Broken Triangles Revisited

Martin C. Cooper[1](✉), Aymeric Duchein[1], and Guillaume Escamocher[2]

[1] IRIT, University of Toulouse III, 31062 Toulouse, France
{cooper,Aymeric.Duchein}@irit.fr
[2] INSIGHT Centre for Data Analytics, University College Cork, Cork, Ireland
guillaume.escamocher@insight-centre.org

Abstract. A broken triangle is a pattern of (in)compatibilities between assignments in a binary CSP (constraint satisfaction problem). In the absence of certain broken triangles, satisfiability-preserving domain reductions are possible via merging of domain values. We investigate the possibility of maximising the number of domain reduction operations by the choice of the order in which they are applied, as well as their interaction with arc consistency operations. It turns out that it is NP-hard to choose the best order.

1 Introduction

The notion of broken triangle has generated a certain amount of interest in the constraints community: it has led to the definition of novel tractable classes [3,7], variable elimination rules [1] and domain reduction rules [4,5]. The merging of pairs of values in the same variable domain which do not belong to a broken triangle has been shown to lead to considerable reduction of search space size for certain benchmark instances of binary CSP [4]. The corresponding reduction operation, known as BTP-merging, is satisfiability-preserving and is therefore worthy of a deeper theoretical analysis as a potentially useful preprocessing operation. An obvious question is whether the order in which BTP-merging operations, and other domain-reduction operations such as arc consistency, are performed has an effect on the number of possible merges.

Definition 1. *A* binary CSP instance *I consists of*

- *a set X of n variables,*
- *a domain $\mathcal{D}(x)$ of possible values for each variable $x \in X$, with d the maximum domain size,*
- *a relation $R_{xy} \subseteq \mathcal{D}(x) \times \mathcal{D}(y)$, for each pair of distinct variables $x, y \in X$, which consists of the set of compatible pairs of values (a, b) for variables (x, y).*

A partial solution *to I on $Y = \{y_1, \ldots, y_r\} \subseteq X$ is a set $\{\langle y_1, a_1\rangle, \ldots, \langle y_r, a_r\rangle\}$ such that $\forall i, j \in [1, r]$, $(a_i, a_j) \in R_{y_i y_j}$. A* solution *to I is a partial solution on X.*

M.C. Cooper—supported by ANR Project ANR-10-BLAN-0210 and EPSRC grant EP/L021226/1.

For simplicity of presentation, Definition 1 assumes that there is exactly one constraint relation for each pair of variables. An instance I is *arc consistent* if for each pair of distinct variables $x, y \in X$, for each value $a \in \mathcal{D}(x)$, there is a value $b \in \mathcal{D}(y)$ such that $(a, b) \in R_{xy}$.

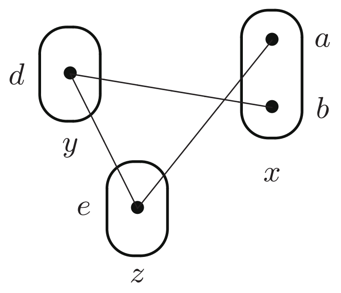

Fig. 1. A broken triangle on two values $a, b \in \mathcal{D}(x)$.

We now formally define the value-merging operation based on absence of certain broken triangles. A broken triangle on values a, b is shown in Figure 1. In all figures in this paper, the pairs of values joined by a solid line are exactly those belonging to the corresponding constraint relation.

Definition 2. *A* broken triangle *on the pair of variable-value assignments $a, b \in \mathcal{D}(x)$ consists of a pair of assignments $d \in \mathcal{D}(y)$, $e \in \mathcal{D}(z)$ to distinct variables $y, z \in X \setminus \{x\}$ such that $(a, d) \notin R_{xy}$, $(b, d) \in R_{xy}$, $(a, e) \in R_{xz}$, $(b, e) \notin R_{xz}$ and $(d, e) \in R_{yz}$. The pair of values $a, b \in \mathcal{D}(x)$ is* BT-free *if there is no broken triangle on a, b.*

BTP-merging *values $a, b \in \mathcal{D}(x)$ in a binary CSP consists in replacing a, b in $\mathcal{D}(x)$ by a new value c which is compatible with all variable-value assignments compatible with at least one of the assignments $\langle x, a \rangle$ or $\langle x, b \rangle$ (i.e. $\forall y \in X \setminus \{x\}$, $\forall d \in \mathcal{D}(y)$, $(c, d) \in R_{xy}$ iff $(a, d) \in R_{xy}$ or $(b, d) \in R_{xy}$)*

When $a, b \in \mathcal{D}(x)$ are BT-free in a binary CSP instance I, the instance I' obtained from I by merging $a, b \in \mathcal{D}(x)$ is satisfiable if and only if I is satisfiable. Furthermore, given a solution to the instance resulting from the merging of two values, we can find a solution to the original instance in linear time [4].

The paper is structured as follows. In Section 2 we investigate the interaction between arc consistency and BTP-merging. In Section 3 we show that finding the best order in which to apply BTP-mergings is NP-hard, even for arc-consistent instances. In Section 4 we prove that this remains true even if we only perform merges at a single variable. In Section 5 we take this line of work one step further by showing that it is also NP-hard to find the best sequence of merges by a weaker property combing virtual interchangeability and neighbourhood substitutability.

2 Mixing Arc Consistency and BTP-merging

BTP-merging can be seen as a generalisation of neighbourhood substitutability [6], since if $a \in \mathcal{D}(x)$ is neighbourhood substitutable for $b \in \mathcal{D}(x)$ then a, b

can be BTP-merged. The possible interactions between arc consistency (AC) and neighbourhood substitution (NS) are relatively simple and can be summarised as follows:

1. The fact that $a \in \mathcal{D}(x)$ is AC-supported or not at variable y remains invariant after the elimination of any other value b (in $\mathcal{D}(x) \setminus \{a\}$ or in the domain $\mathcal{D}(z)$ of any variable $z \neq x$) by neighbourhood substitution.
2. An arc-consistent value $a \in \mathcal{D}(x)$ that is neighbourhood substitutable remains neighbourhood substitutable after the elimination of any other value by arc consistency.
3. On the other hand, a value $a \in \mathcal{D}(x)$ may become neighbourhood substitutable after the elimination of a value $c \in \mathcal{D}(y)$ $(y \neq x)$ by arc consistency.

