On Reformulation of Constraint Satisfaction Problems (Extended Version)

Rainer Weigel, Christian Bliek and Boi V. Faltings
Artificial Intelligence Laboratory
Swiss Federal Institute of Technology in Lausanne (EPFL)
IN-Ecublens, CH-1015 Lausanne, Switzerland

Abstract

The performance of constraint based problem solving depends crucially on the problem representation. The same problem may be easy or difficult to solve depending on the way we formulate it. In this paper, we study reformulations of discrete constraint satisfaction problems (CSPs). We present a technique to transform a CSP P into its Boolean form, which allows us to find different reformulations of P. We identify a new class of tractable CSPs and present sufficient conditions for detecting solvability and unsolvability of a CSP in linear time. Finally we introduce the notion of fault tolerance of solutions, and show how the Boolean form can be used to find them.

Keywords: Constraint-Based Reasoning

1 Introduction

Performance of problem solving depends crucially on the problem representation. The same problem may be easy or difficult to solve depending on the way we describe it. Recently, the problem of obtaining and comparing different problem encodings was raised as an AI challenge in [1]. In this paper, we describe a method to obtain different encodings of a given CSP and pose the following questions: 1.) is there an encoding whose unsolvability can be shown without search? 2.) can one encoding be solved more efficiently than another? and 3.) is it worth to combine encodings for improving search efficiency?

A discrete binary CSP P is defined by (X, D, C, R); where $X = \{X_1, \ldots, X_n\}$ is a set of variables, $D = \{D_1, \ldots, D_n\}$ a set of finite domains associated with the variables, $C = \{C_1, \ldots, C_l\}$ a set of constraints, and $R = \{R_{ij} \subset D_i \times D_j$ for a constraint applicable to X_i and $X_j\}$ a set of allowed tuples that define the constraints.

We first present a classic example for reformulation. In Section 2 we transform a CSP P into its Boolean form and use this form to find reformulations of P in Section 3. In Section 4 we present a new tractable class of CSPs and derive sufficient conditions for a CSP to be solvable or unsolvable. In Section 5 we introduce the notion of fault tolerant solutions, and show how the Boolean form can be used to find them.

Example: A classic "chestnut" for reformulation is the problem of tiling a mutilated checkerboard with dominoes with two opposite corner cut out as discussed for example in [15; 12; 10]. It is for example impossible to cover the board shown in Fig. 1 with dominoes. This is because a domino covers a square of each colour, but there are 30 black squares and 32 white squares. We can describe here two constraint models of the problem, and only by knowing that the models are equivalent we will prove unsolvability of the problem without search. We first formulate the mutilated checkerboard problem as a CSP where the variables in one formulation correspond to the white positions of the board. The domain for each variable is the set $\{$*above, below, left, right*$\}$. Assigning the value *above* to the variable on position $C2$ means that one covers the position $C2$ and $C3$ with a domino. Similarly, assigning *right* would mean that $C2$ and $D2$ are covered etc.. There are unary constraints restricting the values to cover always two positions of the board, i.e., a corner position will get for example only two values. There are also binary constraints between variables telling us that a position can be covered only once. One cannot assign for example the value *right* to the variable $C2$ and at the same time assigning *left* to the variable $E2$, because both assignment cover the position $D2$. One should note that an assignment simply means placing a domino. Thus, since the formulation includes 32 variables, we can place at most 32 dominoes. In a second formulation of the problem we take the black positions as variables (black formulation) with the same domains and constraints as above. To prove the impossibility of a tiling we argue as follows: Since 1.) every assignment in the black formulation corresponds to an assignment in the white formulation, and vice versa, and

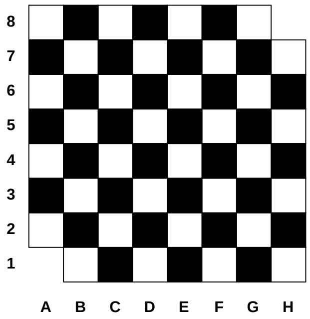


Figure 1: *A mutilated checkerboard with positions A1 and H8 removed.*

$$
\begin{array}{l}
R1.1 \\ R1.2 \\ R1.3 \\ R1.4 \\ R2.1 \\ R2.2 \\ R2.3 \\ R2.4 \\ R3.1 \\ R3.2 \\ R3.3 \\ R3.4 \\ R4.1 \\ R4.2 \\ R4.3 \\ R4.4
\end{array}
\begin{bmatrix}
\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&1&1&0&0&1&0&1&0&1&0&0&1\\
\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&1&1&1&0&0&1&0&1&0&1&0&0\\
\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&0&1&1&1&1&0&1&0&0&0&1&0\\
\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&0&0&1&1&0&1&0&1&1&0&0&1\\
1&1&0&0&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&1&1&0&0&1&0&1&0\\
1&1&1&0&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&1&1&1&0&0&1&0&1\\
0&1&1&1&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&0&1&1&1&1&0&1&0\\
0&0&1&1&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&0&0&1&1&0&1&0&1\\
1&0&1&0&1&1&0&0&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&1&1&0&0\\
0&1&0&1&1&1&1&0&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&1&1&1&0\\
1&0&1&0&0&1&1&1&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&0&1&1&1\\
0&1&0&1&0&0&1&1&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&0&0&1&1\\
1&0&0&1&1&0&1&0&1&1&0&0&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}\\
0&1&0&0&0&1&0&1&1&1&1&0&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}\\
0&0&1&0&1&0&1&0&0&1&1&1&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}\\
1&0&0&1&0&1&0&1&0&0&1&1&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}
\end{bmatrix}
$$

