tuples which satisfy the constraint $c_j = (X_j, T_j) \in C$. A deterministic finite automaton (DFA) M_j can be created from the tuple list T_j. The constraint $regular(X_j, M_j)$ (agan cf. Sect. 2) is the regularization of the constraint c_j. So you can transform all constraints in C to receive a regular CSP [LLH19].

Advantages and Disadvantages of table and regular Transformations. Both approaches have the *advantage*, that no prior transformations are needed. Because the table and regular constraint both reach generalized arc consistency (GAC), the regularization and the tabulation can increase the consistency level, if the constraint(s) to be substituted has/have a lower consistency level. Further advantages are, that no variables or domains must be transferred and it is not necessary to transform the whole CSP (partial transformations are possible). Thus, slow parts of a CSP can be transformed selectively. If the complete CSP is transformed a specialized table or regular solver can be used.

But both approaches in their basic version have the same disadvantage (as well as the aforementioned approaches), as all solutions of all constraints must be listed, which can be very time consuming. The regularization has one big advantage over every other here discussed transformation approach: For many global constraints there are direct transformations into regular constraints. For example the count constraint in our example CSP P_1 can be substituted directly by an DFA M_3 as represented in Figure 1, which can be created as explained in [De15; LLH18]. Using direct transformations allows us to avoid the enumeration of all admissible tuples of a constraint and, thus, leads to much faster transformation.

Furthermore, when comparing tabulation with regularization, tabulation has the advantage that a tabular constraint propagates faster if regularization can not find a small and compact

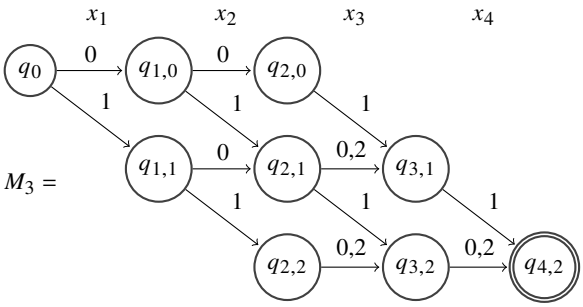

Fig. 1: The DFA M_3 which is equivalent to the count constraint c_3 of the original CSP P_1.

DFA. On the other hand, if regularization finds a small and compact DFA, then regularization is applicable for more and bigger CSPs and propagates faster than a tabulated CSP.

6 Summary and Future Work

We presented and compared different transformations for the conversion of arbitrary CSPs into special CSPs, illustrated these by examples and discussed advantages and disadvantages.

The use of transformations usually leads to an acceleration of the search for a solution (due to the use of specific solvers or propagation algorithms). However the time needed for the transformations must be considered as well. Often the time needed for the transformation exceeds the time savings, that specialized algorithms can achieve. However, for certain CSPs the transformations can lead to considerable time savings. The following questions arise for future research: 1) How can the transformations be sped up? 2) How can one determine ahead of time, which transformations are reasonable for a certain CSP?

The direct transformation of global constraints into respective target constraints could be a solution with regard to the first question. For the transformation into regular CSPs exist direct transformations, which avoid the processing of all valid and invalid constraints. In many cases, this can reduce the time needed for the transformation dramatically.

Machine learning could be one approach for the recognition of a suitable transformation for a given CSP (question 2). In [LBH21], we show first promising steps towards an automated prediction, as to whether the original constraint or the transformation into a regular or tabular constraint is more promising.

References

[AM04] Ansótegui, C.; Manyà, F.: Mapping Problems with Finite-Domain Variables into Problems with Boolean Variables. In: SAT 2004 - The 7th International

Conference on Theory and Applications of Satisfiability Testing, Vancouver, BC, Canada, Online Proceedings. 2004.

[Ar08] Argelich, J.; Cabiscol, A.; Lynce, I.; Manyà, F.: Encoding Max-CSP into Partial Max-SAT. In: 38th IEEE International Symposium on Multiple-Valued Logic (ISMVL), Dallas, Texas, USA. IEEE Computer Society, pp. 106–111, 2008.

[Ba02] Bacchus, F.; Chen, X.; van Beek, P.; Walsh, T.: Binary vs. non-binary constraints. Artif. Intell. 140/1/2, pp. 1–37, 2002.

[De03a] Dechter, R.: Consistency-Enforcing and Constraint Propagation. In: Constraint processing. Elsevier Morgan Kaufmann, chap. 3, pp. 51–84, 2003.

[De03b] Dechter, R.: Constraint processing. Elsevier Morgan Kaufmann, 2003.

[De15] Demassey, S.: Global Constraint Catalog, `http://sofdem.github.io/gccat/`, 07.04.2015, 2015.

[DP89] Dechter, R.; Pearl, J.: Tree Clustering for Constraint Networks. Artif. Intell. 38/3, pp. 353–366, 1989.

[Ga07] Gavanelli, M.: The Log-Support Encoding of CSP into SAT. In: Principles and Practice of Constraint Programming - CP, 13th International Conference, Providence, RI, USA, September 23-27, Proceedings. Pp. 815–822, 2007.