Indeed, it has been shown that the maximum cumulated number of eliminations by arc consistency and neighbourhood substitution can be achieved by first establishing arc consistency and then applying any convergent sequence of NS eliminations (i.e. any valid sequence of eliminations by neighbourhood substitution until no more NS eliminations are possible) [2].

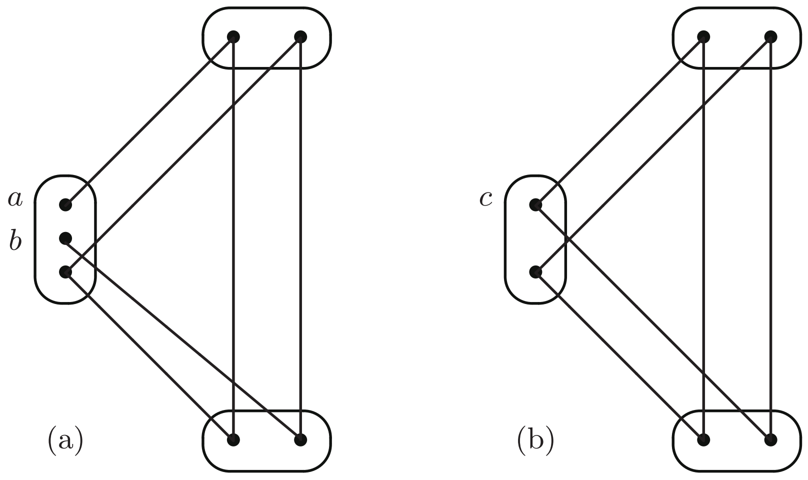

Fig. 2. (a) An instance in which applying AC leads to the elimination of all values (starting with the values a and b), but applying BTP merging leads to just one elimination, namely the merging of a with b (with the resulting instance shown in (b)).

The interaction between arc consistency and BTP-merging is not so simple and can be summarised as follows:

1. The fact that $a \in \mathcal{D}(x)$ is AC-supported or not at variable y remains invariant after the BTP-merging of any other pair of other values b, c (in $\mathcal{D}(x) \setminus \{a\}$ or in the domain $\mathcal{D}(z)$ of any variable $z \neq x$). However, after the BTP-merging of two arc-inconsistent values the resulting merged value may be arc consistent. An example is given in Figure 2(a). In this 3-variable instance, the two values $a, b \in \mathcal{D}(x)$ can be eliminated by arc consistency (which in turn leads to

the elimination of all values), or alternatively they can be BTP-merged (to produce the value c) resulting in the instance shown in Figure 2(b) in which no more eliminations are possible by AC or BTP-merging.
2. A single elimination by AC may prevent a sequence of several BTP-mergings. An example is given in Figure 3(a). In this 4-variable instance, if the value b is eliminated by AC, then no other eliminations are possible by AC or BTP-merging in the resulting instance (shown in Figure 3(b)), whereas if a and b are BTP-merged into a new value d (as shown in Figure 3(c)) this destroys a broken triangle thus allowing c to be BTP-merged with d (as shown in Figure 3(d)).
3. On the other hand, two values in the domain of a variable x may become BTP-mergeable after an elimination of a value $c \in \mathcal{D}(y)$ $(y \neq x)$ by arc consistency.

3 The Order of BTP-mergings

It is known that BTP-merging can both create and destroy broken triangles [4]. This implies that the choice of the order in which BTP-mergings are applied may affect the total number of merges that can be performed. Unfortunately, maximising the total number of merges in a binary CSP instance turns out to be NP-hard, even when bounding the maximum size of the domains d by a constant as small as 3. For simplicity of presentation, we first prove this for the case in which the instance is not necessarily arc consistent. We will then prove a tighter version, namely NP-hardness of maximising the total number of merges even in arc-consistent instances.

Theorem 1. *The problem of determining if it is possible to perform k BTP-mergings in a boolean binary CSP instance is NP-complete.*

Proof. For a given sequence of k BTP-mergings, verifying if this sequence is correct can be performed in $\mathcal{O}(kn^2d^2)$ time because looking for broken triangles for a given couple of values takes $\mathcal{O}(n^2d^2)$ [4]. As we can verify a solution in polynomial time, the problem of determining if it is possible to perform k BTP-mergings in a binary CSP instance is in NP. So to complete the proof of NP-completeness it suffices to give a polynomial-time reduction from the well-known 3-SAT problem. Let I_{3SAT} be an instance of 3-SAT (SAT in which each clause contains exactly 3 literals) with variables $X_1, \ldots, X_N$ and clauses $C_1, \ldots, C_M$. We will create a boolean binary CSP instance I_{CSP} which has a sequence of $k = 3 \times M$ mergings if and only if I_{3SAT} is satisfiable.

For each variable X_i of I_{3SAT}, we add a new variable z_i to I_{CSP}. For each occurrence of X_i in the clause C_j of I_{3SAT}, we add two more variables x_{ij} and y_{ij} to I_{CSP}. Each $\mathcal{D}(z_i)$ contains only one value c_i and each $\mathcal{D}(x_{ij})$ (resp. $\mathcal{D}(y_{ij})$) contains only two values a_i and b_i (resp. a'_i and b'_i). The roles of variables x_{ij} and y_{ij} are the following:

$$X_i = true \Leftrightarrow \forall j,\ a_i, b_i \text{ can be merged in } \mathcal{D}(x_{ij}) \tag{1}$$

$$X_i = false \Leftrightarrow \forall j,\ a'_i, b'_i \text{ can be merged in } \mathcal{D}(y_{ij}) \tag{2}$$

