Artificial Intelligence 234 (2016) 196–218

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Broken triangles: From value merging to a tractable class of general-arity constraint satisfaction problems

Martin C. Cooper [a,*], Aymeric Duchein [a], Achref El Mouelhi [b], Guillaume Escamocher [c], Cyril Terrioux [b], Bruno Zanuttini [d]

[a] IRIT, Université de Toulouse, CNRS, INPT, UPS, UT1, UT2J, France
[b] Aix-Marseille Université, CNRS, ENSAM, Université de Toulon, LSIS UMR 7296, Marseille, France
[c] INSIGHT Centre for Data Analytics, University College Cork, Cork, Ireland
[d] GREYC, UMR 6072, Normandie Université, UNICAEN, CNRS, ENSICAEN, Caen, France

ARTICLE INFO

Article history:
Received 17 September 2015
Received in revised form 1 February 2016
Accepted 3 February 2016
Available online 4 February 2016

Keywords:
CSP
Constraint satisfaction
Domain reduction
Tractable class
Hybrid tractability
NP-completeness
Global constraints

ABSTRACT

A binary CSP instance satisfying the broken-triangle property (BTP) can be solved in polynomial time. Unfortunately, in practice, few instances satisfy the BTP. We show that a local version of the BTP allows the merging of domain values in arbitrary instances of binary CSP, thus providing a novel polynomial-time reduction operation. Extensive experimental trials on benchmark instances demonstrate a significant decrease in instance size for certain classes of problems. We show that BTP-merging can be generalised to instances with constraints of arbitrary arity and we investigate the theoretical relationship with resolution in SAT. A directional version of general-arity BTP-merging then allows us to extend the BTP tractable class previously defined only for binary CSP. We investigate the complexity of several related problems including the recognition problem for the general-arity BTP class when the variable order is unknown, finding an optimal order in which to apply BTP merges and detecting BTP-merges in the presence of global constraints such as AllDifferent.

1. Introduction

At first sight one could assume that the discipline of constraint programming has come of age. On the one hand, efficient solvers are regularly used to solve real-world problems in diverse application domains while, on the other hand, a rich theory has been developed concerning, among other things, global constraints, tractable classes, reduction operations and symmetry. However, there often remains a large gap between theory and practice, which is perhaps most evident when we look at the large number of deep results concerning tractable classes which have yet to find any practical application. The research reported in this paper is part of a long-term project to bridge the gap between theory and practice. Our aim is not only to develop new tools but also to explain why present tools work so well.

Most research on tractable classes has been based on classes defined by placing restrictions either on the types of constraints [1,2] or on the constraint hyper-graph whose vertices are the variables and whose hyper-edges are the constraint scopes [3,4]. Another way of defining classes of binary CSP instances consists of imposing conditions on the microstruc-

* Corresponding author.
E-mail addresses: cooper@irit.fr (M.C. Cooper), Aymeric.Duchein@irit.fr (A. Duchein), achref.elmouelhi@lsis.org (A. El Mouelhi), guillaume.escamocher@insight-centre.org (G. Escamocher), cyril.terrioux@lsis.org (C. Terrioux), bruno.zanuttini@unicaen.fr (B. Zanuttini).

http://dx.doi.org/10.1016/j.artint.2016.02.001
0004-3702/

ture, a graph whose vertices are the possible variable-value assignments with an edge linking each pair of compatible assignments [5,6]. If each vertex of the microstructure, corresponding to a variable-value assignment $\langle x, a\rangle$, is labelled (or coloured) by the variable x, then this so-called coloured microstructure retains all information from the original instance. The broken-triangle property (BTP) is a simple local condition on the coloured microstructure which defines a tractable class of binary CSP [7]. The BTP corresponds to forbidding a simple pattern, known as a broken triangle, in the coloured microstructure for a given variable order. Inspired by the BTP, investigation of other forbidden patterns in the coloured microstructure has led to the discovery of new tractable classes [8–10] as well as new reduction operations based on variable or value elimination [11,12]. The BTP itself has also been directly generalised in several different ways. For example, it has been shown that under an assumption of strong path consistency, the BTP can be considerably relaxed since not all broken triangles need be forbidden to define a tractable class [13–15]. Indeed, even without any assumptions of consistency, it is not necessary to forbid all broken triangles [12]. Imposing the BTP in the dual problem leads directly to a tractable class of general-arity CSPs [16]. The BTP has also been generalised to the Broken Angle Property which defines a tractable class of Quantified Constraint Satisfaction Problems [17].

In this paper we show that the absence of broken triangles on a pair of values in a domain allows us to merge these two values while preserving the satisfiability of the instance. Furthermore, given a solution to the reduced instance, it is possible to find a solution to the original instance in linear time (Section 3). We then investigate the interactions between arc consistency and BTP-merging operations (Section 4) and show that it is NP-hard to find the best sequence of BTP-merging (and arc consistency) operations (Section 5). The effectiveness of BTP-merging in reducing domains in binary CSP benchmark problems is investigated in Section 6. In the second half of the paper we consider general-arity CSPs. Section 7 shows how to generalise BTP-merging to instances containing constraints of any arity (where all constraints are given in the form of either tables, lists of compatible tuples or lists of incompatible tuples). We then go on to consider global constraints, and in particular the AllDifferent constraint, in Section 8. Finally, a directional version of the general-arity BTP allows us to define a tractable class of general-arity CSP instances which is incomparable with the tractable class obtained by directly imposing the BTP in the dual [16] (Section 9). However, on the negative side, we then show that it is NP-complete to determine the existence of a variable order for which an instance falls into this tractable class. The results of Sections 3, 7, 9 and Sections 4, 5 first appeared in two conference papers (respectively [18] and [19]).

2. The Constraint Satisfaction Problem

For simplicity of presentation we use two different representations of constraint satisfaction problems. In the binary case, our notation is fairly standard, whereas in the general-arity case we use a notation close to the representation of SAT instances. This is for presentation only, though, and our algorithms do *not* need instances to be represented in this manner.

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 $x \in X$,
- 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.

For simplicity of presentation, Definition 1 assumes that there is exactly one constraint relation for each pair of variables. The number of constraints e is the number of pairs of variables x, y such that $R_{xy} \neq \mathcal{D}(x) \times \mathcal{D}(y)$. An instance I is *arc consistent* if for each pair of distinct variables $x, y \in X$, each value $a \in \mathcal{D}(x)$ has an AC-support at y, i.e. a value $b \in \mathcal{D}(y)$ such that $(a, b) \in R_{xy}$.

In our representation of general-arity CSP instances, we require the notion of *tuple* which is simply a set of variable-value assignments. For example, in the binary case, the tuple $\{\langle x, a\rangle, \langle y, b\rangle\}$ is *compatible* if $(a, b) \in R_{xy}$ and *incompatible* otherwise.

Definition 2. A *(general-arity) CSP instance* I consists of

- a set X of n variables,
- a domain $\mathcal{D}(x)$ of possible values for each variable $x \in X$,
- a set NoGoods(I) consisting of incompatible tuples.

A *partial solution* to I on $Y = \{y_1, \ldots, y_r\} \subseteq X$ is a tuple $t = \{\langle y_1, a_1\rangle, \ldots, \langle y_r, a_r\rangle\}$ such that no subset of t belongs to NoGoods(I). A *solution* is a partial solution on X.

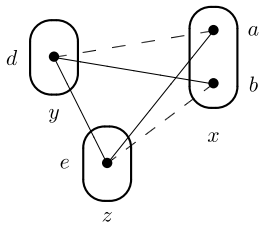

Fig. 1. A broken triangle on two values a, b for a given variable x.

3. Value merging in binary CSP based on the BTP

In this section we consider a method, based on the BTP, for reducing domain size while preserving satisfiability. Instead of eliminating a value, as in classic reduction operations such as arc consistency or neighbourhood substitution, we merge two values. We show that the absence of broken-triangles [7] on two values for a variable x in a binary CSP instance allows us to merge these two values in the domain of x while preserving satisfiability. This rule generalises the notion of virtual interchangeability [20] as well as neighbourhood substitution [21].

It is known that if for a given variable x in an arc-consistent binary CSP instance I, the set of (in)compatibilities (known as a broken-triangle) shown in Fig. 1 occurs for no two values $a, b \in \mathcal{D}(x)$ and no two assignments to two other variables, then the variable x can be eliminated from I without changing the satisfiability of I [7,11]. In figures, each bullet represents a variable-value assignment, assignments to the same variable are grouped together within the same oval and compatible pairs of assignments are linked by solid lines. In Fig. 1 (and in other figures illustrating forbidden patterns) incompatible pairs of assignments are linked by broken lines. Even when this variable-elimination rule cannot be applied, it may be the case that for a given pair of values $a, b \in \mathcal{D}(x)$, no broken triangle occurs. We will show that if this is the case, then we can perform a domain-reduction operation which consists in merging the values a and b.

Definition 3. *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 $\langle x, b\rangle$. A *value-merging condition* is a polytime-computable property $P(x, a, b)$ of assignments $\langle x, a\rangle$, $\langle x, b\rangle$ in a binary CSP instance I such that when $P(x, a, b)$ holds, the instance I' obtained from I by merging $a, b \in \mathcal{D}(x)$ is satisfiable if and only if I is satisfiable.

We now formally define the value-merging condition based on the BTP.

Definition 4. 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.

Proposition 5. *In a binary CSP instance, being BT-free is a value-merging condition. 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.*

Proof. Let I be the original instance and I' the new instance in which a, b have been merged into a new value c. Clearly, if I is satisfiable then so is I'. It suffices to show that if I' has a solution s which assigns c to x, then I has a solution. Let s_a, s_b be identical to s except that s_a assigns a to x and s_b assigns b to x. Suppose that neither s_a nor s_b are solutions to I. Then, there are variables $y, z \in X \setminus \{x\}$ such that $\langle a, s(y)\rangle \notin R_{xy}$ and $\langle b, s(z)\rangle \notin R_{xz}$. By definition of the merging of a, b to produce c, and since s is a solution to I' containing $\langle x, c\rangle$, we must have $(b, s(y)) \in R_{xy}$ and $(a, s(z)) \in R_{xz}$. Finally, $(s(y), s(z)) \in R_{yz}$ since s is a solution to I'. Hence, $\langle y, s(y)\rangle$, $\langle z, s(z)\rangle$, $\langle x, a\rangle$, $\langle x, b\rangle$ forms a broken-triangle, which contradicts our assumption. Hence, the absence of broken triangles on assignments $\langle x, a\rangle$, $\langle x, b\rangle$ allows us to merge these assignments while preserving satisfiability.