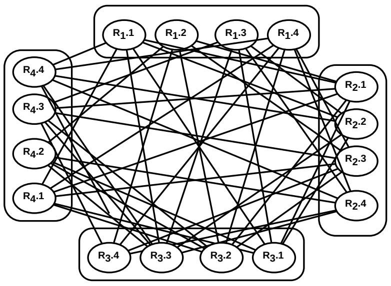


Figure 2: *(left:)* A_{bb} *of the 4-queens problem (right:) The constraint graph of the Boolean form*

2.) because we can make at most 30 assignments in the black formulation, we conclude that there are at most 30 assignments in the white formulation and thus the problem is not solvable.

2 The Boolean CSP

We now describe an algorithm to transform a discrete CSP P into a form that involves only Boolean (two-valued) variables. This form will be called the *Boolean form* of P. The idea behind the *Boolean form* is that an assignment of a value a to a variable X in an CSP can be interpreted as an assignment "1" to the Boolean variable $X.a$ in its Boolean form.

Algorithm 1 *Transformation algorithm*

Step 1: For each variable X_i and for each $d_j \in D_i$ create variables $X_i.d_j$ with domain $\{0,1\}$.

Step 2: Create for each X_i a m-ary constraint with connection $(X_i.d_1, \ldots, X_i.d_m)$ and with m allowed tu-ples $\{(1,0,0,\ldots 0),(0,1,0,\ldots 0),(0,0,1,\ldots 0),\ldots\}$. These constraints will be called *uniqueness* constraint, and their allowed tuples guarantee that one can assign only a single value to a variable in the original CSP.

Step 3: Create for every constraint C connecting the variables X_i and X_j and for every disallowed value tuple (d_i, d_j) in $D_i \times D_j$ a constraint in the Boolean CSP with connection $(X.d_i, Y.d_j)$ and with allowed tuples $\{(0,0),(1,0),(0,1)\}$.

The extension to convert non-binary constraints is straightforward. Like for binary constraints we create a new constraint for every disallowed *k-tuple*. Instead of representing compatibility as in the standard formulation one represents *incompatibility* in the Boolean CSP. We can see in step 3 that incompatibility means simply that two incompatible Boolean variable cannot have the value 1 at the same time, which is formulated simply by omitting the $(1,1)$ tuple.

Remark: The formulation of the Boolean CSP is similar to the formulation of a CSP as an integer linear programming problem as described in [17] and also close to a CSP encoding in an ATMS system [5].

Example: The 4-queens problem is that of finding possibilities to place 4 queens on a chessboard so that no two queens can attack each other. A representation ROW is a formulation with 4 variables each of which corresponds to a row of the chessboard. With every variable we associate 4 values that correspond to the 4 columns of the board. Assigning the value i to variable j means that we place a queen in the i-th column of the j-th row. The condition that no two queens attack each other can be described by 6 constraints. The constraint $C1$ with connection $(R1, R2)$ for example, allows the following definitions $\{(1,3),(1,4),(2,4),(3,1),(4,1),(4,2)\}$. Algorithm 1 transforms the row representation into the Boolean form. The variables are $R_i.1,..,R_i.4$ for $i = 1,\ldots,4$ each with domain $\{0,1\}$. In step 2 we have to introduce *4-ary* constraints with connection $\{R_i.1,..,R_i.4\}$ for $i = 1,\ldots,4$. In step 3 we iterate over all constraints in ROW. The constraint $C1$ with connection (R_1, R_2) disallows for example the tuple $(1,1)$. Thus we build a new constraint in the Boolean CSP with connection $(R_1.1, R_2.1)$ and with definitions $\{(0,0),(1,0),(0,1)\}$. The constraint graph is shown in Fig. 2 on the right.