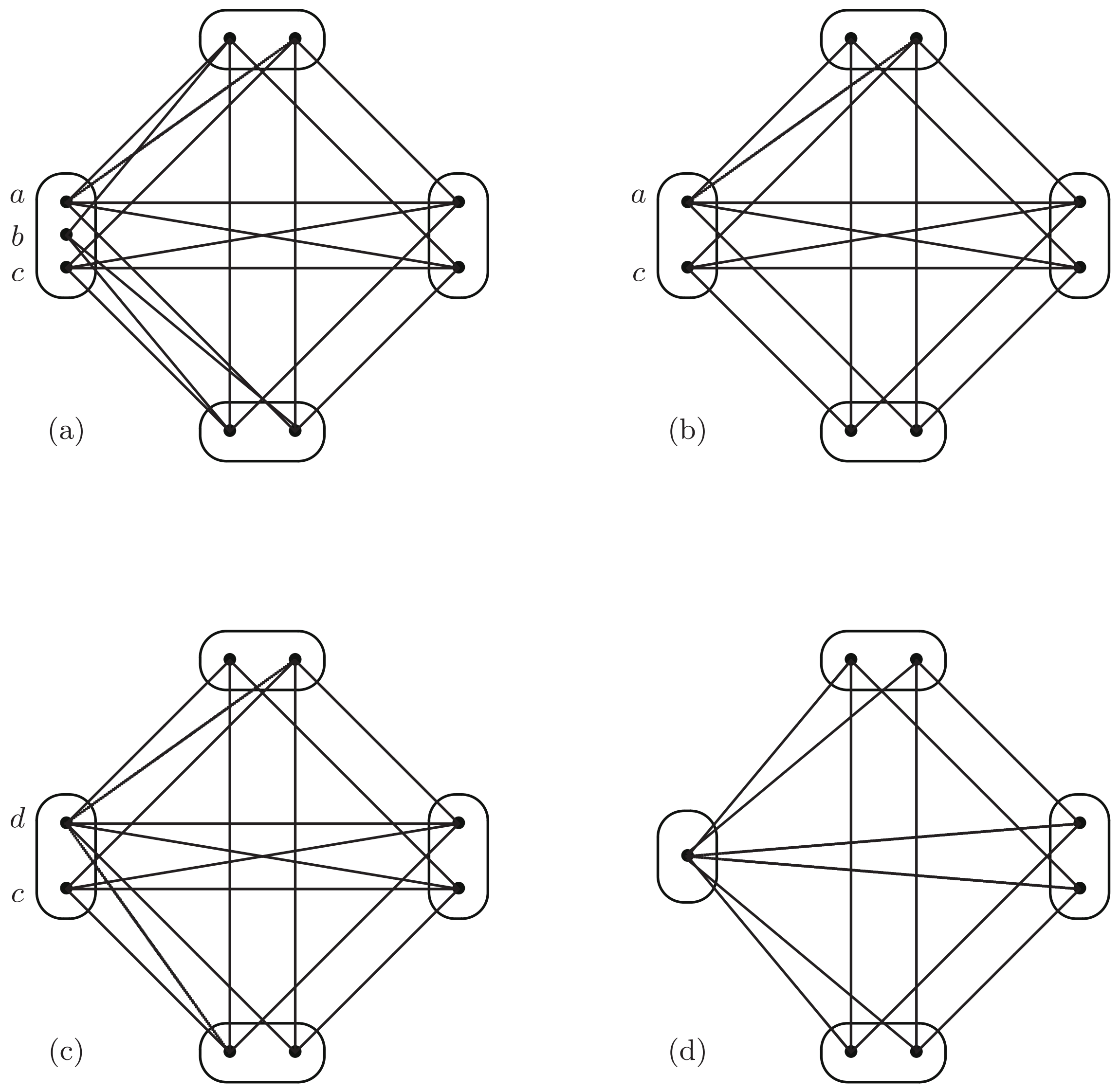

Fig. 3. (a) An instance in which applying AC leads to one elimination (the value b) (as shown in (b)), but applying BTP merging leads to two eliminations, namely a with b (shown in (c)) and then d with c (shown in (d)).

In order to prevent the possibility of merging both (a_i, b_i) and (a'_i, b'_i), we define the following constraints for z_i, x_{ij} and y_{ij}: $\forall j$ $R_{x_{ij}z_i} = \{(b_i, c_i)\}$ and $R_{y_{ij}z_i} = \{(b'_i, c_i)\}$; $\forall j$ $\forall$ k $R_{x_{ij}y_{ik}} = \{(a_i, a'_i)\}$. These constraints are shown in Figure 4(a) for a single j (where a pair of points not joined by a solid line are incompatible). By this gadget, we create a broken triangle on each y_{ij} when merging values in the x_{ij} and vice versa.

Then for each clause $C_i = (X_j, X_k, X_l)$, we add the following constraints in order to have at least one of the literals X_j, X_k, X_l true: $R_{y_{ji}y_{ki}} = \{(a'_j, b'_k)\}$, $R_{y_{ki}y_{li}} = \{(a'_k, b'_l)\}$ and $R_{y_{li}y_{ji}} = \{(a'_l, b'_j)\}$. This construction, shown in Figure 4(b), is such that it allows two mergings on the variables y_{ji}, y_{ki}, y_{li} before a broken triangle is created. For example, merging a'_j, b'_j and then a'_k, b'_k creates a broken triangle on a'_i, b'_i. So a third merging is not possible.

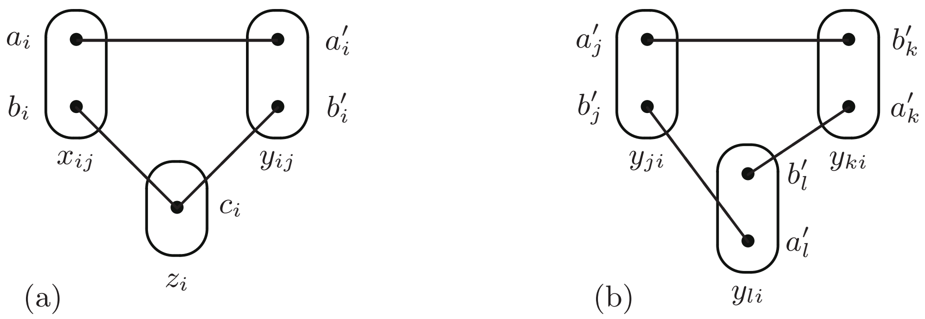

Fig. 4. (a) Representation of the variable X_i and its negation (by the possibility of performing a merge in $\mathcal{D}(x_{ij})$ or $\mathcal{D}(y_{ij})$, respectively, according to rules (1),(2)). (b) Representation of the clause $(X_j \vee X_k \vee X_l)$. Pairs of points joined by a solid line are compatible and incompatible otherwise.