Reconstructing a solution to I from a solution s to I' simply requires checking which of s_a or s_b is a solution to I. Checking if s_a or s_b is a solution only requires checking the (at most) $n - 1$ binary constraints that include x. Thus finding a solution to the original instance can be achieved in linear time. □

We can see that the BTP-merging rule, given by Proposition 5, generalises neighbourhood substitution [21]: if b is neighbourhood substitutable by a, then no broken triangle occurs on a, b and merging a and b produces a CSP instance which is identical (except for the renaming of the value a as c) to the instance obtained by simply eliminating b from $\mathcal{D}(x)$. BTP-merging also generalises the merging rule proposed by Likitvivatanavong and Yap [20]. The basic idea behind their rule is that if the two assignments $\langle x, a\rangle$, $\langle x, b\rangle$ have identical compatibilities with all assignments to all other variables except concerning at most one other variable, then we can merge a and b. This is clearly subsumed by BTP-merging.

The BTP-merging operation is not only satisfiability-preserving but, from Proposition 5, we know that we can also reconstruct a solution in polynomial time to the original instance I from a solution to an instance I^m to which we have applied

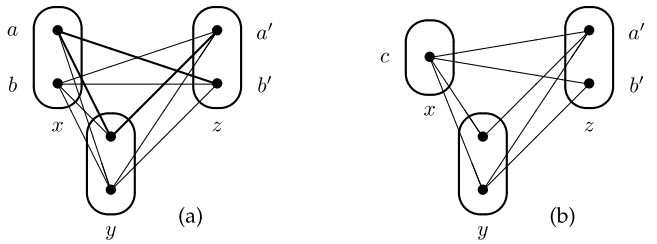

Fig. 2. (a) A broken triangle (shown in bold) exists on values a', b' at variable z. (b) After BTP-merging of values a and b in $\mathcal{D}(x)$, this broken triangle has disappeared.

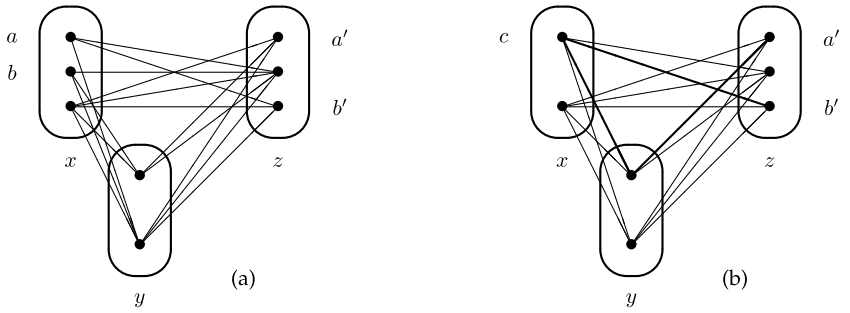

Fig. 3. (a) This instance contains no broken triangle. (b) After BTP-merging of values a and b in $\mathcal{D}(x)$, a broken triangle (shown in bold) has appeared on values $a', b' \in \mathcal{D}(z)$.

a sequence of merging operations until convergence. It is known that for the weaker operation of neighbourhood substitutability, all solutions to the original instance can be generated in $O(N(de+n^2))$ time, where N is the number of solutions to the original instance, n is the number of variables, d the maximum domain size and e the number of constraints [22]. We now show that a similar result also holds for the more general rule of BTP-merging.

Proposition 6. *Let I be a binary CSP instance and suppose that we are given*

- *a sequence of m triples of the form $(x_i, a_i, b_i)_{i=0}^{m-1}$, implicitly defining a sequence of instances $I^0 = I, I^1, \cdots, I^m$ such that I^{i+1} is obtained from I^i by BTP-merging values a_i, b_i for x_i $(i = 0, \ldots, m-1)$,*
- *the set of all N solutions to the instance I^m.*

All solutions to I can then be enumerated with delay $O(mn)$ after a preprocessing step in $O(mnd^2)$ (hence in total time $O(n^2d^3 + Nn^2d)$).

Proof. We start by computing, for each constraint R_{xy} in the original instance I, its successive versions $R_{xy}^{t_1}, \ldots, R_{xy}^{t_{m_{xy}}}$, where $t_1, \ldots, t_{m_{xy}} \in \{1, \ldots, m\}$ record by which BTP-merging operation this version was produced. Since each BTP-merging operation can change only $O(n)$ constraints (those involving x_i), this preprocessing step requires time $O(mnd^2)$.

Now given a solution s to I^i we proceed inductively as follows. If $i = 0$ then we output s, otherwise we test whether s_a or s_b (or both) are solutions to I^{i-1}, where s_a (resp. s_b) is obtained from s by setting x_i to a_i (resp. to b_i), as in the proof of Proposition 5. For each of them found to be a solution to I^{i-1}, we recurse with I^{i-1}. This requires $O(n)$ time per step, since again there are at most $n-1$ constraints to be checked (those involving x_i) and these have been precomputed. Finally, since at each step either s_a or s_b is guaranteed to be a solution to I^{i-1}, we indeed generate solutions to I with delay $O(mn)$. □

The weaker operation of neighbourhood substitution has the property that two different convergent sequences of eliminations by neighbourhood substitution necessarily produce isomorphic instances I_1^m, I_2^m [22]. This is not the case for BTP-merging. Firstly, and perhaps rather surprisingly, BTP-merging can have as a side-effect to eliminate broken triangles. This is illustrated in the 3-variable instance shown in Fig. 2. In order to avoid cluttering up figures with broken lines linking each pair of incompatible assignments, in all figures illustrating binary CSP instances, we use the convention that those pairs of assignments which are not explicitly linked with a solid line are incompatible. The instance in Fig. 2(a) contains a broken triangle on values a', b' for variable z, but after BTP-merging of values $a, b \in \mathcal{D}(x)$ into a new value c, as shown in Fig. 2(b), there are no broken triangles in the instance. Secondly, BTP-merging of two values in $\mathcal{D}(x)$ can introduce a broken triangle on a variable $z \neq x$, as illustrated in Fig. 3. The instance in Fig. 3(a) contains no broken triangle, but after the BTP-merging of $a, b \in \mathcal{D}(x)$ into a new value c, a broken triangle has been created on values $a', b' \in \mathcal{D}(z)$.

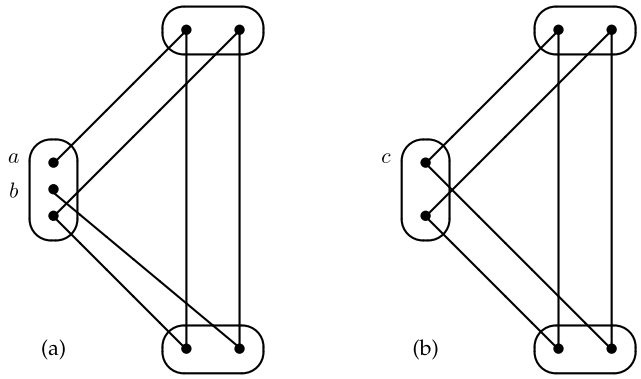

Fig. 4. (a) An instance in which applying AC leads to the elimination of all values (starting with the values a and b), but applying BTP merging leads to just one elimination, namely the merging of a with b (with the resulting instance shown in (b)).

4. Mixing arc consistency and BTP-merging

Given the omnipresence of arc consistency in constraint solvers, it is natural to investigate its relationship and interaction with BTP-merging. Values which can be BTP-merged may or may not be arc consistent. Trivially, two values $a, b \in \mathcal{D}(x)$ which are compatible with all assignments to all other variables can be BTP-merged, but cannot be eliminated by arc consistency. Conversely, if $a \in \mathcal{D}(x)$ has no AC-support at y but otherwise is compatible with all assignments to all other variables, $b \in \mathcal{D}(x)$ has no AC-support at $z \neq y$ but otherwise is compatible with all assignments to all other variables, and $R_{yz} \neq \emptyset$, then a, b can both be eliminated by arc consistency but a, b cannot be BTP-merged. Having established the incomparability of arc consistency and BTP-merging, we now investigate their possible interactions.

We have already observed that BTP-merging is a generalisation of neighbourhood substitutability, 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 [22]:

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) [22].

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 Fig. 4(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 new value c) resulting in the instance shown in Fig. 4(b) in which no more eliminations are possible by AC or BTP-merging.
2. A single elimination by AC may prevent one or more BTP-mergings. An example is given in Fig. 5(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 Fig. 5(b)), whereas if a and b are BTP-merged into a new value d (as shown in Fig. 5(c)) this destroys a broken triangle thus allowing c to be BTP-merged with d (as shown in Fig. 5(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 $d \in \mathcal{D}(z)$ ($z \neq x$) by arc consistency. An example is given in Fig. 6. In this 4-variable instance, initially a and b cannot be BTP-merged (Fig. 6(a)), but after value d is eliminated from $\mathcal{D}(z)$ by AC, the broken triangle has disappeared and a, b can be BTP merged (Fig. 6(b)).