2.1 The Binary Boolean Form

The arity of the Boolean CSP corresponds to the maximal arity of the uniqueness constraint, which is equal to the maximal domain size of the original problem. In order to obtain the *binary Boolean* formulation P_{bb} of P, we have to project the non-binary uniqueness constraints to sets of binary constraints. The constraint graph will be called $G_{bb}(P)$. The adjacency matrix A_{bb} of $G_{bb}(P)$ is a symmetric $nm \times nm$ matrix where the rows and columns corresponds to the variables in the Boolean CSP. An example of the adjacency matrix A_{bb} of the 4-queens problem is depicted in fig. 2 on the left. The projected uniqueness constraints form cliques in $G_{bb}(P)$

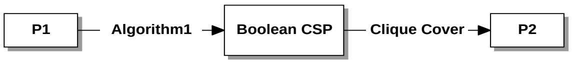


Figure 3: *Reformulating CSP1 into CSP2*

and can thus be represented as n blocks of size $m \times m$ on the diagonal of the matrix. An entry 1 outside the blocks represents a constraint of the Boolean form introduced in the third step of the algorithm. A solution to the CSP in this formulation can be described by n Boolean variables set to 1. Note that by the projection of a uniqueness constraint we lose the information that there must be exactly one Boolean variable from the connection of a uniqueness constraint, which must be "1" in a solution. Therefore we have to introduce a *counting constraint*, which guarantees that we have n Boolean variables set to 1 in a solution.

3 Reformulating CSPs

Generating Reformulations: Given a Boolean CSP with constraint graph G_{bb}. A *clique cover*[1] of G_{bb} induces a constraint problem P. Each clique of the clique cover corresponds to a variable in P. The domain size of a variable in P is always equal to the size of the corresponding clique plus 1, i.e., for every Boolean variable in a clique there exists a value and in addition there is a "nil" value, meaning that all Boolean variables in a clique can have a value 0. The problem P is in fact a *meta* CSP [6] obtained by clustering the nodes of each clique in the clique cover. Note that in general building a meta CSP is exponential in the size of the subproblems, because the number of solutions in a subproblem can be exponential. Here however it remains linear, because the clusters are cliques of Boolean variables which implies that there will always be $m + 1$ solutions. This limits of course the reformulations one can obtain using our approach, but at the same time distinguishes our approach from other types of reformulations. Reformulations based on interchangeability described in [7] for example cannot be obtained by finding different clique covers.

If two problems $P1$ and $P2$ are induced by different clique covers of G_{bb}, then we say that $P1$ is a *reformulation* of $P2$, and vice versa. Given a CSP $P1$, we obtain a reformulation $P2$ by first building the Boolean form of $P1$ and create then CSP $P2$ based on a different clique cover of G_{bb} as shown in Fig. 3. We consider now the case of a reformulation obtained by a clique cover of size n. Taking the counting constraint into account, we conclude that every clique must have one single Boolean variable set to 1 in order to find a solution. This implies that we can safely remove the "nil" value from the domain of the variables in such a reformulation. The removal of the "nil" values in these situations corresponds to the introduction of a *redundant constraint*[2] in the Boolean form. This constraint guarantees that exactly one variable in each of the cliques will have the value 1 in any solution of the problem. Note that we do not get these redundant constraints for free, because finding these clique covers is in general intractable. Suppose that the Boolean form of the 9-queens is given and we know that clustering the variables into rows corresponds to a clique cover of size 9. The current instantiation indicated by the black board positions in Fig. 4 forces the variable $R7.9$ to be 1, because $R_7.1, R_7.2, \ldots R_7.8$ have necessarily the value 0. Without the redundant constraint, it would be also possible to instantiate either one of the variables $R_6.8, R_2.4, R_4.9, R_2.9$, and $R_8.9$ with the value 1. These instantiations however would all lead to a dead end because they would all be in conflict with $R7.9$ being 1.

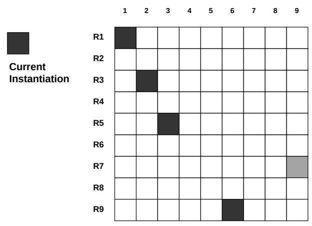


Figure 4: *Row Representation of the 9-queens*

Consider now a reformulation with a clique cover of size m bigger than n. We cannot induce redundant constraints in this reformulation in the same way we have done above. The counting constraint allows additional pruning only during search when certain variables are instantiated. Thus it is difficult to draw conclusion about the utility of the counting constraint. On the other hand one can also see that the pruning can be obtained "earlier" for a reformulation corresponding to a clique cover of size m very close to n. An example where the clique cover size is large compared to n, and where we could not find a situation during search were the counting constraint allows additional pruning is the *diagonal* representation of the *n-queens* problem [14]. This formulation is obtained by clustering the "main diagonal" variables $R_i.i$ for $i = 1 \ldots n$ and those variables which correspond to the upper and lower down sloping diagonals. This formulation has $2n - 1$ variables, but only n Boolean variables set to 1 are needed in a solution. It follows that there are always $n - 1$ diagonals with the "nil" value in

[1] A *clique cover* of size k is a partition of the vertices $V = A_1 + A_2 + \ldots A_k$ such that each A_i is a clique. $k(G)$ is the size of a smallest possible clique cover of G; it is called the *clique cover number*.

[2] A constraint is redundant if it can be deduced from the other constraints in the problem.

a solution of the problem.