If the clause C_i contains a negated literal $\overline{X_j}$ instead of X_j, it suffices to replace y_{ji} by x_{ji}. Indeed, Figure 5 shows the construction for the clause $(\overline{X_j} \vee X_k \vee \overline{X_l})$ together with the gadgets for each variable. The maximum number of mergings that can be performed are one per occurrence of each variable in a clause, which is exactly $3 \times M$. Given a sequence of $3 \times M$ mergings in the CSP instance, there is a corresponding solution to I_{3SAT} given by (1) and (2). The above reduction allows us to code I_{3SAT} as the problem of testing the existence of a sequence of $k = 3 \times M$ mergings in the corresponding instance I_{CSP}. This reduction being polynomial, we have proved the NP-completeness of the problem of determining whether k BTP merges are possible in a boolean binary CSP instance.

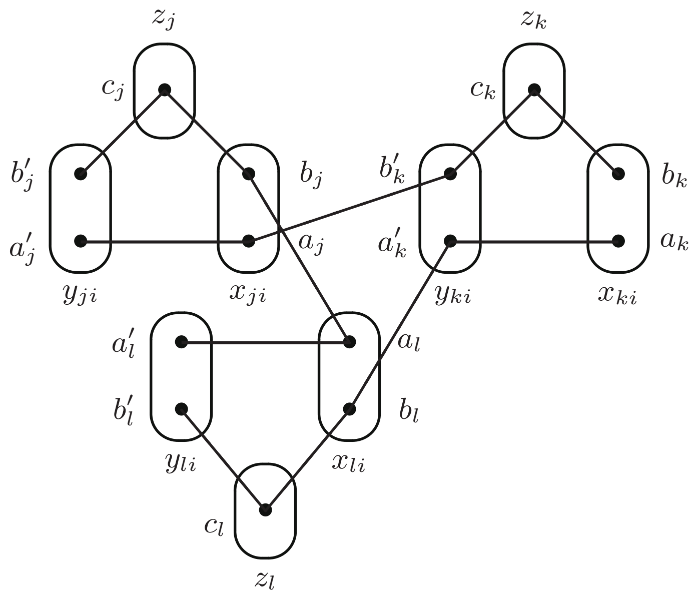

Fig. 5. Gadget representing the clause $(\overline{X_j} \vee X_k \vee \overline{X_l})$.

The reduction given in the proof of Theorem 1 supposes that no arc consistency operations are used. We will now show that it is possible to modify the reduction so as to prevent the elimination of any values in the instance I_{CSP} by arc consistency, even when the maximum size of the domains d is bounded by a constant as small as 3. Recall that an arc-consistent instance remains arc-consistent after any number of BTP-mergings.

Theorem 2. *The problem of determining if it is possible to perform k BTP-mergings in an arc-consistent binary CSP instance is NP-complete, even when only considering binary CSP instances where the size of the domains is bounded by 3.*

Proof. In order to ensure arc consistency of the instance I_{CSP}, we add a new value d_i to the domain of each of the variables x_{ij}, y_{ij}, z_i. However, we cannot simply make d_i compatible with all values in all other domains, because this would allow all values to be merged with d_i, destroying in the process the semantics of the reduction.

In the three binary constraints concerning the triple of variables x_{ij}, y_{ij}, z_i, we make d_i compatible with all values in the other two domains *except* d_i. In other words, we add the following tuples to constraint relations, as illustrated in Figure 6:

- $\forall i \forall j$, (a_i, d_i), (b_i, d_i), $(d_i, c_i) \in R_{x_{ij} z_i}$
- $\forall i \forall j$, (a'_i, d_i), (b'_i, d_i), $(d_i, c_i) \in R_{y_{ij} z_i}$
- $\forall i \forall j$, (a_i, d_i), (b_i, d_i), (d_i, a'_i), $(d_i, b'_i) \in R_{x_{ij} y_{ij}}$

This ensures arc consistency, without creating new broken triangles on a_i, b_i or a'_i, b'_i, while at the same time preventing BTP-merging with the new value d_i. It is important to note that even after BTP-merging of one of the pairs a_i, b_i or a'_i, b'_i, no BTP-merging is possible with d_i in $\mathcal{D}(x_{ij})$, $\mathcal{D}(y_{ij})$ or $\mathcal{D}(z_i)$ due to the presence of broken triangles on this triple of variables. For example, the pair of values $a_i, d_i \in \mathcal{D}(x_{ij})$ belongs to a broken triangle on $c_i \in \mathcal{D}(z_i)$ and $d_i \in \mathcal{D}(y_{ij})$, and this broken triangle still exists if the values $a'_i, b'_i \in \mathcal{D}(y_{ij})$ are merged.

We can then simply make d_i compatible with all values in the domain of all variables outside this triple of variables. This ensures arc consistency, and does

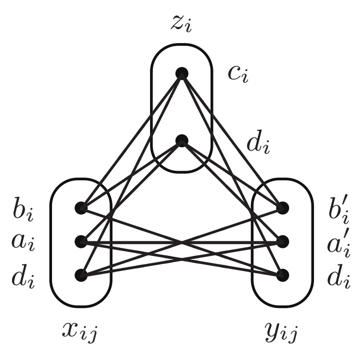

Fig. 6. Ensuring arc consistency between the variables z_i, y_{ij}, x_{ij} by addition of new values d_i.

not introduce any broken triangles on a_i, b_i or a'_i, b'_i. With these constraints we ensure arc consistency without changing any of the properties of I_{CSP} used in the reduction from 3-SAT described in the proof of Theorem 1. For each pair of values $a_i, b_i \in \mathcal{D}(x_{ij})$ and $a'_i, b'_i \in \mathcal{D}(y_{ij})$, no new broken triangle is created since these two values always have the same compatibility with all the new values d_k. As we have seen, the constraints shown in Figure 6 prevent any merging of the new values d_k.

Corollary 1. *The problem of determining if it is possible to perform k value eliminations by arc consistency and BTP-merging in a binary CSP instance is NP-complete, even when only considering binary CSP instances where the size of the domains is bounded by 3.*

From a practical point of view, an obvious question concerns the existence of a non-optimal but nevertheless useful heuristic for choosing the order of BTP-merges. Imagine that a heuristic concerning the order in which to apply BTP-merges involves first finding all possible merges before choosing between them. If this is the case, then once a merge has been chosen and performed, the list of possible merges has to be recalculated. This process is thus already an order of magnitude slower than the simple technique (applied in the experiments in [4]) consisting of performing a merge as soon as it is detected.