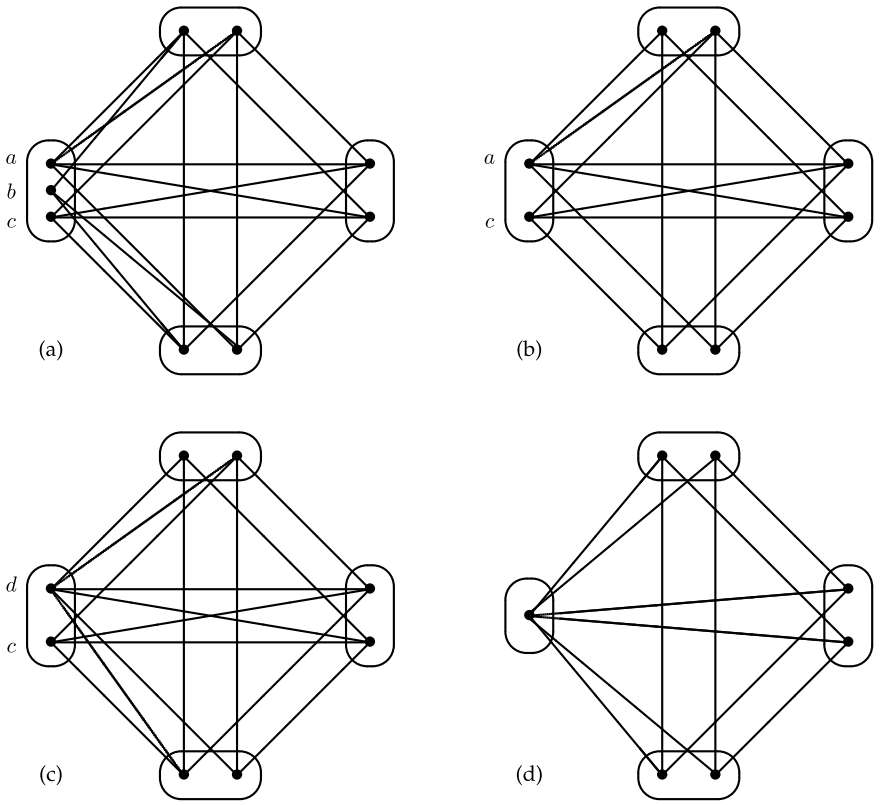

Fig. 5. (a) An instance in which applying AC leads to one elimination (the value b) (as shown in (b)), but applying BTP merging leads to two eliminations, namely a with b (shown in (c)) and then d with c (shown in (d)).

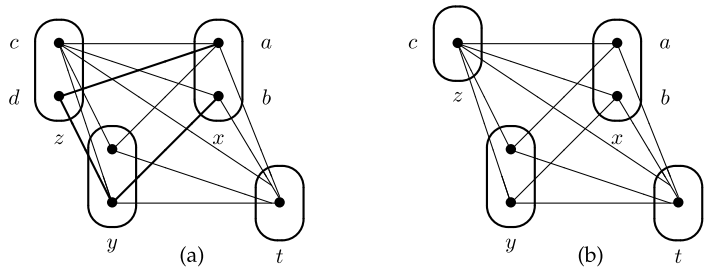

Fig. 6. (a) A broken triangle (shown in bold) exists on values a, b at variable x. (b) After removing value d from $\mathcal{D}(z)$ by AC, this broken triangle has disappeared.

5. The order of BTP-mergings

We saw in Section 3 that BTP-merging can both create and destroy broken triangles. 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 7. *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 $\mathcal{O}(n^2d^2)$. 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.

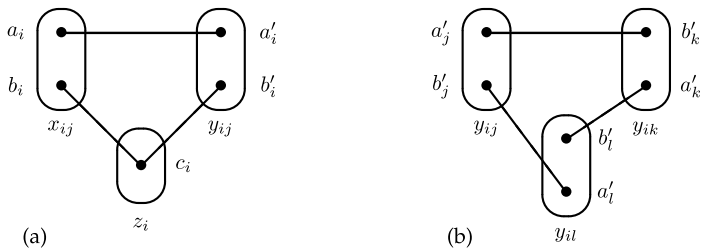

Fig. 7. (a) Representation of the variable X_i and its negation (by the possibility of performing a merge in $\mathcal{D}(x_{ij})$ or $\mathcal{D}(y_{ij})$, respectively, according to rules (1), (2)). (b) Representation of the clause $(X_j \vee X_k \vee X_l)$. Pairs of points joined by a solid line are compatible and incompatible otherwise.

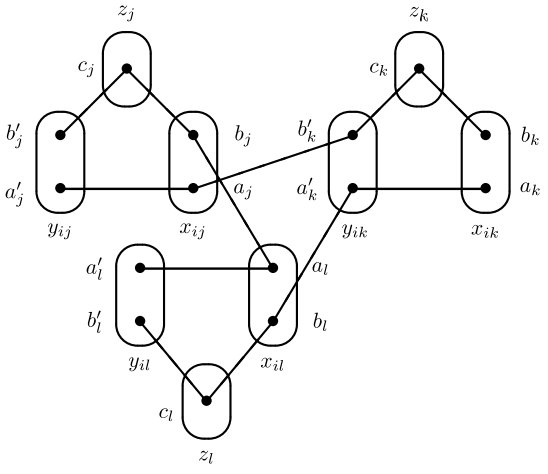

Fig. 8. Gadget representing the clause $(\overline{X_j} \vee X_k \vee \overline{X_l})$.

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:

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

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

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 Fig. 7(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.

The idea is that BTP-merging a_i and b_i in any $\mathcal{D}(x_{ij})$ $(1 \leq j \leq N)$ prevents us from BTP-merging a'_i and b'_i in any $\mathcal{D}(y_{ik})$ $(1 \leq k \leq N)$, thus ensuring that the value of X_i is the same in each clause in which it occurs. If X_i is prevented from being assigned either false or true according to the rules (1) and (2) (because of the clause gadgets described below), then I_{3SAT} will be detected as unsatisfiable since the total number of merges will be less than $3 \times M$.

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: $R_{y_{ij}y_{ik}} = \{(a'_j, b'_k)\}$, $R_{y_{ik}y_{il}} = \{(a'_k, b'_l)\}$ and $R_{y_{il}y_{ij}} = \{(a'_l, b'_j)\}$. This construction, shown in Fig. 7(b), is such that it allows two mergings on the variables y_{ij}, y_{ik}, y_{il} 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.

If the clause C_i contains a negated literal $\overline{X_j}$ instead of X_j, it suffices to replace y_{ij} by x_{ij}. Indeed, Fig. 8 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). To give a concrete example, consider the gadget shown in Fig. 8 representing the clause $\overline{X_j} \vee X_k \vee \overline{X_l}$. This gadget is made up of three triangles of the type shown in Fig. 7(a). To perform three merges in this gadget, we must perform exactly one merging in each of these triangles. For example, if we merge the pairs of values (a_j, b_j), (a_k, b_k) and (a_l, b_l),

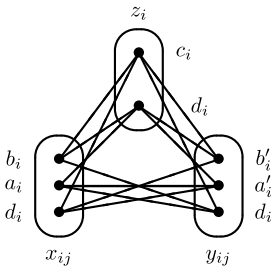

Fig. 9. Ensuring arc consistency between the variables z_i, y_{ij}, x_{ij} by addition of new values d_i.

then this sequence of merges corresponds to the assignment $(X_j, X_k, X_l) = (true, true, true)$ which satisfies the clause. On the other hand, the assignment $(X_j, X_k, X_l) = (true, false, true)$, which does not satisfy the clause, is impossible due to the central triangle of variables (of the type shown in Fig. 7(b)) which prevents us from simultaneously merging the three pairs of values (a_j, b_j), (a'_k, b'_k) and (a_l, b_l).

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. □

The reduction in the proof of Theorem 7 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 8. *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 Fig. 9:

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

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. 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 7. 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 Fig. 9 prevent any merging of the new values d_k. □

Corollary 9. *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.*

A related question concerns the complexity of finding the optimal order of BTP-mergings within the domain of a single variable. It turns out that this too is NP-complete. The proof of this theorem [19] is based on a similar technique to that used in the proof of Theorem 7.

Theorem 10. *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.*

6. Experimental trials

In this section, we study BTP-merging from a practical viewpoint.

6.1. Experimental protocol

To test the utility of BTP-merging we performed extensive experimental trials on CSP benchmark instances available from the International CP Competition.[1] Among the 7272 CSP benchmark instances, we consider all the instances including only binary constraints (namely 3795 instances). For each of these instances, we performed BTP-mergings until convergence with a time-out of one hour. In total, we obtained results for 2944 instances out of 3795 benchmark instances. In the other instances, the search for all BTP-mergings did not terminate within the time-out. Note that some of the considered instances have constraints defined by predicates. In such cases, these constraints are first expressed in extension before applying the BTP-merging algorithm. The runtime of this transformation is included in the reported runtime.

BTP-mergings are performed by checking first for virtual interchangeability and then by looking for BTP-mergeable pairs of values. These two steps are repeated until a fixed point is reached. By so doing, the virtual interchangeability step allows us to merge more quickly some pairs of values since the virtual interchangeability rule is easier to check than the BTP rule. Our experiments (not reported here) have shown that this version of BTP-merging is significantly faster than one presented in [18] while leading to a similar number of mergings.

For the BTP-merging step, we consider the variables according to a given ordering. Among the different variable orderings we tried, we opted in our experimental trials for one which orders the variables according to increasing degree (the degree of a variable being the number of constraints whose scope contains the variable). Note that this ordering differs from the one used in [18] which corresponds to a lexicographical ordering. In practice, we obtain a similar number of mergings with these two orderings but the algorithm is significantly faster with the first one. In general, we observed that the different variable orderings we tried had more impact on runtime than on the number of mergings performed.

For a given variable x, we check for each pair of values $a, b \in D(x)$ whether a, b are BTP-mergeable. If a broken triangle on a, b is found, we save it in a data structure. Then if, later, we have to check again the BTP-mergeability of a, b, we start with this saved broken triangle. If this triangle is still broken, we can immediately deduce that a, b are not BTP-mergeable, thus avoiding some useless checks. On the other hand, if this triangle is no longer broken, we check whether a, b are BTP-mergeable. If no broken triangle occurs on a, b (that is a, b are BTP-mergeable), we immediately merge a, b. This greedy algorithm is a natural choice since by Theorem 8 it is NP-hard to optimise the order of BTP-merges. For efficiency reasons, when merging a, b, we keep one value (assume without loss of generality that a is this value) and delete the other one instead of creating a new value c and removing a and b as evoked in Definition 3. Then a is made compatible with each variable-value assignment compatible with the assignment $\langle x, b\rangle$.