[Ge02] Gent, I. P.: Arc Consistency in SAT. In (van Harmelen, F., ed.): Proceedings of the 15th Eureopean Conference on Artificial Intelligence, ECAI, Lyon, France. IOS Press, pp. 121–125, 2002.

[Ge07] Gent, I. P.; Jefferson, C.; Miguel, I.; Nightingale, P.: Data Structures for Generalised Arc Consistency for Extensional Constraints. In: Twenty-Second AAAI Conference on Artificial Intelligence, Vancouver, British Columbia, Canada. AAAI Press, pp. 191–197, 2007.

[HK06] van Hoeve, W.-J.; Katriel, I.: Global Constraints. In: Handbook of Constraint Programming. First, Chapter 6, Elsevier, Amsterdam, 2006, ISBN: 978-0-080-46380-3.

[HPB04] Hellsten, L.; Pesant, G.; van Beek, P.: A Domain Consistency Algorithm for the Stretch Constraint. In (Wallace, M., ed.): Principles and Practice of Constraint Programming - CP. Vol. 3258. LNCS, Springer, pp. 290–304, 2004.

[HU79] Hopcroft, J. E.; Ullman, J. D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley, 1979, ISBN: 0-201-02988-X.

[IM94] Iwama, K.; Miyazaki, S.: SAT-Varible Complexity of Hard Combinatorial Problems. In: Technology and Foundations - Information Processing, Volume 1, IFIP 13th World Computer Congress, Hamburg, Germany, pp. 253–258, 1994.

[LBH21] Löffler, S.; Becker, I.; Hofstedt, P.: ML-based Decision Support for CSP Modelling with Regular Membership and Table Constraints. In (Rocha, A. P.; Steels, L.; van den Herik, H. J., eds.): 13th International Conference on Agents and Artificial Intelligence, ICAART, Volume 2. SCITEPRESS, 2021.

[LLH18] Löffler, S.; Liu, K.; Hofstedt, P.: The Regularization of CSPs for Rostering, Planning and Resource Management Problems. In: Artificial Intelligence Applications and Innovations - 14th IFIP WG 12.5 International Conference, AIAI 2018, Rhodes, Greece, Proceedings. Pp. 209–218, 2018.

[LLH19] Löffler, S.; Liu, K.; Hofstedt, P.: The Regularization of Small Sub-Constraint Satisfaction Problems. In (Hofstedt, P.; Abreu, S.; John, U.; Kuchen, H.; Seipel, D., eds.): Declarative Programming and Knowledge Management - Conference on Declarative Programming, DECLARE 2019, Unifying INAP, WLP, and WFLP, Cottbus, Germany, Revised Selected Papers. Vol. 12057. LNCS, Springer, pp. 106–115, 2019.

[Ma98] Marriott, K.: Programming with Constraints - An Introduction. MIT Press, Cambridge, 1998, ISBN: 978-0-262-13341-8.

[Pe01] Pesant, G.: A Filtering Algorithm for the Stretch Constraint. In (Walsh, T., ed.): Principles and Practice of Constraint Programming - CP. Vol. 2239. LNCS, Springer, pp. 183–195, 2001.

[Pe04] Pesant, G.: A Regular Language Membership Constraint for Finite Sequences of Variables. In (Wallace, M., ed.): Principles and Practice of Constraint Programming - CP. Vol. 3258. LNCS, Springer, pp. 482–495, 2004.

[Pe15a] Petke, J.: Bridging Constraint Satisfaction and Boolean Satisfiability. Springer, 2015, ISBN: 978-3-319-21809-0.

[Pe15b] Petke, J.: SAT encodings. In: Bridging Constraint Satisfaction and Boolean Satisfiability. Artificial Intelligence: Foundations, Theory, and Algorithms, Chapter 4, Springer, 2015, ISBN: 978-3-319-21809-0.

[Pe60] Peirce, C. S. (S.; Hartshorne, C.; Burks, A. W. (W.; Weiss, P.: Collected papers of Charles Sanders Peirce / edited by Charles Hartshorne and Paul Weiss. Belknap Press of Harvard University Press, Cambridge, 1960.

[Pr09] Prestwich, S. D.: CNF Encodings. In (Biere, A.; Heule, M.; van Maaren, H.; Walsh, T., eds.): Handbook of Satisfiability. Vol. 185, Frontiers in Artificial Intelligence and Applications, IOS Press, pp. 75–97, 2009.

[RBW06] Rossi, F.; Beek, P. v.; Walsh, T.: Handbook of Constraint Programming. Elsevier, Amsterdam, 2006, ISBN: 978-0-080-46380-3.

[RPD90] Rossi, F.; Petrie, C. J.; Dhar, V.: On the Equivalence of Constraint Satisfaction Problems. In: 9th European Conference on Artificial Intelligence, ECAI 1990, Stockholm, Sweden, 1990. Pp. 550–556, 1990.

[SF94] Sabin, D.; Freuder, E. C.: Contradicting Conventional Wisdom in Constraint Satisfaction. In: Eleventh European Conference on Artificial Intelligence, Amsterdam, The Netherlands. Pp. 125–129, 1994.

[Wa00] Walsh, T.: SAT v CSP. In: Principles and Practice of Constraint Programming - CP, 6th International Conference, Singapore, Proceedings. Pp. 441–456, 2000.