4 Optimal Sequence of BTP-mergings at a Single Variable

We now show that even when only considering a single domain, finding the optimal order of BTP-mergings is NP-Complete. For simplicity of presentation, we first prove this for the case in which the instance is not necessarily arc-consistent. We then prove a tighter version for arc-consistent instances.

Theorem 3. *The problem of determining if it is possible to perform k BTP-mergings within a same domain in a binary CSP instance is NP-Complete.*

Proof. For a given sequence of k BTP-mergings within a domain of a binary CSP instance I, verifying if this sequence is correct can be performed in $\mathcal{O}(kN^2d^2)$ time, with N being the number of variables in I and d being the size of the largest domain in I, because looking for broken triangles for a given couple of values takes $\mathcal{O}(N^2d^2)$. As we can verify a solution in polynomial time, the problem of determining if it is possible to perform k BTP-mergings within a single domain in a binary CSP instance is in NP. So to complete the proof of NP-Completeness, it suffices to give a polynomial-time reduction from the well-known SAT problem.

Let I_{SAT} be a SAT instance with n variables $X_1, X_2, \ldots, X_n$ and m clauses $C_1, C_2, \ldots, C_m$. We reduce I_{SAT} to a binary CSP instance I_{CSP} containing a variable v_0 such that I_{SAT} is satisfiable if and only if we can make $k = n + m$ BTP-mergings within $\mathcal{D}(v_0)$. I_{CSP} is defined in the following way:

1. I_{CSP} contains $1+n(2+4m+9(n-1))$ variables $v_0, v_1, v_2, \ldots, v_{n(2+4m+9(n-1))}$.

2. $\mathcal{D}(v_0)$ contains $3\times n+m$ values with the following names: $x_1, x_2, \ldots, x_n, x_1T, x_2T, \ldots, x_nT, x_1F, x_2F, \ldots, x_nF, c_1, c_2, \ldots, c_m$. All other domains in I_{CSP} only contain one value.
3. $\forall i \in [1, n]$, x_iT and x_iF can never be BTP-merged. The idea here is to allow exactly one BTP-merging among the three values x_i, x_iT and x_iF: x_i and x_iT if X_i is assigned True in the SAT instance I_{SAT}, x_i and x_iF if X_i is assigned False instead.
4. $\forall(i,j) \in [1,n] \times [1,m]$ such that C_j does not contain X_i (respectively $\overline{X_i}$), c_j can never be BTP-merged with x_iT (respectively x_iF).
5. $\forall(i,j) \in [1,n]\times[1,m]$ such that C_j contains X_i (respectively $\overline{X_i}$), c_j can only be BTP-merged with x_iT (respectively x_iF) if either c_j or x_iT (respectively x_iF) has been previously BTP-merged with x_i.
6. $\forall i, j$ with $1 \leq i < j \leq n$, the nine following couples can never be BTP-merged: x_i and x_j, x_i and x_jT, x_i and x_jF, x_iT and x_j, x_iT and x_jT, x_iT and x_jF, x_iF and x_j, x_iF and x_jT, x_iF and x_jF. The idea here is to prevent any BTP-merging between two CSP values corresponding to two different SAT variables.

When we say that two values a and b can never be BTP-merged, it means that we add two variables v_a and v_b, with only one value a' in $\mathcal{D}(v_a)$, and only one value b' in $\mathcal{D}(v_b)$, such that a is compatible with a' and incompatible with b', b is compatible with b' and incompatible with a', a' is compatible with b' and all other edges containing either a' or b' are incompatible. The purpose of making a' (respectively b') incompatible will all values in the instance except a and b' (respectively b and a') is twofold. First, it ensures that no future BTP-merging can establish a compatibility between a' (respectively b') and b (respectively a) and thus destroy the broken triangle. Second, it ensures that the only broken triangle introduced by a' and b' is on a and b, so that the addition of a' and b' does not prevent any other BTP-merging than the one between a and b.

When we say that two values a and b can only be BTP-merged if either a or b has been previously BTP-merged with some third value c, it means that we add two variables v_a and v_b, with only one value a' in $\mathcal{D}(v_a)$, and only one value b' in $\mathcal{D}(v_b)$, such that a is compatible with a' and incompatible with b', b is compatible with b' and incompatible with a', c is compatible with both a' and b', a' is compatible with b' and all other edges containing either a' or b' are incompatible. Here again, the purpose of making a' (respectively b') incompatible will all values in the instance except a, b' and c (respectively b, a' and c) is twofold. First, it ensures that no future BTP-merging that does not include c can establish a compatibility between a' (respectively b') and b (respectively a) and thus destroy the broken triangle. Second, it ensures that the only broken triangle introduced by a' and b' is on a and b, so that the addition of a' and b' does not prevent any other BTP-merging than the one between a and b.

For every couple of values that can never be BTP-merged, and for every couple of values that can only be BTP-merged when one of them has been previously BTP-merged with some third value, we add two new single-valued variables to I_{CSP}. Therefore, the third point in the definition of I_{CSP} adds $2n$

variables to I_{CSP}, the fourth and fifth points in the definition of I_{CSP} add $4nm$ variables to I_{CSP} and the sixth point in the definition of I_{CSP} adds $9n(n-1)$ variables to I_{CSP}. Therefore, the total number of single-valued variables added to I_{CSP} is $n(2+4m+9(n-1))$, as expected from the first point in the definition of I_{CSP}.

- The number of BTP-mergings is limited by $n+m$:
  From the third point in the definition of I_{CSP}, for all $i \in [1,n]$, we can BTP-merge at most once within the triple $\{x_i, x_iT, x_iF\}$. From the sixth point in the definition of I_{CSP}, we cannot BTP-merge two values within $\mathcal{D}(v_0)$ if they are associated to two different SAT variables X_i and X_j. Therefore, we have at most m BTP-mergings remaining, one for each c_j for $1 \leq j \leq m$.
- If we can BTP-merge $n+m$ times, then we have a solution for I_{SAT}:
  Since we have done the maximum number of BTP-mergings, we know that for all $i \in [1,n]$, x_i has been BTP-merged with either x_iT or x_iF, but not both. So we create the following solution for I_{SAT}: $\forall i \in [1,n]$, we assign $True$ to X_i if x_i and x_iT have been BTP-merged, and $False$ otherwise. From the fourth and fifth points in the definition of I_{CSP}, we know that for each j in $[1,m]$, C_j is satisfied by the literal associated with the value C_j has been BTP-merged with.
- If we have a solution for I_{SAT}, then we can BTP-merge $n+m$ times:
  $\forall i \in [1,n]$, we BTP-merge x_i with x_iT if $True$ has been assigned to X_i, with x_iF otherwise. $\forall (i,j) \in [1,n] \times [1,m]$, we BTP-merge c_j and the value that x_i has been BTP-merged with if X_i is satisfied in C_j and c_j has not been BTP-merged yet. From the fifth point in the definition of I_{CSP}, we know we can BTP-merge each c_j once.