We also implemented the deletion of values by neighbourhood substitution, by virtual interchangeability or by arc-consistency (which is enforced by the AC-2001 algorithm [23]). In the remainder of this section, we denote AC+P the application of AC followed by merging according to the property P where P may be BTP-merging, neighbourhood substitution (NS) or virtual interchangeability (VI). For solving, we use MAC (for Maintaining Arc-Consistency [24]) based on AC-2001 together with the variable ordering heuristic dom/wdeg [25]. The choice of MAC is a natural choice since most state-of-the-art solvers rely on it. All the algorithms are implemented in C++.

The experimentations were performed on 8 Dell PowerEdge M820 blade servers with two processors (Intel Xeon E5-2609 v2 2.5 GHz and 32 GB of memory) under Linux Ubuntu 14.04.

6.2. Comparisons between BTP-merging and AC+BTP-merging

We compare in this subsection the results obtained by BTP-merging and by AC+BTP-merging. First, as shown in Fig. 10, AC+BTP-merging is able to process (i.e. find a fixed point in which no more BTP-mergings are possible) more instances within the time-out than BTP-merging alone. More precisely, AC+BTP-merging succeeds in terminating within the time-out for 2944 instances against 2856 for BTP-merging. In both cases, for more than one third of these instances, some mergings occur. Fig. 11 compares the percentages of values removed by BTP-merging and AC+BTP-merging for the instances for which both BTP-merging and AC+BTP-merging terminate. Clearly, AC+BTP-merging outperforms BTP-merging since the percentage of values removed by AC+BTP-merging is always greater than or equal to the number of values removed by BTP-merging. We can see that for certain types of problem, (AC+)BTP-merging is very effective (more than 90% of deleted values), whereas for others hardly any merging of values occurred. In particular, we have observed that often the instances for which no merging is possible have some disequality constraints (which makes sense, since even a conjunction of disequality constraints as simple as $(x \neq y) \wedge (x \neq z) \wedge (y \neq z)$ with $a, b \in D(x), a \in D(y), b \in D(z)$, induces a broken triangle on a, b). For instance, for the graph colouring instances, (AC+)BTP-mergings only occur when the instances have variables with degree 0 or 1. In contrast, (AC+)BTP-merging is very effective for some real-world instances from frequency assignment problems (`fapp*`, `graph*` or `scen*`) or for some patterned instances (like `BlackHole*` or `os-taillard*`). Note that at best, BTP-merging reduces

[1] http://www.cril.univ-artois.fr/CPAI08.

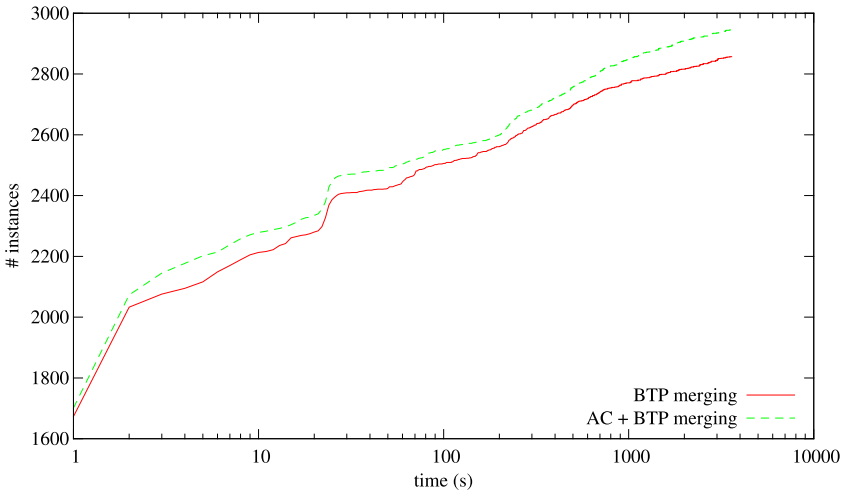

Fig. 10. Number of instances processed by BTP-merging with and without AC preprocessing depending on the elapsed time (in seconds).

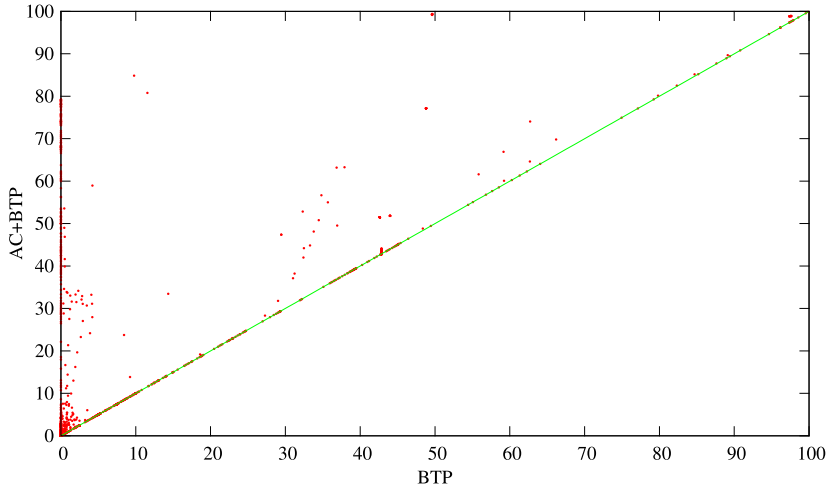

Fig. 11. Percentage of values removed by BTP vs percentage of values removed by AC and BTP-merging (AC+BTP) for each considered instance.

all variable domains to singletons (and so cannot remove all the values in a domain). For example, this is the case for all instances `hanoi*` which satisfy the broken-triangle property [26]. Tables 1 and 2 provide some detailed results for some selected instances. These instances have been selected in such a way that all observed trends are represented.

Regarding runtime, BTP-merging and AC+BTP-merging are often close as shown in Fig. 12. However, for a few instances, such as `langford-4-14`, AC+BTP-merging requires more time than BTP-merging. Such a result is often explained by the fact that the values used to quickly find broken triangles in BTP-merging have been removed by AC in AC+BTP-merging. In contrast, in most cases, achieving an AC preprocessing is useful since it saves time. Moreover, sometimes, it turns out to be very useful since it makes it possible to process more instances. For example, for the instance `fapp25-2230-8`, the values removed by AC make it possible for the BTP-merging step to terminate. Finally, we can note that a large part of the considered instances are processed quickly. Indeed, for about 57% of instances, achieving (AC+)BTP-merging requires less than one second.

6.3. Comparisons with neighbourhood substitution and virtual interchangeability

As shown in Section 3, the BTP-merging rule generalises the notion of neighbourhood substitution as well as virtual interchangeability. Hence, when we compare the percentage of values removed by BTP-merging with the number of values removed by neighbourhood substitution or virtual interchangeability, BTP-merging is always better than or equivalent to neighbourhood substitution or virtual interchangeability. The same trend is observed when the instances are preprocessed by AC. Figs. 13 and 14 compare the percentage of values removed by BTP-merging and by neighbourhood substitution or virtual interchangeability after having enforced AC. Nevertheless, even when the percentages are equal, we have no guarantee that BTP-merging removes the same values as neighbourhood substitution or virtual interchangeability. So, in order to

Table 1
For each selected instance, the number n of variables, the number e of constraints, the total number of values, the number of values removed by neighbourhood substitution (NS) or by virtual interchangeability (VI), the number of values removed by BTP-merging, the number of values for which neighbourhood substitution or virtual interchangeability hold among the values removed by BTP-merging and the runtime in seconds of BTP-merging. A dash means that the information is unknown because the runtime of BTP-merging exceeds the time-out of one hour.

Instance	n	e	# values	NS	VI	BTP			
				# del.	# del.	# del.	# NS	# VI	Time
bqwh-15-106-18_ext	106	597	385	0	0	0	0	0	<0.01
le-450-5a-2-ext	450	5714	900	0	0	0	0	0	0.04
geo50-20-d4-75-100_ext	50	393	1000	0	0	0	0	0	0.05
rand-24-24-276-139-53021_ext	24	276	576	0	0	0	0	0	0.03
langford-4-14	56	1540	3136	0	0	0	0	0	8.18
haystacks-21	441	4430	9261	0	20	20	0	20	6.44
rand-2-40-180-84-900-56_ext	40	84	7200	0	358	358	0	358	23.47
mulsol-i-4-31	185	3946	5735	300	300	300	300	300	8.01
e0ddr2-10-by-5-7	50	265	6215	366	0	366	218	0	224.11
inithx-i-2-28	645	13,979	18,060	2349	2349	2349	2349	2349	220.76
scen1-f9	916	5548	28,596	368	532	1200	251	588	669.72
fapp01-0200-8	200	1108	26,963	0	113	113	0	113	2528.42
ehi-85-297-33_ext	297	4094	2079	0	0	891	0	0	1.18
os-taillard-4-105-5	16	48	2500	812	0	812	406	0	102.73
scen10-w1-f3	680	1138	25,192	893	6626	7419	2411	6770	345.57
BlackHole-4-7-e-0_ext	112	1261	2102	697	887	896	463	887	5.71
BlackHole-4-13-e-1_ext	208	4217	7334	2541	3209	3226	1691	3209	151.31
os-taillard-4-95-7	16	48	2508	1558	0	1573	777	49	123.43
scen4	680	3967	26,856	0	268	3103	214	455	445.89
graph13-w0	916	458	35,176	0	34,260	34,260	17,130	34,260	0.36
large-92-unsat_ext	92	4186	8464	8280	8275	8280	4233	8279	2.83
lard-92-92	92	4186	8556	7163	5303	8347	840	5314	448.00
fapp25-2230-8	2230	11,974	610,084	44,168	–	–	–	–	–
hanoi-5_ext	30	29	6808	0	27	6778	41	6752	0.45

Table 2
For each selected instance, the total number of values, the number of values removed by AC, by AC and neighbourhood substitution (NS) or by virtual interchangeability (VI) after AC preprocessing, the number of values removed by BTP-merging after AC preprocessing, the number of values for which neighbourhood substitution or virtual interchangeability hold among the values removed by BTP-merging and the runtime in seconds of BTP-merging.