We do not try to find a ranking of different reformulations of a problem with respect to problem solving efficiency. Indeed, we know that if we add redundant constraints, the number of possible partial solutions will be smaller. However, we do not know if we can take advantage of the redundant constraint during search. The reason are the following:

- Variable and value ordering heuristics might not lead us to the situation where the redundant constraint allows additional pruning.
- If we are interested in only a single solution, then it is possible that a redundant can make search even less efficient. The reason for this is that additional pruning can misguide the variable and value ordering heuristics to regions in the search space without solutions. Thus exploring regions which might otherwise be not explored.

A ranking of representation would for sure require a deep understanding of the relationship between the representation and the shape of the search space. Nevertheless we expect that, in a problem with n variables, the search for all solutions in formulations with a clique cover of size n will be more efficient on average than search in representations corresponding to clique cover of size bigger than n.

Combining Reformulations: Even if we do not know which is the "best" representation, we can ask whether combining representations can be of interest. One can see, that combining two representation is useful for additional pruning, if one representation can contribute *redundant constraints*, which are not available in the other. Combining representations means simply taking the Boolean form and add the redundant constraints from other representations. In the row representation of the n-queens we do not know that there must be exactly one queen per column. However we can get this additional knowledge by combining row and column formulation. This formulation includes the Boolean form of the row representation from Section 2 and in addition the uniqueness constraints from the column representation. That is, the uniqueness constraints with connections $\{R_1.i, .., R_4.i\}$ for $i = 1, \ldots, 4$.

Cheng et al. proposed in [2] *redundant modelling* as an approach for increasing the pruning in the constraint solving process. This approach is based on the idea that different models of the same problem can be combined with channelling constraints, and that the combined model allows more pruning than a single one. Consider for example the row representation and the column representation, which can be combined with channelling constraints of the following form: "If row x has the value y, the column y has the value x, and vice versa". Case studies based on the n-queens problem and on a real-life nurse rostering problem have shown, that search efficiency could indeed be improved. Combining different reformulations using the Boolean form is conceptually similar to redundant modelling. This is because the same "information content" is represented in both approaches. We know for example in the combined n-queens problems, that there is only one queen per row and per column and that no two queens attack each other.

Thus redundant modelling with channelling constraints and combing formulations via the Boolean form are equivalent with respect to pruning because exactly the same constraints are available in both formulations. However, with the Boolean form, we can explain that the additional pruning reported in [2] is due to redundant uniqueness constraints coming from different reformulations. Thus one can argue that redundant modelling corresponds to adding redundant constraints in the Boolean formulation. Note however that the Boolean form must be represented only once, compared to redundant modelling, where one has to represent every model and furthermore add all the channelling constraints.

4 Clique Covers and Stable Sets of G_{bb}

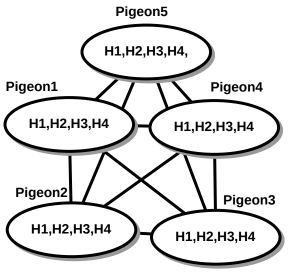
We first note that the constraint graph $G_{bb}(P)$ of a CSP P is the *complement* graph of the microstructure $\mu(P)$ of P, i.e., $G_{bb}(P) = \overline{\mu(P)}$. We know from [9] that for a solvable problem P the maximum clique size $\omega(\mu)$ is always equal to the number of variables in P. Let $\alpha(G_{bb})$ denote the *stability number*[3] and $k(G_{bb})$ the *clique cover number* of G_{bb}. Since the intersection of a clique and a stable set can be at most one node, we know that $\alpha(G_{bb}) \leq k(G_{bb})$. Thus with $G_{bb}(P) = \overline{\mu(P)}$, we conclude that $\omega(\mu) = \alpha(G_{bb}) \leq k(G_{bb})$. This leads us to the following theorem, which gives us a sufficient condition for P to be unsolvable:

Theorem 1 *Given a binary CSP P with n variables. If the* clique cover number $k(G_{bb})$ *of the constraint graph of the binary Boolean CSP is smaller than n then P is not solvable.*

Proof: $\omega(\mu(P)) = \alpha(G_{bb}) \leq k(G_{bb}) < n$. Since $\omega(\mu(P)) < n$, we can conclude that P is not solvable. $\square$

Proving the unsolvability of a problem P often requires searching a space exponential in the size of the problem. However reformulations of P may be proven unsolvable without search.