Therefore I_{SAT} is satisfiable if and only if we can perform $k = n+m$ BTP-mergings within $\mathcal{D}(v_0)$, and we have the result.

We now generalise the result of Theorem 3 to arc-consistent binary CSP instances.

Theorem 4. *The problem of determining if it is possible to perform k BTP-mergings within a same domain in an arc-consistent binary CSP instance is NP-Complete.*

Proof. We transform the binary CSP instance I_{CSP} from the proof of Theorem 3 into an arc-consistent binary CSP instance I'_{CSP}. To do so, we add a new value d_i in $\mathcal{D}(v_i)$ for $1 \leq i \leq n(2+4m+9(n-1))$ such that all d_i are incompatible with each other and compatible with all other points in I'_{CSP}. This ensures arc consistency. It remains to show that:

1. For any couple of values $(a,b) \in \mathcal{D}(v_0)$, adding the values d_i does not create the broken triangle on a and b, even if a or b is the result of a previous BTP-merging:
   Suppose that we have two values $a, b \in \mathcal{D}(v_0)$ such that adding the values d_i

creates a broken triangle on a and b. Let $a' \in \mathcal{D}(v_a)$ and $b' \in \mathcal{D}(v_b)$ be the other two values forming the broken triangle. Since it was the new values d_i that created this particular broken triangle, either a' or b' is one of the d_i. Without loss of generality, we assume that a' is one of the d_i. But since the d_i are compatible with all values from $\mathcal{D}(v_0)$, both a and b are compatible with a', even if a or b is the result of a previous BTP-merging. Therefore, there cannot be any broken triangle on a and b caused by the new values d_i.

2. For all $i \in [1, n(2 + 4m + 9(n-1))]$, it is never possible to BTP-merge the two values in $\mathcal{D}(v_i)$:
   We assume, for simplicity of presentation and without loss of generality, that the SAT instance I_{SAT} has more than one variable and that no clause contains both a variable and its negation. Let $i \in [1, n(2 + 4m + 9(n-1))]$. Let a and d_i be the two points in $\mathcal{D}(v_i)$. From the proof of Theorem 3, we know that a is compatible with only one value from $\mathcal{D}(v_0)$. Let b be this value. If b is associated with one of the SAT variables from I_{SAT}, then from the sixth point in the definition of I_{CSP} in the proof of Theorem 3 we know that there is at least one value $c \in \mathcal{D}(v_0)$ that can never be BTP-merged with b, and therefore will always be incompatible with a. If on the other hand c is associated with one of the SAT clauses from I_{SAT}, then from the fourth point in the definition of I_{CSP} in the proof of Theorem 3 we know that there is at least one value $c \in \mathcal{D}(v_0)$ that can never be BTP-merged with b, and therefore will always be incompatible with a. Therefore, we have a value $c \in \mathcal{D}(v_0)$ that is always incompatible with a, even if c is the result of a previous BTP-merging. Let $j \in [1, n(2 + 4m + 9(n-1))]$, such that $j \neq i$. Since the d_i are incompatible with each other, and compatible with all other values in I'_{CSP}, then d_j is compatible with both a and c, and d_i is compatible with c and incompatible with d_j. Therefore we have a broken triangle on a and d_i that can never be destroyed. Therefore a and d_i can never be BTP-merged and we have the result.

One motivation for studying the single-variable version of the problem was that if all values in $\mathcal{D}(x)$ can be BTP-merged, then the variable x can be eliminated since its domain becomes a singleton. Our proof of NP-hardness in the single-variable case relied on a large domain which was not actually reduced to a singleton. There remains therefore an interesting open question concerning the complexity of eliminating the largest number of variables by sequences of BTP-merging operations.

5 Virtual Interchangeability and Neighbourhood Substitution

Since testing whether two values $a, b \in \mathcal{D}(x)$ are BTP-mergeable requires testing all pairs of assignments to all pairs of distinct variables $y, z \neq x$, it is natural to investigate weaker versions which are less costly to test. Two such weaker versions are neighbourhood substitutability [6] and virtual interchangeability [8].

Given two values $a, b \in \mathcal{D}(x)$, a is neighbourhood substitutable by b (NS), and so can be merged with b, if for all variables $y \neq x$, $\forall c \in \mathcal{D}(y)$, $(a, c) \in R_{xy} \Rightarrow (b, c) \in R_{xy}$. Two values $a, b \in \mathcal{D}(x)$ are virtual interchangeable (VI), and so can be merged, if for all variables $y \neq x$ *except at most one*, $\forall c \in \mathcal{D}(y)$, $(a, c) \in R_{xy} \Leftrightarrow (b, c) \in R_{xy}$. Applying both VI and neighbourhood substitution (NS) operations until convergence provides a weaker and less time-costly alternative to applying BTP-merging operations until convergence. An interesting question is therefore whether it is possible to find in polynomial time an optimal (i.e. longest) sequence of VI and NS operations. The following theorem shows that this problem is in fact also NP-hard.

Theorem 5. *Determining whether there exists a sequence of VI and NS operations of length k that can be applied to a binary CSP instance is NP-complete.*

Proof. Since checking the validity of a sequence of VI and NS operations can be achieved in polynomial time, the problem is in NP. To complete the proof we demonstrate a polynomial reduction from 3SAT. Let I_{3SAT} be an instance of 3SAT. We will show how to construct a binary CSP instance I_{CSP} and a value k so that it is possible to perform k merges by VI and NS if and only if I_{3SAT} is satisfiable.