Instance	# values	AC	NS	VI	BTP			
		# del.	# del.	# del.	# del.	# NS	# VI	Time
bqwh-15-106-18_ext	385	0	0	0	0	0	0	<0.01
le-450-5a-2-ext	900	0	0	0	0	0	0	0.04
geo50-20-d4-75-100_ext	1000	0	0	0	0	0	0	0.05
rand-24-24-276-139-53021_ext	576	0	0	0	0	0	0	0.03
langford-4-14	3136	1428	0	0	0	0	0	35.18
haystacks-21	9261	0	0	20	20	0	20	6.45
rand-2-40-180-84-900-56_ext	7200	0	0	358	358	0	358	23.50
mulsol-i-4-31	5735	0	300	300	300	300	300	7.93
e0ddr2-10-by-5-7	6215	0	366	0	366	218	0	225.09
inithx-i-2-28	18,060	0	2349	2349	2349	2349	2349	225.93
scen1-f9	28,596	7604	0	383	390	168	383	329.40
fapp01-0200-8	26,963	9155	21	159	174	6	159	1003.46
ehi-85-297-33_ext	2079	2	0	0	889	0	0	1.19
os-taillard-4-105-5	2500	287	810	0	818	404	22	102.44
scen10-w1-f3	25,192	5134	0	6321	6808	2138	6451	190.51
BlackHole-4-7-e-0_ext	2102	280	634	802	802	427	802	2.93
BlackHole-4-13-e-1_ext	7334	793	2422	3007	3007	1623	3007	79.57
os-taillard-4-95-7	2508	451	1346	4	1406	656	122	106.43
scen4	26,856	19,534	0	774	2161	406	924	44.44
graph13-w0	35,176	0	0	34,260	34,260	17,130	34,260	0.36
large-92-unsat_ext	8464	0	8280	8275	8280	4233	8279	2.82
lard-92-92	8556	4350	4114	3878	4114	2494	3886	3.93
fapp25-2230-8	610,084	590,850	5294	14,469	16,449	4209	15,840	226.09
hanoi-5_ext	6808	6778	0	0	0	0	0	<0.01

make a finer comparison, we check, for each BTP-mergeable pair of values, whether neighbourhood substitution or virtual interchangeability may also hold. Table 1 gives these results for a selection of the considered instances. For a few instances, all the values removed by BTP-merging can also be deleted by neighbourhood substitution or virtual interchangeability. In most cases (e.g. for the instances `inithx-i-2-28` or `mulsol-i-4-31`), the removed values belong to domains of variables having a degree 0 or 1. At the opposite extreme, for some instances, such as `ehi-85-297-33_ext`, none of

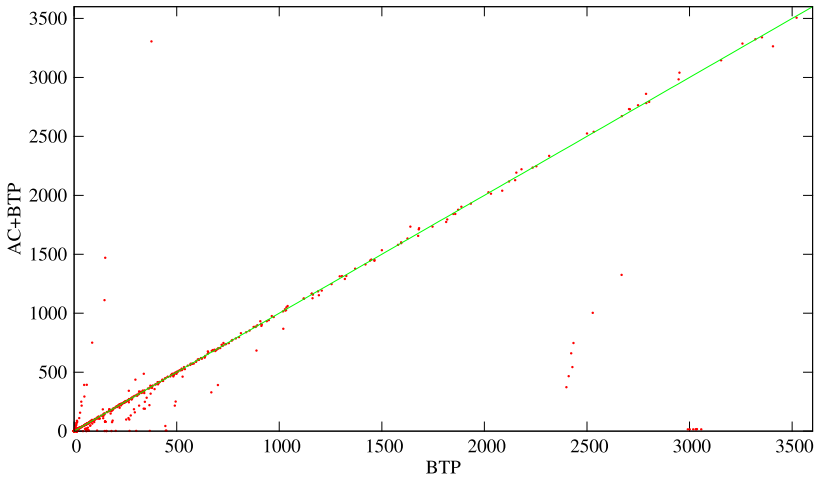

Fig. 12. Runtime of BTP-merging vs runtime of AC+BTP-merging for each considered instance.

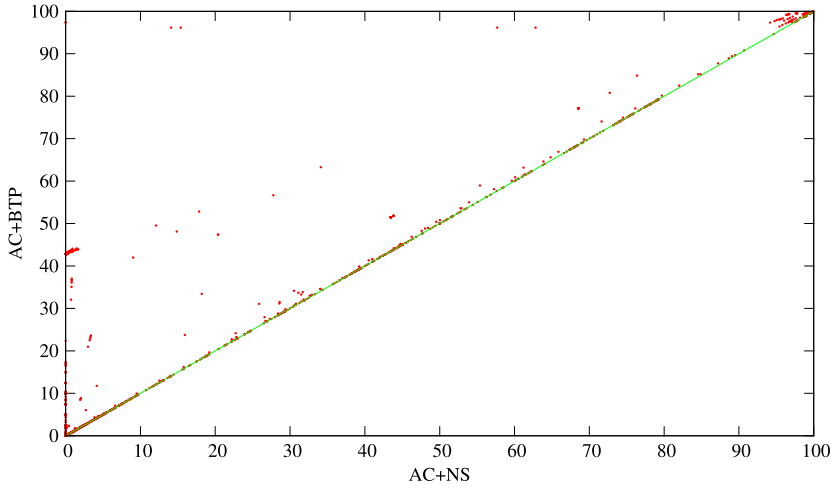

Fig. 13. Percentage of values removed by AC and neighbourhood substitution (AC+NS) vs percentage of values removed by AC and BTP-merging (AC+BTP) for each considered instance.

the values removed by BTP-merging can be removed by neighbourhood substitution or virtual interchangeability. For the majority of instances, BTP-merging removes some values which are removed neither by neighbourhood substitution nor by virtual interchangeability. We observe the same trends when the instances are preprocessed by AC (Table 2).

6.4. Impact on solving

In this subsection, we investigate the impact of removed values on the solving performed by MAC. For these experiments, we only consider those 828 instances which are arc-consistent and for which AC+BTP-merging removes at least one value. First, we observe that MAC with AC+BTP-merging solves 697 instances against 688 for MAC alone within the time-out of one hour. Note that the runtime of MAC with AC+BTP-merging includes the runtime of the solving and the AC+BTP-merging. Fig. 15 provides a comparison of the runtimes of MAC and MAC with AC+BTP-merging for the selected instances. Clearly, for most instances, MAC outperforms MAC with AC+BTP-merging with respect to runtime. This result is clearly due to the cost of achieving AC+BTP-merging which sometimes turns out to be too expensive with respect to the runtime of solving. However, in some cases, MAC with AC+BTP-merging is faster than MAC alone and, overall, is able to solve more instances.

In order to better assess the impact on solving, we now consider the number of nodes developed by MAC and MAC with AC+BTP-merging. We can see in Fig. 16 that solving by MAC with AC+BTP-merging turns out to be more efficient than we would have thought by just studying the total runtime. Indeed, thanks to the values removed by AC+BTP-merging, MAC with AC+BTP-merging is often able to develop less nodes than MAC alone. The total number of nodes developed by MAC with AC+BTP-merging is 27% less than the total number of nodes developed by MAC (32 millions compared to 44 millions). These preliminary results concerning solving are promising. However, in order to make MAC with AC+BTP-merging competitive, we have now to look for better algorithms for achieving AC+BTP-merging or techniques for identifying which instances could

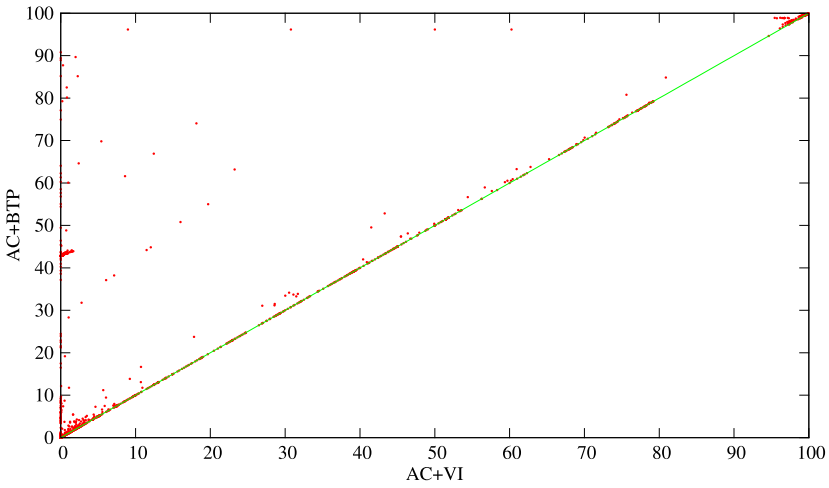

Fig. 14. Percentage of values removed by AC and virtual interchangeability (AC+VI) vs percentage of values removed by AC and BTP-merging (AC+BTP) for each considered instance.

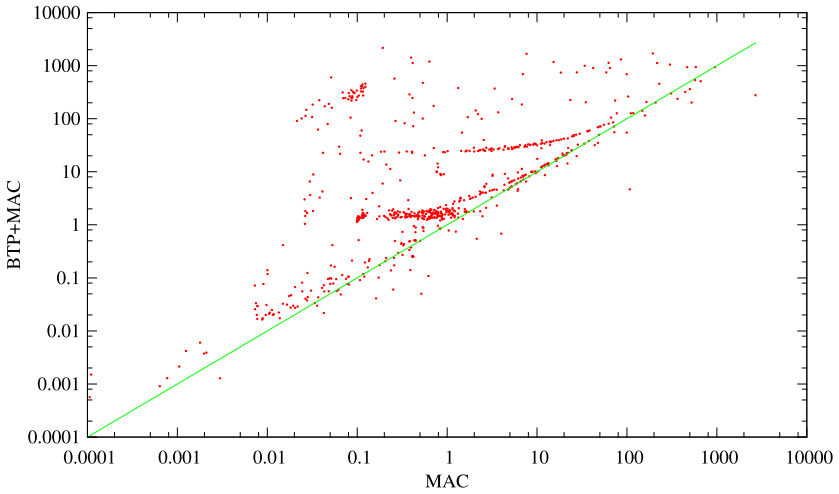

Fig. 15. Runtime of MAC vs runtime of MAC after BTP-merging. Runtimes are given in seconds.