[3]A *stable set* is a subset X of vertices and no two of which are adjacent. It is also called a *independent set*. $\alpha(G)$ is the number of vertices in a stable set of maximum cardinality; it is called the *stability number* of G.



$$
\begin{array}{l}
\mathrm{P1.H1}\\ \mathrm{P2.H1}\\ \mathrm{P3.H1}\\ \mathrm{P4.H1}\\ \mathrm{P5.H1}\\ \mathrm{P1.H2}\\ \mathrm{P2.H2}\\ \mathrm{P3.H2}\\ \mathrm{P4.H2}\\ \mathrm{P5.H2}\\ \mathrm{P1.H3}\\ \mathrm{P2.H3}\\ \mathrm{P3.H3}\\ \mathrm{P4.H3}\\ \mathrm{P5.H3}\\ \mathrm{P1.H4}\\ \mathrm{P2.H4}\\ \mathrm{P3.H4}\\ \mathrm{P4.H4}\\ \mathrm{P5.H4}
\end{array}
\left[\begin{array}{cccccccccccccccccccc}
\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0\\
\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0\\
\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0\\
\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0\\
\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1\\
1&0&0&0&0&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&1&0&0&0&0&1&0&0&0&0\\
0&1&0&0&0&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&0&1&0&0&0&0&1&0&0&0\\
0&0&1&0&0&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&0&0&1&0&0&0&0&1&0&0\\
0&0&0&1&0&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&0&0&0&1&0&0&0&0&1&0\\
0&0&0&0&1&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&0&0&0&0&1&0&0&0&0&1\\
1&0&0&0&0&1&0&0&0&0&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&1&0&0&0&0\\
0&1&0&0&0&0&1&0&0&0&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&0&1&0&0&0\\
0&0&1&0&0&0&0&1&0&0&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&0&0&1&0&0\\
0&0&0&1&0&0&0&0&1&0&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&0&0&0&1&0\\
0&0&0&0&1&0&0&0&0&1&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&0&0&0&1&0\\
1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}\\
0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}\\
0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}\\
0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}\\
0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}
\end{array}\right]
$$

Figure 5: *Pigeon Hole CSP.*

Examples: The pigeon hole problem is such a well-known insoluble example. The problem is to put n pigeons into $n-1$ holes. However, every hole admits only a single pigeon. The problem can be formulated as a CSP with n variables corresponding to the n pigeons, and every variable has $n-1$ values corresponding to the holes. The 5 pigeon problem is shown in Fig. 5. On the right we can see the matrix representation of a reformulation of the 5 pigeon problem. In this reformulation there are only 4 blocks of 1 on the diagonal, i.e., there is a clique cover of size 4. Thus we conclude by Theorem 4 that the problem is not solvable.

Another example is the problem of tiling a mutilated checkerboard with dominoes described above. Here the white formulation corresponds to a clique cover of size 30 and thus the problem is not solvable.

The *super-n-queens* problem is to place n queens on an $n \times n$ chessboard so that no two queens attack each other even if we allow the queens to make knight moves. One can easily see that the super-3-queens problem one can place at most a single queen. This means that 9 Boolean variables corresponding to the positions of a 3×3 board form a clique. Unsolvability of the super-n-queens can be proven without search for $n = 6$ and $n = 7$. For the case $n = 6$ we can find a clique cover of size four, i.e., the four blocks of size 3×3. For the case $n = 7$ we can use the four blocks of size 3×3 and add furthermore one clique for each row respectively column which remains. Thus we end up with a clique cover of size 6 for a problem where 7 queens must be placed.

Approximations: Unfortunately, the usefulness of the theorem is limited in practice, since calculating $k(G)$ is known to be NP-complete for an arbitrary graph [13]. In the next theorem we describe a class of CSPs where $k(G_{bb})$ and $\alpha(G_{bb})$ can be calculated in linear time.

Theorem 2 *Let P be a binary CSP with n variables. If the constraint graph $G_{bb}(P)$ is triangulated, then one can find a solution of P in linear time.*

Proof: An edge in G_{bb} represents incompatibility. A stable set of size n contains n pairwise not incompatible Boolean variables. We conclude that a stable set of size n of G_{bb} is a solution of P. Since G_{bb} is triangulated, we can find a maximum stable set of $G_{bb}(P)$ in linear time by Gavril's theorem [8]. □

With the theorem above we have identified a new tractable class of CSPs. However, constraint problems with a triangulated constraint graph G_{bb} are rare. We can use Theorem 2 to derive sufficient conditions for a CSP P to be solvable. The idea is to triangulate the constraint graph $G_{bb}(P)$ in linear time [8]. An edge added by the triangulation means that the pair of Boolean variables connected by the edge is made incompatible. Thus the problem P' obtained by triangulating $G_{bb}(P)$ contains more incompatibility, and is thus more constrained than P. Since the constraint graph of $G_{bb}(P')$ is triangulated, we use Theorem 2 to state the following corollary:

Corollary 1 *Let $G_{bb}(P')$ be obtained by triangulation of the $G_{bb}(P)$. If P' is solvable, then P is solvable, and we can find a solution of P in linear time.*

A sufficient condition for P to be unsolvable can be derived as follows. Instead of adding edges in order to obtain a triangulated graph $G_{bb}(P')$, one can also remove edges from the graph to find a maximal triangulated subgraph $G_{bb}(P)$. This can be done with the algorithm MaxChord [16], which can be implemented in $O(|E|\Delta)$ where Δ is the maximum degree of any vertex in G. Again, since $G_{bb}(P')$ is now triangulated, we can enumerate a solution P' in linear time. We can state the following corollary:

Corollary 2 *Let $G_{bb}(P')$ be a triangulated subgraph $G_{bb}(P)$. If P' is unsolvable, then P is unsolvable, and we decide the unsolvability of P in linear time.*

The idea of relaxing constraints can be applied to resource allocation problems:

Resource Allocation (RA) Problems: Resource Allocation (RA) problems can be defined as follows (see [3]): *"Given a set of tasks with fixed endpoints, and given, for each task a set of resources that can carry out the task, assign one resource to each of the tasks such that no resource is assigned to two different tasks at the same time".*

The *RA* problem can be modelled as a discrete binary CSP where the variables are the tasks, the values are the resources and there are constraints of mutual exclusion between two variables if the corresponding tasks intersect in time. Consider the interval representation of the small example in Fig. 6 where we have 7 tasks $T1, \ldots, T7$ and resources $R1, \ldots, R4$. Two task intersecting in time are not allowed to have the same resource. If we assign for example the resource $R1$ to $T1$, then $R2$ must be

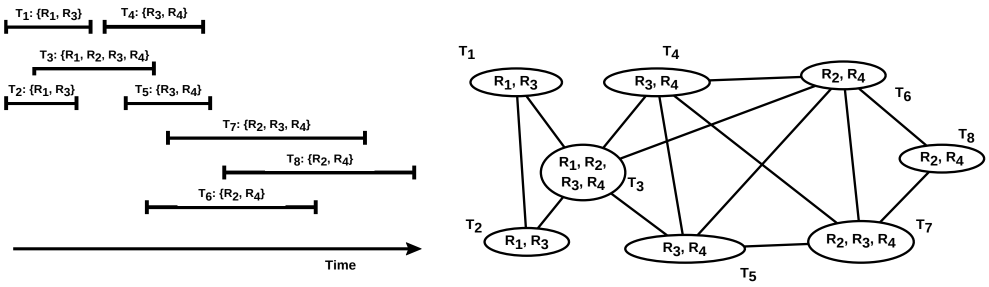


Figure 6: *(left): Interval orders: a schedule of seven tasks whose start time and duration are fixed. For each task, a set of possible resources is shown. (right): the corresponding constraint graph.*

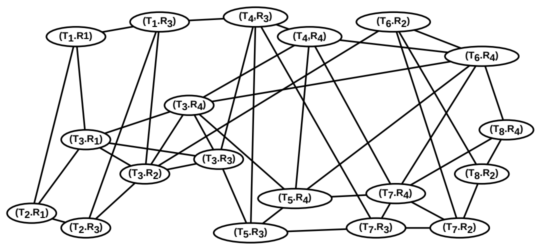


Figure 7: G_{bb} *of the RA problem*

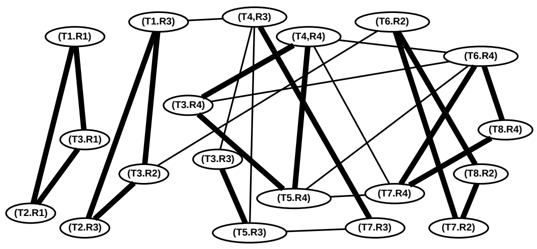


Figure 8: *A clique cover of size 7 of the RA example in Fig. 6*

assigned to $T2$ etc. One can easily see that this small example is unsolvable. The constraint graph of the RA problem is shown on the right of Fig. 6. Nodes in the constraint graph represent tasks to be executed, and their labels are the set of resources that can carry out the task. The constraint graph of the Boolean formulation $G_{bb}(RA)$ is shown in Fig. 7.

To apply Corollary 2 to the RA problem in Fig. 6 we have to find a triangulated subgraph of $G_{bb}(RA)$. To do so we remove the edges between nodes with the same task. The result is a set of disconnected graphs, each containing the nodes for a given resource. Each subgraph is an interval graph because of the underlying interval graph of the problem RA[4]. Thus one can see that the resulting graph is always an interval graph. Note that this was already described by B.Y. Choueiry in [4]. She called the resulting graph the *co-microstructure* and derived a disjunctive decomposition scheme based on the maximal cliques of the interval graph.