For each variable x in I_{3SAT}, we introduce Boolean variables called x and $\overline{x}$ in I_{CSP}: variable x in I_{3SAT} is assigned true (respectively, false) if and only if the two domain values in $\mathcal{D}(x)$ (respectively, $\mathcal{D}(\overline{x})$) are merged in the corresponding sequence of merges in I_{CSP}. Figure 7(a) shows the gadget for choosing the truth value for x. The variables x and $\overline{x}$ are both connected to another variable, not shown in this figure: this prevents the values in their domains being merged before the values in the domains of t or t' are merged. Initially, the only merges (by VI and NS) which can occur are in the domain of variable s: we can either merge values 1 and 2 or values 2 and 3. Once one of these merges is performed, the other it not possible. Figure 7 shows the propagation of merges which occurs in the second of these cases. Figure 7(b) shows the result of this merge in $\mathcal{D}(s)$. In Figure 7(b) the only merge that is possible is the merging of values 1 and 3 (by NS) in $\mathcal{D}(t)$. The result is shown in Figure 7(c). Now, the two values in the domain of x can be merged (by VI) since x is constrained by a single other variable (not shown in the figure). It is important to note that no other merges are possible. By a similar chain of propagations, if we had chosen to merge 1 and 2 in $\mathcal{D}(s)$, then the values 1 and 3 in $\mathcal{D}(t')$ would have been merged, and finally the two values in $\mathcal{D}(\overline{x})$. This gadget therefore allows us to choose a truth value for the corresponding variable x of I_{3SAT}.

In order to code the instance I_{3SAT}, we need to be able to have several copies of each variable. This is achieved by the gadget shown in Figure 8(a). The variables x_1 and x_2 are each assumed to be constrained by a single other variable not shown in the figure. The variable x is the variable in the gadget of Figure 7. If the values in $\mathcal{D}(x)$ are merged then this allows the merging (by VI) of the pair of values $0, 1 \in \mathcal{D}(u)$ and the merging of the pair of values 2,3. In the resulting instance, the two values in the domain of x_i $(i = 1, 2)$ can be merged.

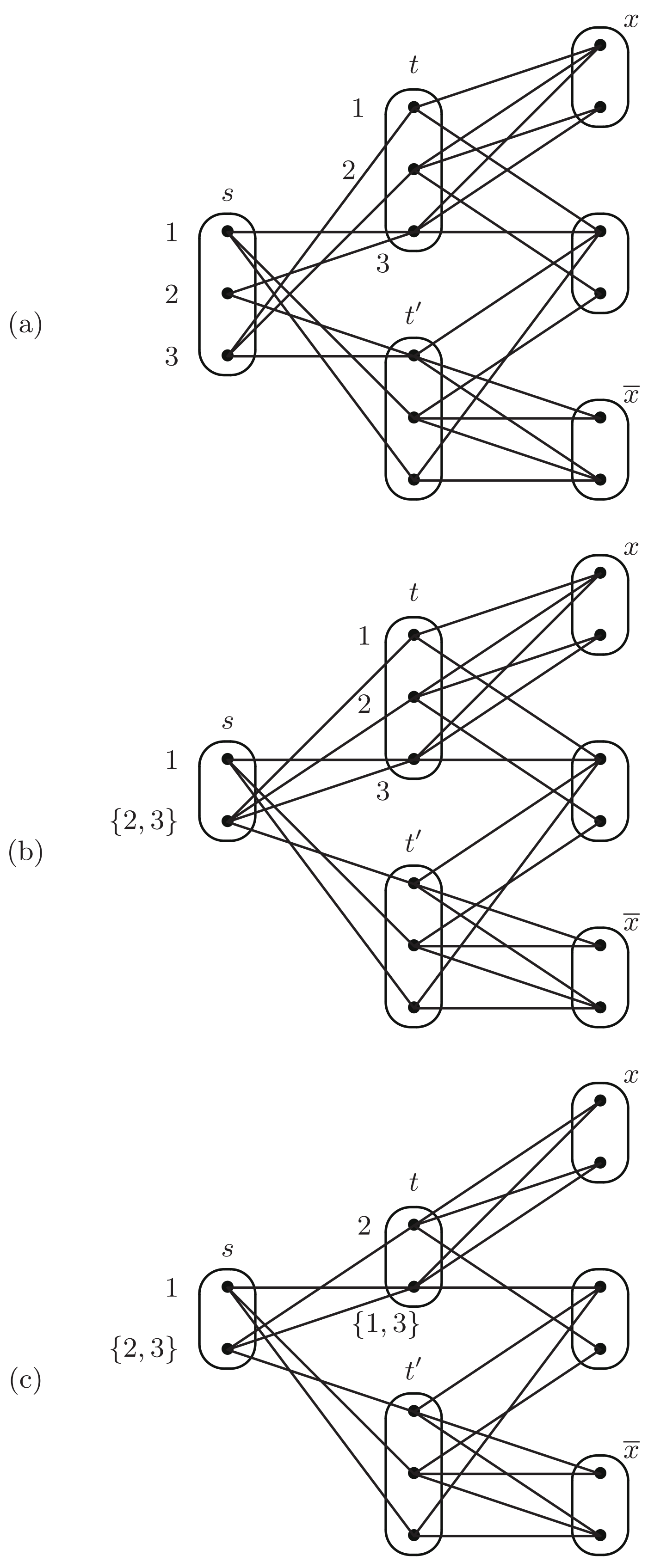

Fig. 7. (a) Gadget for choosing a truth value for x: true if the two values in $\mathcal{D}(x)$ are merged; false if the two values in $\mathcal{D}(\overline{x})$ are merged. This same gagdet (b) after merging the values 2 and 3 in $\mathcal{D}(s)$, then (c) after merging the values 1 and 3 in $\mathcal{D}(t)$.