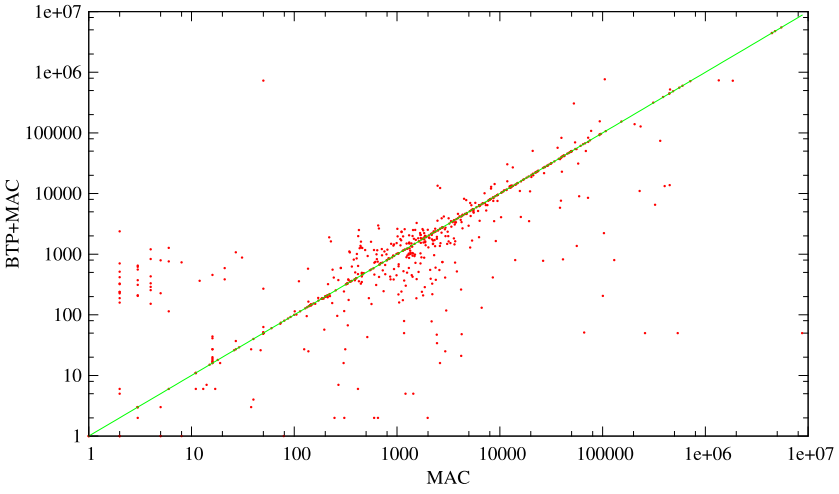

Fig. 16. Number of nodes developed by MAC vs number of nodes developed by MAC after BTP-merging.

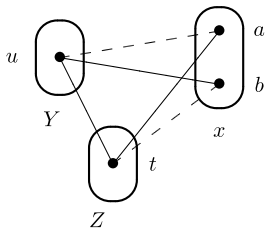

Fig. 17. A general-arity broken triangle on values $a, b \in \mathcal{D}(x)$.

best profit from BTP-merging during preprocessing. Note that the cost of searching for broken triangles precludes using BTP-merging during search.

An interesting phenomenon which is worthy of further investigation is that the number of nodes in the search tree may actually increase due to merging (since it tends to make constraints less tight) even though domain size has decreased. Likitvivatanavong and Yap [20] mention that search runtime may increase after merging virtual interchangeable values and indeed they observed that the number of instances for which search runtime increased was approximately the same as the number of instances in which search runtime decreased. An open theoretical question concerning the performance of MAC with or without BTP-merging is the existence of conditions under which BTP-merging is guaranteed not to increase the number of nodes in the search tree. Similarly, further experimental trials would be necessary to uncover relationships between the expected gain by BTP-merging and parameters such as average domain size, constraint density and constraint tightness.

7. Generalising BTP-merging to constraints of arbitrary arity

In the remainder of the paper, we assume that the constraints of a general-arity CSP instance I are given in the form described in Definition 2, i.e. as a set of incompatible tuples NoGoods(I), where a tuple is a set of variable-value assignments. For simplicity of presentation, we use the predicate Good(I, t) which is true iff the tuple t is a partial solution, i.e. t does not contain any pair of distinct assignments to the same variable and $\nexists t' \subseteq t$ such that $t' \in$ NoGoods(I). We first generalise the notion of broken triangle and merging to the general-arity case, before showing that absence of broken triangles allows merging.

Definition 11. A *general-arity broken triangle* (GABT) on values $a, b \in \mathcal{D}(x)$ consists of a pair of tuples t, u (containing no assignments to variable x) satisfying the following conditions:

1. Good($I, t \cup u$) $\wedge$ Good($I, t \cup \{\langle x, a\rangle\}$) $\wedge$ Good($I, u \cup \{\langle x, b\rangle\}$)
2. $t \cup \{\langle x, b\rangle\} \in$ NoGoods(I) $\wedge$ $u \cup \{\langle x, a\rangle\} \in$ NoGoods(I)

The pair of values $a, b \in \mathcal{D}(x)$ is *GABT-free* if there is no broken triangle on a, b.

A general-arity broken triangle is illustrated in Fig. 17. This figure is identical to Fig. 1 except that Y, Z are now sets of variables and t, u are tuples. Note that the sets Y and Z may overlap. As in the binary case, a dashed line represents a nogood (i.e. a tuple not in the constraint relation on its variables). A solid line now represents a partial solution.

If the constraints are represented by nogoods, as in our Definition 2, then to decide whether there is a GABT on a, b in a CSP instance, one can use the second condition in Definition 11 and explore all pairs $t \cup \{\langle x, b\rangle\}, u \cup \{\langle x, a\rangle\} \in$ NoGoods(I). On the other hand, if the constraints are represented as lists of allowed tuples, then one can use the first condition in Definition 11 and explore all pairs $t \cup \{\langle x, a\rangle\}, u \cup \{\langle x, b\rangle\}$ of tuples explicitly allowed by the constraints in I (since the second condition implies that under this representation, there is a constraint over the variables of t and x, and one over the variables of u and x). Whatever the representation, a pair t, u can be checked to be a GABT on a, b by evaluating the properties of Definition 11, all of which involve only constraint checks. Hence deciding whether a pair a, b is GABT-free is polytime for constraints given in extension (as the set of satisfying assignments) as well as for those given by nogoods (the set of assignments violating the constraint).

Definition 12. *Merging* values $a, b \in \mathcal{D}(x)$ in a general-arity CSP instance I consists of 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 $\langle x, b\rangle$, thus producing an instance I' with the new set of nogoods defined as follows:

$$\begin{aligned}\text{NoGoods}(I') = &\{t \in \text{NoGoods}(I) \mid \langle x, a\rangle, \langle x, b\rangle \notin t\}\\ &\cup \{t \cup \{\langle x, c\rangle\} \mid t \cup \{\langle x, a\rangle\} \in \text{NoGoods}(I) \ \wedge\\ &\quad \exists t' \in \text{NoGoods}(I) \text{ s.t. } t' \subseteq t \cup \{\langle x, b\rangle\}\}\end{aligned}$$

$$\cup \{t \cup \{\langle x, c\rangle\} \mid t \cup \{\langle x, b\rangle\} \in \text{NoGoods}(I) \ \wedge$$
$$\exists t' \in \text{NoGoods}(I) \text{ s.t. } t' \subseteq t \cup \{\langle x, a\rangle\}\}$$

A *value-merging condition* is a polytime-computable property $P(x, a, b)$ of assignments $\langle x, a\rangle$, $\langle x, b\rangle$ in a CSP instance I such that when $P(x, a, b)$ holds, the instance I' is satisfiable if and only if I is satisfiable.

This merging operation can be performed in polynomial time whether constraints are represented positively in extension or negatively as nogoods. For representations using nogoods this is clear from Definition 12. For representations in extension, simply observe that as in the binary case, the operation amounts to gathering together tuples which satisfy $\text{Good}(I, \cdot)$ and containing $\langle x, a\rangle$ or $\langle x, b\rangle$, and setting x to c in them.

Proposition 13. *In a general-arity CSP instance, being GABT-free is a value-merging condition. 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.*

Proof. In order to prove that satisfiability is preserved by this merging operation, it suffices to show that if s is a solution to I' containing $\langle x, c\rangle$, then either $s_a = (s \setminus \{\langle x, c\rangle\}) \cup \{\langle x, a\rangle\}$ or $s_b = (s \setminus \{\langle x, c\rangle\}) \cup \{\langle x, b\rangle\}$ is a solution to I. Suppose, for a contradiction that this is not the case. Then there are tuples $t, u \subseteq s \setminus \{\langle x, c\rangle\}$ such that $t \cup \{\langle x, b\rangle\} \in \text{NoGoods}(I)$ and $u \cup \{\langle x, a\rangle\} \in \text{NoGoods}(I)$. Since t, u are subsets of the solution s to I' and t, u contain no assignments to x, we have $\text{Good}(I, t \cup u)$. Since $t \cup \{\langle x, c\rangle\}$ is a subset of the solution s to I', we have $t \cup \{\langle x, c\rangle\} \notin \text{NoGoods}(I')$. By the definition of $\text{NoGoods}(I')$ given in Definition 12, and since $t \cup \{\langle x, b\rangle\} \in \text{NoGoods}(I)$, we know that $\nexists t' \in \text{NoGoods}(I)$ such that $t' \subseteq t \cup \{\langle x, a\rangle\}$. But then $\text{Good}(I, t \cup \{\langle x, a\rangle\})$. By a symmetric argument, we can deduce $\text{Good}(I, u \cup \{\langle x, b\rangle\})$. This provides the contradiction we were looking for, since we have shown that a general-arity broken triangle occurs on t, u, $\langle x, a\rangle$, $\langle x, b\rangle$.

Reconstructing a solution to the original instance can be achieved in linear time, since it suffices to verify which (or both) of s_a or s_b is a solution to I. □

7.1. Relationship with resolution in SAT

We now show that in the case of Boolean domains, there is a close relationship between merging two values a, b on which no GABT occurs and a common preprocessing operation used by SAT solvers. Given a propositional CNF formula φ in the form of a set of clauses (each clause C_i being represented as a set of literals) and a variable x occurring in φ, recall that *resolution* is the process of inferring the clause $(C_0 \cup C_1)$ from the two clauses $(\{\bar{x}\} \cup C_0), (\{x\} \cup C_1)$. Define the formula $Res(x, \varphi)$ to be the result of performing all such resolutions on φ, removing all clauses containing x or $\bar{x}$, and removing subsumed clauses:

$$Res(x, \varphi) = \min_{\subset}(\{C \mid C \in \varphi; x, \bar{x} \notin C\} \cup \{(C_0 \cup C_1) \mid (\{\bar{x}\} \cup C_0), (\{x\} \cup C_1) \in \varphi\})$$

It is a well-known fact that $Res(x, \varphi)$ is satisfiable if and only if φ is.