Since interval graphs are triangulated and by the corollary above, and by knowing that removing these edges from $G_{bb}(RA)$ results in a less constrained problem, we derive a sufficient condition for a RA problem to be unsolvable as follows:

Corollary 3 *Given a resource allocation problem RA with $n = |T|$ tasks. We identify a clique cover of $G_{bb}(RA)$ in linear time, and if the size of the clique cover is less than n then the problem RA is unsolvable.*

A clique cover of the problem of size 7 is shown in Fig. 8. The cliques are indicated with bold edges[5]. There are of course limitations with this approach. Consider the case where a subproblem induced by some tasks is over-constrained and another subproblem is under-constrained. The under-constrained subproblem will contribute more cliques than there are tasks in the subproblem, and the over-constrained subproblem contributes less cliques. Thus the overall clique cover size might not allow us to conclude unsolvability of the problem. Such a situation is shown in Fig. 9. This example contains the same tasks as the example in Fig. 6, however there are more resources available for the tasks 7 and task 8. The under-constraint region with task 7 and task 8 will lead to two more cliques in a minimal clique cover. Thus we end up with a clique cover of size 9 for a problem with 8 tasks, and cannot conclude unsolvability with the corollary. Nevertheless, one can use the sufficient condition for proving unsolvability during search. For doing so, one needs to calculate a minimum clique

[4]Node deletion in interval graphs is hereditary on induced subgraphs, i.e., an interval graph remains an interval graph after removing nodes

[5]We should mention the difference between a frequently used test for unsolvability of RA problems. This test calculates the maximum cliques of the interval graph in linear time and verifies if the number of resources available of the subproblems induced by each maximal clique is larger than its clique size. Note that this test would fail to show the unsolvability of the example.

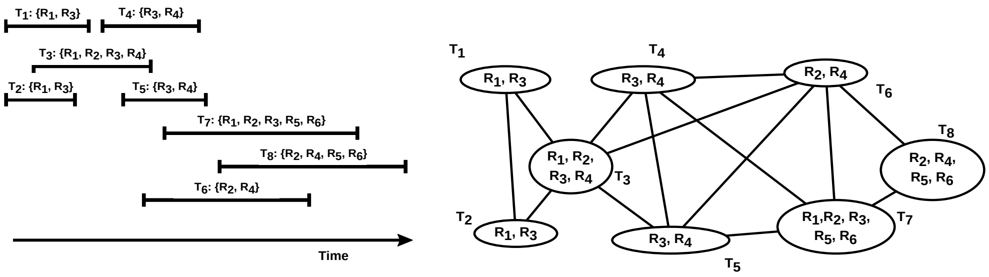
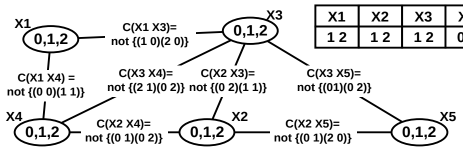


Figure 9: *Another unsolvable RA example*



X1	X2	X3	X4	X5
1 2	1 2	1 2	0 2	1 2

Figure 10: Small example with the non-allowed tuples for the constraints

cover in a preprocessing step. Assigning resources to tasks leads to the removal of some nodes in the graph $G_{bb}(RA)$ and thus decreases the clique cover size calculated in the first step. Consider again Fig. 9 where assigning a resource $R4$ to task 6 leads to the removal of 3 cliques from the clique cover. Thus we can conclude unsolvability after this assignment. Backtracking will occur and a test for the other values of task 6 will be done. After these test one can finally conclude the unsolvability of the problem.

Discussion: We have not yet evaluated the approximations proposed in Corollary 1 and 2 on random problems in order to see how incomplete they are. We do believe that especially for hard random problems these methods are unlikely to be useful. Nevertheless as we have seen in the resources allocation example, one can integrate such a test for unsolvability into a search procedure, which might be able to detect deadends during search early.

5 Fault Tolerant Solutions

We define a *fault tolerant solution* as follows:

Definition 1 A solution S for a CSP P is said to be fault tolerant, iff for a value a for any variable X in S, there exists another value b in the domain of X, such that S remains a solution after a is replaced by b.

The "fault tolerance" of a solution can be interesting in a number of applications. This is the case in a production environment when a machine breaks down, or in personal allocation problem when an employee is temporary not available. Whenever a value of the current solution is invalidated, then we should be able to immediately repair the solution. If our current solution is fault tolerant, then we know that for every value a exists another value b which serves equally well in the solution and can be used to fix the solution. Enumerating fault tolerant solutions in the standard formulation of a CSP is difficult, since the fact that a variable can take only a single value is *implicitly* coded in the problem. The uniqueness constraint in the Boolean CSP of a problem, however, make this fact explicit. Thus one can easily modify these constraints.

To obtain a formulation which allows us to find fault tolerant solutions we modify the Boolean form as follows: We replace the allowed tuples of the uniqueness constraints by those tuples which allow everything from the crossproduct of the domains except the zero tuple and the tuples that contain only a single 1. Note that the resulting CSP allows only fault tolerant solutions, other solutions are no longer valid. A solution of the modified Boolean CSP will necessarily have at least two Boolean variables set to the value 1 for every uniqueness constraint. Thus a solution can always be presented in the form of a cross product. Consider the small CSP in Fig. 10 with 5 variables all having the domain $(0, 1, 2)$. The constraints allow all tuples in $D \times D$ except the listed ones. Exhaustive enumeration in the standard formulation returns 64 solutions. The modified version of the Boolean form allows only a single solution containing 10 Boolean variables set to 1. This single solution correspond to 32 solutions of the original CSP. These fault tolerant solutions are presented in the cross product representation in Fig. 10.