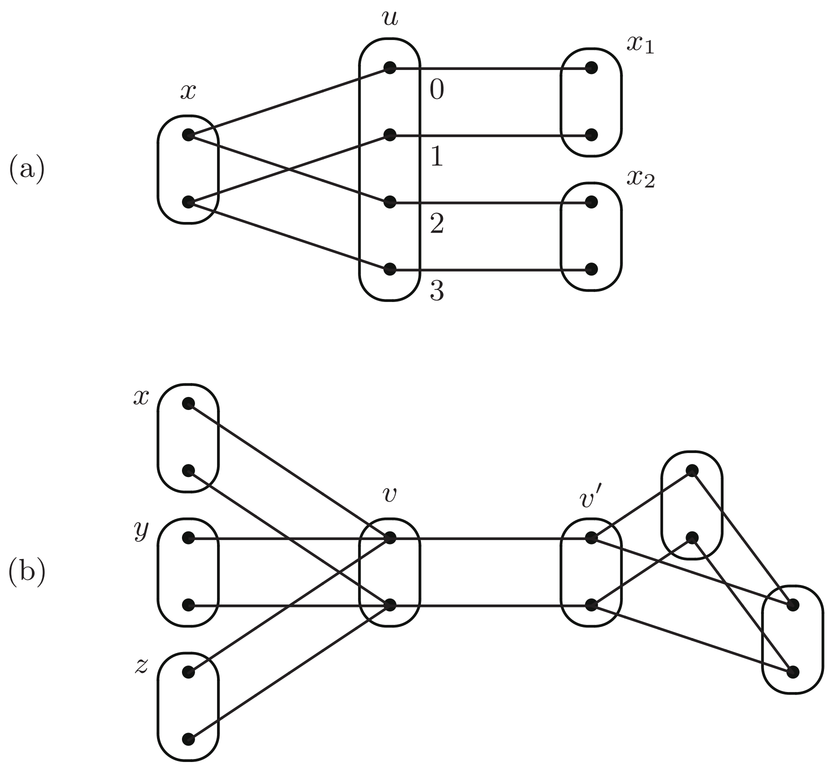

Fig. 8. (a) Gadget for making copies of a variable x: if the two values in $\mathcal{D}(x)$ can be merged, then the two values in $\mathcal{D}(x_1)$ and the two values in $\mathcal{D}(x_2)$ can be merged. (b) Gadget used in the simulation of a clause: the variable v is true (i.e. the two values in its domain can be merged) if and only if x, y, z are all true.

This gadget therefore allows us to make multiple copies of the variable x all with the same truth value.

To complete the proof, we now show how to code each clause c of I_{3SAT}. There are exactly seven assignments to the variables in c which satisfy this clause. For each of these assignments, we add a gadget of the form shown in Figure 8(b). The variables x, y, z are the output of the gadgets introduced above and correspond to the variables x, y, z occurring in the clause c in I_{3SAT}. In the example shown in the figure, the satisfying assignment is $x = y = z = \mathit{true}$. When the two values in each of the domains of these three variables can be merged (and only in this case), the values in the domain of v can also be merged. The triangle of variables to the right of v in Figure 8(b) prevents the merging of the values in $\mathcal{D}(v)$ when only two of the three variables x, y, z are assigned true.

In order to have the same number of copies of x and $\overline{x}$ in our construction, we also add a gadget similar to Figure 8(b) for the one non-satisfying assignment to the three variables of the clause c: in this case, the variable v is constrained by two other variables (as is the variable v' in Figure 8(b)) which prevents the merging of the values in $\mathcal{D}(v)$.

Suppose that there are n variables and m clauses in I_{3SAT}. The maximum total number of merges which can be performed in I_{CSP} is 3 per gadget shown in

Figure 7, 4 per gadget shown in Figure 8(a) and 1 per gadget shown in Figure 8(b) (provided the gadget corresponds to a truth assignment which satisfies the clause c). Each clause c requires four copies of each of the three variables x occurring in c (as well as four copies of $\overline{x}$). For each copy of each literal assigned the value true, there are 4 merges in the gadget of Figure 8(a). For the first occurrence of each variable, produced by the gadget of Figure 7, there is one less merge (3 instead of 4). Finally, for each satisfied clause there is one merge. This implies that we can perform a total of $k = 48m - n + m = 49m - n$ merges in I_{CSP} if and only if I_{3SAT} is satisfiable. Since this reduction is clearly polynomial, this completes the proof.

6 Conclusion

We have investigated the possibility of maximising the number of domain reduction operations in binary CSP instances by choosing an optimal order in which to apply them. Whereas for consistency and neighbourhood-substitution operations, the number of domain reduction operations can be maximised in polynomial time, the problem becomes NP-hard when we allow merging operations, such as virtual interchangeability or BTP-merging. We emphasise that this does not detract from the possible utility of such value-merging operations in practice, which is an independent question.

Different tractable subproblems of binary CSP have been defined based on the absence of certain broken triangles [3,5,7]. Instances can be solved by eliminating variables one by one and, in each case, the recognition of instances in the tractable class can be achieved in polynomial time by a greedy algorithm since the elimination of one variable cannot prevent the elimination of another variable. BTP-merging, on the other hand, performs reduction operations on a lower level than the elimination of variables. Given the NP-completeness results in this paper, recognizing those instances which can be reduced to a trivial problem with only singleton domains by some sequence of BTP-merges is unlikely to be tractable, but this remains an open problem.

References

1. Cohen, D.A., Cooper, M.C.: Guillaume Escamocher and Stanislav Živný, Variable and Value Elimination in Binary Constraint Satisfaction via Forbidden Patterns. J. Comp. Systems Science (2015). http://dx.doi.org/10.1016/j.jcss.2015.02.001
2. Martin, C.: Cooper, Fundamental Properties of Neighbourhood Substitution in Constraint Satisfaction Problems. Artif. Intell. **90**(1–2), 1–24 (1997)
3. Cooper, M.C., Jeavons, P.G., Salamon, A.Z.: Generalizing constraint satisfaction on trees: Hybrid tractability and variable elimination. Artif. Intell. **174**(9–10), 570–584 (2010)
4. Cooper, M.C., El Mouelhi, A., Terrioux, C., Zanuttini, B.: On broken triangles. In: O'Sullivan, B. (ed.) CP 2014. LNCS, vol. 8656, pp. 9–24. Springer, Heidelberg (2014)

5. Cooper, M.C.: Beyond consistency and substitutability. In: O'Sullivan, B. (ed.) CP 2014. LNCS, vol. 8656, pp. 256–271. Springer, Heidelberg (2014)
6. Freuder, E.C.: Eliminating interchangeable values in constraint satisfaction problems. In: Proceedings AAAI 1991, pp. 227–233 (1991)
7. Jégou, P., Terrioux, C.: The extendable-triple property: a new CSP tractable class beyond BTP. In: AAAI (2015)
8. Likitvivatanavong, C., Yap, R.H.C.: Eliminating redundancy in CSPs through merging and subsumption of domain values. ACM SIGAPP Applied Computing Review **13**(2) (2013)