Eliminating variables in this manner from SAT instances, to get an equisatisfiable formula with less variables, is a common preprocessing step in SAT solving, and is typically performed provided it does not increase the size of the formula [27]. A particular case is when it amounts to simply removing all occurrences of x, which is the case, for instance, if x or $\bar{x}$ is unit or pure in φ, or if all resolutions on x yield a tautological clause.

Definition 14. A variable x is said to be *erasable* from a CNF φ if

$$Res(x, \varphi) \subseteq \{C \mid C \in \varphi; x, \bar{x} \notin C\} \ \cup \ \{C_0 \mid (\{\bar{x}\} \cup C_0) \in \varphi\} \ \cup \ \{C_1 \mid (\{x\} \cup C_1) \in \varphi\}$$

A CNF φ can be seen as the CSP instance I_φ on the set X of variables occurring in φ, with $\mathcal{D}(x) = \{\top, \bot\}$ for all $x \in X$, and $\text{NoGoods}(I_\varphi) = \{\overline{C} \mid C \in \varphi\}$, where $\overline{(\{x_1, \cdots x_p, \bar{x}_{p+1}, \cdots, \bar{x}_q\})} = \{\langle x_1, \bot\rangle, \ldots, \langle x_p, \bot\rangle, \langle x_{p+1}, \top\rangle, \ldots, \langle x_q, \top\rangle\}$.

Proposition 15. *Assume that no GABT occurs on values $\bot, \top$ for x in I_φ. Assume moreover that no clause in φ is subsumed by another one.*[2] *Then x is erasable from φ.*

Proof. Rephrasing Definition 11 (1) in terms of clauses, for any two clauses $(\{\bar{x}\} \cup C_0), (\{x\} \cup C_1) \in \varphi$ we have one of (i) $\exists C \in \varphi, C \subseteq (C_0 \cup C_1)$, (ii) $\exists C' \in \varphi, C' \subseteq (C_0 \cup \{x\})$, or (iii) $\exists C'' \in \varphi, C'' \subseteq (C_1 \cup \{\bar{x}\})$. Moreover, in Case (ii) C' must contain x, for otherwise the clause $(\{\bar{x}\} \cup C_0)$ would be subsumed in φ, contradicting our assumption. Similarly, in Case (iii) C'' must contain $\bar{x}$.

In Case (i) the resolvent $(C_0 \cup C_1)$ of $(\{\bar{x}\} \cup C_0), (\{x\} \cup C_1)$ is subsumed by C in $Res(x, \varphi)$, and hence does not occur in it. Similarly, in the second case $(C_0 \cup C_1)$ is subsumed by the resolvent of $(\{\bar{x}\} \cup C_0)$ and C', which is precisely C_0. The third

[2] This is without loss of generality since such clauses can be removed in polytime and such removal preserves logical equivalence.

case is dual. We finally have that the only resolvents added are of the form C_0 (resp. C_1) for some clause $(\{\bar{x}\} \cup C_0)$ (resp. $(\{x\} \cup C_1)$) of φ, as required. □

We can show the converse is also true provided that a very reasonable property holds.

Proposition 16. *Assume that φ satisfies: $\forall(\{x\} \cup C) \in \varphi, \nexists C' \subseteq C, (\{\bar{x}\} \cup C') \in \varphi$ and dually $\forall(\{\bar{x}\} \cup C) \in \varphi, \nexists C' \subseteq C, (\{x\} \cup C') \in \varphi$. If x is erasable from φ, then no GABT occurs on values $\bot, \top$ for x in I_φ.*

Proof. Assume for a contradiction that there is a GABT on values $\bot, \top$ for x in I_φ, let t, u be witnesses to this, and write $t \cup \{\langle x, \top\rangle\} = \overline{(\{\bar{x}\} \cup C_0)}$, $u \cup \{\langle x, \bot\rangle\} = \overline{(\{x\} \cup C_1)}$. Then the clause $(C_0 \cup C_1)$ is produced by resolution on x. Since x is erasable, $(C_0 \cup C_1)$ is equal to or subsumed by a clause $C \in Res(x, \varphi)$, where (applying Definition 14 in reverse) either C, or $(\{x\} \cup C)$, or $(\{\bar{x}\} \cup C)$ is in φ. The first case contradicts $\text{Good}(I_\varphi, t \cup u)$, so assume by symmetry $(\{x\} \cup C) \in \varphi$. From $C \notin \varphi$ and $C \in Res(x, \varphi)$ we get $\exists C' \subseteq C, (\{\bar{x}\} \cup C') \in \varphi$. But then the pair of clauses $(\{x\} \cup C), (\{\bar{x}\} \cup C') \in \varphi$ violates the assumption of the claim. □

8. BTP-merging in the presence of global constraints

Global constraints are an important feature of constraint programming. They not only facilitate modelling of complex problems but many global constraints also have dedicated efficient filtering algorithms [28]. In the presence of global constraints there are specific questions which need to be addressed to know whether BTP-merging is useful. The first thing to verify is that mergings are possible in the presence of one or more global constraints. A second important point is whether these BTP-mergings can be detected in polynomial time. A third point is to determine whether the semantics of the global constraint(s) are preserved by the operation of merging two values. For those global constraints that are decomposable into the conjunction of low-arity constraints, we can also ask whether BTP-merging applied to the decomposed version is equivalent to BTP-merging applied to the original global constraint(s). The answers to these questions depend on the global constraints. This section presents results concerning the important global constraint AllDifferent. These results are both negative and positive.

Proposition 17. *Determining whether two values can be GABTP-merged in a CSP instance consisting of two AllDifferent constraints is coNP-complete.*

Proof. It suffices to show that the problem of testing the existence of a general-arity broken triangle (GABT) in a CSP instance consisting of two AllDifferent constraints is NP-complete. We denote this problem by ∃GABT(2AllDiff). Clearly, the validity of a GABT can be checked in polynomial time. Testing the satisfiability of a CSP instance consisting of two AllDifferent constraints (a problem which we denote by CSP(2AllDiff)) is known to be NP-complete [29]. Thus to complete the proof it suffices to exhibit a polynomial reduction from CSP(2AllDiff) to ∃GABT(2AllDiff).

Let I be an instance, over variables X, consisting of two AllDifferent constraints with scopes S_1, S_2. Without loss of generality, we suppose that $S_1 \cup S_2 = X$. Let x, y, z be three variables not in X with domains containing only values not occurring in the domains of the variables in X, including $a, b \in \mathcal{D}(x)$ with $a \in \mathcal{D}(y)$, $a \notin \mathcal{D}(z)$, $b \in D(z)$, $b \notin D(z)$. We construct a new instance I' with variables $X \cup \{x, y, z\}$, with domains as in I for variables in X and the domains of variables x, y, z as just described. The instance I' has just two constraints: AllDifferent constraints with scopes $S_1 \cup \{y, x\}$ and $S_2 \cup \{z, x\}$. We will show that I' has a GABT on $a, b \in \mathcal{D}(x)$ if and only if I has a solution. A GABT on $a, b \in \mathcal{D}(x)$ consists of tuples t, u (containing no assignments to variable x) satisfying the following conditions: $\text{Good}(I', t \cup u)$, $\text{Good}(I', t \cup \{\langle x, a\rangle\})$, $\text{Good}(I', u \cup \{\langle x, b\rangle\})$, $t \cup \{\langle x, b\rangle\} \in \text{NoGoods}(I')$ and $u \cup \{\langle x, a\rangle\} \in \text{NoGoods}(I')$. Since $u \cup \{\langle x, a\rangle\} \in \text{NoGoods}(I')$, but $\text{Good}(I', u)$, we must have $\langle y, a\rangle \in u$, since y is the only variable other than x containing a in its domain. Similarly, we can deduce that $\langle z, b\rangle \in t$. Now $\text{Good}(I', t \cup u)$ implies that $(t \setminus \{\langle z, b\rangle\}) \cup (u \setminus \{\langle y, a\rangle\})$ is a solution to I. On the other hand, suppose that s is a solution to I. Let $u = s[S_1] \cup \{\langle y, a\rangle\}$ and $t = s[S_2] \cup \{\langle z, b\rangle\}$ (where $s[S]$ represents the subset of s corresponding to assignments to variables in S). Then the tuples t and u satisfy the conditions: $\text{Good}(I', t \cup u)$, $\text{Good}(I', t \cup \{\langle x, a\rangle\})$, $\text{Good}(I', u \cup \{\langle x, b\rangle\})$, $t \cup \{\langle x, b\rangle\} \in \text{NoGoods}(I')$ and $u \cup \{\langle x, a\rangle\} \in \text{NoGoods}(I')$. Thus t, u form a GABT on $a, b \in \mathcal{D}(x)$.

We have shown that I' has a GABT on $a, b \in \mathcal{D}(x)$ if and only if I has a solution. Since the reduction from CSP(2AllDiff) to ∃GABT(2AllDiff) is clearly polynomial, this completes the proof. □

Another problem with merging values in the presence of global constraints is that the global constraint may lose its semantics when values are merged. To give an example, consider an instance I in which a variable x (with domain $\mathcal{D}(x) = A$) occurs in the scope of a single constraint, an AllDifferent constraint on variables X. Since there is only one constraint on variable x, there can be no GABT on any pair of values in $\mathcal{D}(x)$. It is easy to see that we can, in fact, GABTP-merge all the values in $\mathcal{D}(x)$. When the domain of x becomes a singleton, we can clearly eliminate x. However, the resulting constraint on the variables $X \setminus \{x\}$ combines both an AllDifferent constraint on $X \setminus \{x\}$ and a constraint which says that the set of values assigned to these variables does not contain all of A. This constraint clearly does not have the same semantics as an

AllDifferent constraint. In general, merging values can transform global constraints which have efficient filtering algorithms into new global constraints which do not have efficient filtering algorithms.