Another way to find fault tolerant solutions would be to generate a CSP with $2n$ variables, where every variable in the original CSP appears twice. Furthermore one adds a disequality constraint between each variable and its duplicate, i.e., $X_j = d_i \Rightarrow X'_j \neq d_i$ and add the constraint between X_i and X'_j if there was a constraint between X_i and X_j for example. The disadvantage of this approach is that one needs to represent the whole CSP twice. Furthermore it wouldn't be possible to find solution sets with more that two values could be assigned to a single variable.

Completely dropping the uniqueness constraints of the Boolean form is also interesting. It means that we do not impose constraints on the number of values that can be 1 in a solution. The solution of such a CSP contains all partial solutions, all complete solutions, and all solution sets with multiple values for a single variable. With such a form one can for example enumerate the *maximal cross product representation* of solution sets, which is closely related to the maximal consistent decision described in [11].

6 Conclusion

In this paper we presented an algorithm to transform a CSP P into its Boolean form. This Boolean form was then used to find reformulations of P. Reformulations, as defined here, differ with each other only in redundant constraints. This shows that one formulation can allow pruning in some situations, which is not possible in other formulations. Discovering certain properties of the constraint graph G_{bb} of the Boolean formulation led us to a new tractable class of CSPs as well as to sufficient conditions for deciding solvability (respectively unsolvability) by adding (respectively removing) edges in the constraint graph G_{bb}. Finally we introduced the notion of fault tolerant solutions, which can be obtained using the Boolean form.

References

[1] B. Selman, H. Kautz and D. McAllester, 'Ten Challenges in Propositional Reasoning and Search', in *Proc. of the 16 th IJCAI*, pp. 50–54, Nagoya, Japan, (1997).

[2] B.M.W. Cheng, J.H.M. Lee, and J.C.K Wu, 'Speeding up constraint propagation by redundant modeling', in *Second International Conference on Principles and Practice of Constraint Programming*, pp. 91–103, (1996).

[3] Berthe Y. Choueiry, *Abstraction Methods for Resource Allocation*, Ph.D. dissertation, Swiss Federal Institute of Technology in Lausanne, 1994.

[4] Berthe Y. Choueiry and Guevara Noubir, 'Using complete no-good sets to build a disjunctive decomposition scheme for discrete constraint satisfaction problems', in *FLAIRS'97*, pp. 40–46, Daytona Beach, FL, (1997).

[5] J. de Kleer, 'A Comparison of Assumption-Based Truth Maintenance and Constraint Satisfaction', in *Proc. of the 11 th IJCAI*, pp. 290–296, Detroit, MI, (1989).

[6] Eugene C. Freuder, 'Constraint solving techniques', in *Constraint Programming of series F: Computer and Sytems Sciences*, eds., E. Tyngu B. Mayoh and J. Penjaen, 51–74, NATO ASI Series, (1992).

[7] Eugene C. Freuder and Daniel Sabin, 'Interchangeability Supports Abstraction and Reformulation for Multi-Dimensional Constraint Satisfaction', in *Proc. of AAAI-96*, pp. 191–196, Portland,, (1996).

[8] F. Gavril, 'Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph', *SIAM J. Comput.*, **1-2**, (1972).

[9] Philippe Jégou, 'Decomposition of domains based on the micro-structure of Finite Constraint-Satisfaction Problems', in *Proc. of AAAI-93*, pp. 731–736, Washington, DC, (1993).

[10] Richard E. Korf, 'Toward a model of representation changes', *Artificial Intelligence*, **14**, 41–78, (1980).

[11] David Lesaint, 'Maximal Sets of Solutions for Constraint Satisfaction Problems', in *Proc. of the 11 th ECAI*, pp. 110–114, Amesterdam, The Netherlands, (1994).

[12] J. McCarthy. A tough nut for proof procedures. AI Project Memo 16, Stanford University, July 1964. available at http://www-formal.stanford.edu/jmc/nut.html.

[13] D.S. Johnson M.R. Garey, *Computers and Intractability: A Guide to the Theory of NP-completeness.*, Freeman, San Francisco, CA, 1978.

[14] B. Nadel, 'Representation selection for constraint satisfaction: A case study using n-queens', *IEEE Expert*, **5(3)**, 16–23, (1990).

[15] Allen Newell, 'Limitations of the current stock of ideas about problem solving.', in *Electronic handling of information*, ed., Allen Kent a.o.., Spartan books, 1965, Washington, DC, (1965).

[16] P.M. Dearing, D.R. Shier, D.D. Warner, 'Maximal chordal sugraphs', *Discrete Applied Mathematics.*, **20**, 181–190, (1988).

[17] Igor Rivin and Ramin Zabih, 'An Algebraic Approach to Constraint Satisfaction Problems', in *Proc. of the 11 th IJCAI*, pp. 285–289, Detroit, MI, (1989).