After these negative results, we now give some positive results. It turns out that we can take advantage of the semantics of (global) constraints to reduce the complexity of searching for broken triangles. Suppose that instance I contains only AllDifferent constraints. Instead of looking for GABTP-merges, we can decompose the AllDifferent constraints into binary constraints and look for BTP-merges in the resulting instance I_{bin}. The presence of a general-arity broken triangle on $a, b \in \mathcal{D}(x)$ in I implies the presence of a broken triangle on $a, b \in \mathcal{D}(x)$ in I_{bin}, but the converse is not true. Thus BT-merging in I_{bin} is a strictly weaker operation than GABT-merging in I. The advantages of BT-merging in I_{bin} is that (1) it can be detected in linear time, and (2) it conserves the semantics of the AllDifferent constraints, as we will now show.

Lemma 18. *Suppose that instance I contains only binary difference constraints $x \neq y$. For each variable x, let S_x denote the set of variables constrained by x. Two distinct values a, b in the domain of a variable x can be BTP-merged if and only if one of the following conditions holds:*

1. *there is at most one variable $y \in S_x$ such that $\{a, b\} \cap D_y \neq \emptyset$*
2. *either $\forall y \in S_x$, $a \notin D_y$ or $\forall y \in S_x$, $b \notin D_y$.*

Proof. Since I contains only difference constraints, if y, z are two distinct variables in S_x, then the pair of assignments $\langle y, a\rangle$, $\langle z, b\rangle$ are necessarily compatible. Furthermore, from Definition 4, a broken triangle on $a, b \in \mathcal{D}(x)$ necessarily consists of assignments $\langle y, a\rangle$, $\langle z, b\rangle$ where x, y, z are distinct variables. Absence of a broken triangle on $a, b \in \mathcal{D}(x)$ is thus equivalent to there being at most one variable $y \in S_x$ such that $\{a, b\} \cap D_y \neq \emptyset$, or $\forall y \in S_x$, $a \notin D_y$ or $\forall y \in S_x$, $b \notin D_y$. □

Lemma 19. *Suppose that instance I contains only binary difference constraints and that $a, b \in \mathcal{D}(x)$ are BT-free. After BT-merging of $a, b \in \mathcal{D}(x)$, the variable x can be eliminated without the introduction of new constraints, producing an instance I' which is satisfiable if and only if I is satisfiable.*

Proof. If $y \neq x$, then $\forall d \in \mathcal{D}(y)$, $\langle y, d\rangle$ is either compatible with $\langle x, a\rangle$ or $\langle x, b\rangle$, since the only possible constraint between y and x is $y \neq x$. Hence, once $a, b \in \mathcal{D}(x)$ are merged, the resulting new value c is compatible with all assignments to all other variables. It follows immediately that x and all binary constraints with x in their scope can be eliminated while preserving the satisfiability of the instance. □

Proposition 20. *If I is an instance containing only binary difference constraints, then the result of applying BTP-merges (and eliminating the corresponding variables) until convergence is unique and can be found in $O(n^2d^2)$ time and $O(nd^2)$ space, where d is the maximum domain size.*

Proof. For each variable x and for each pair of distinct values $a, b \in \mathcal{D}(x)$, we can establish in $O(n)$ time three counters $N^x_{\{a\}}$, $N^x_{\{b\}}$, $N^x_{\{ab\}}$, where $N^x_A = |\{y \mid y \in S_x \ \wedge \ A \cap \mathcal{D}(y) \neq \emptyset\}|$.

By Lemma 18, to determine whether a, b can be BTP-merged, it suffices to check whether $N^x_{\{a,b\}} \leq 1$ or $N^x_{\{a\}} = 0$ or $N^x_{\{b\}} = 0$. After each BTP-merge, and the elimination of the corresponding variable, the constraints on the remaining variables remain unchanged. Thus, when a variable y is eliminated, due to the BT-merging of two values in its domain, for each variable $x \in S_y$: for each $a \in \mathcal{D}(y) \cap \mathcal{D}(x)$, we decrement the counter $N^x_{\{a\}}$ and for each pair $a, b \in \mathcal{D}(x)$ such that $a \in \mathcal{D}(y)$ or $b \in \mathcal{D}(y)$, we decrement the counter $N^x_{\{ab\}}$. Updating these data structures can be achieved in $O(nd^2)$ each time a variable y is eliminated. Since at most n variables can be eliminated, the total time complexity is $O(n^2d^2)$. The space complexity required to store the counters is $O(nd^2)$.

We now show that all maximal sequences of BTP-merges result in the same instance. For this we observe that if $a, b \in \mathcal{D}(x)$ can be BTP-merged in an instance I, and c, d can also be BTP-merged in I, then a, b can be BTP-merged in the instance I' obtained from I by BTP-merging $c, d \in \mathcal{D}(y)$. Indeed, by Lemma 19, the BTP-merge of $c, d \in \mathcal{D}(y)$ leads immediately to the elimination of the variable y, and clearly, such elimination cannot invalidate the characterisation of Lemma 18. By symmetry it also holds that c, d can be BTP-merged in the instance obtained from I by BTP-merging a, b, hence the order of BTP-merges does not matter. □

We have seen that applying the definition of GABT-merging to CSP instances containing AllDifferent constraints is coNP-complete and can also alter the semantics of the global constraints. However, Lemma 18 provides a weaker form of merging (which is equivalent to BT-merging if the instance contains only AllDifferent constraints that have been decomposed into an equivalent set of binary difference constraints) which can be applied in $O(n^2d^2)$ time. It is worth pointing out that this is much more efficient than a brute-force application of the definition of BT-merging in a binary CSP instance until convergence, which has worst-case time complexity $O(n^4d^5)$.

9. A tractable class of general-arity CSP

In binary CSP, the broken-triangle property defines an interesting tractable class when broken triangles are forbidden according to a given variable ordering. Unfortunately, this tractable class was limited to binary CSPs [7]. Section 7 described a general-arity version of the broken-triangle property whose absence on two values allows these values to be merged while preserving satisfiability. An obvious question is whether GABT-freeness can be adapted to define a tractable class. In this section we show that this is possible for a fixed variable ordering, but not if the ordering is unknown.

Definition 11 defined a general-arity broken triangle (GABT). What happens if we forbid GABTs according to a given variable ordering? Absence of GABTs on two values a, b for the last variable x in the variable ordering allows us to merge a and b while preserving satisfiability. It is possible to show that if GABTs are absent on all pairs of values for x, then we can merge all values in the domain $\mathcal{D}(x)$ of x to produce a singleton domain. This is because (as we will show later) merging a and b, to produce a merged value c, cannot introduce a GABT on c, d for any other value $d \in \mathcal{D}(x)$. Once the domain $\mathcal{D}(x)$ becomes a singleton $\{a\}$, we can clearly eliminate x from the instance, by deleting $\langle x, a\rangle$ from all nogoods, without changing its satisfiability. It is at this moment that GABTs may be introduced on other variables, meaning that forbidding GABTs according to a variable ordering does not define a tractable class.

Nevertheless, we will show that strengthening the general-arity BTP allows us to avoid this problem. The resulting directional general-arity version of BTP (for a known variable ordering) then defines a tractable class which includes the binary BTP tractable class as a special case.

Note that the set of general-arity CSP instances whose dual instance satisfies the BTP also defines a tractable class which can be recognised in polynomial time even if the ordering of the variables in the dual instance is unknown [16]. This DBTP class is incomparable with the class we present in the present paper (which is equivalent to BTP in binary CSP) since DBTP is known to be incomparable with the BTP class already in the special case of binary CSP [16]. A general-arity broken triangle can be said to be centred on a pair of *values* in the domain of a variable whereas a broken triangle in the dual instance is centred on a pair of *tuples* in a constraint relation. One consequence of this is that eliminating tuples from constraint relations cannot introduce broken triangles in the dual instance, whereas the (directional) GABTP is only invariant under elimination of domain values. On the other hand, the (directional) GABTP is invariant under adding a complete constraint (i.e. whose relation is the direct product of the domains of the variables in its scope) whereas this operation can introduce broken triangles in the dual instance. Another important difference is that directional GABTP depends on an order on the variables whereas DBTP depends on an order on the constraints.

9.1. Directional general-arity BTP

Recall that we assume that a CSP instance I is given in the form of a set of incompatible tuples NoGoods(I), where a tuple is a set of variable-value assignments, and that the predicate Good(I, t) is true iff the tuple t does not contain any pair of distinct assignments to the same variable and $\nexists t' \subseteq t$ such that $t' \in$ NoGoods(I). We suppose given a total ordering $<$ of the variables of a CSP instance I. We write $t^{<x}$ to represent the subset of the tuple t consisting of assignments to variables occurring before x in the order $<$, and $Vars(t)$ to denote the set of all variables assigned by t.

Definition 21. A *directional general-arity (DGA) broken triangle* on assignments a, b to variable x in a CSP instance I is a pair of tuples t, u (containing no assignments to variable x) satisfying the following conditions:

1. $t^{<x}$ and $u^{<x}$ are non-empty
2. Good($I, t^{<x} \cup u^{<x}$) $\wedge$ Good($I, t^{<x} \cup \{\langle x, a\rangle\}$) $\wedge$ Good($I, u^{<x} \cup \{\langle x, b\rangle\}$)
3. $\exists t'$ s.t. $Vars(t') = Vars(t)$ $\wedge$ $(t')^{<x} = t^{<x}$ $\wedge$ $t' \cup \{\langle x, a\rangle\} \notin$ NoGoods(I)
4. $\exists u'$ s.t. $Vars(u') = Vars(u)$ $\wedge$ $(u')^{<x} = u^{<x}$ $\wedge$ $u' \cup \{\langle x, b\rangle\} \notin$ NoGoods(I)
5. $t \cup \{\langle x, b\rangle\} \in$ NoGoods(I) $\wedge$ $u \cup \{\langle x, a\rangle\} \in$ NoGoods(I)

I satisfies the *directional general-arity broken-triangle property (DGABTP)* according to the variable ordering $<$ if no directional general-arity broken triangle occurs on any pair of values a, b for any variable x.

Points (1), (2) and (5) of Definition 21 are illustrated by Fig. 18. The two important differences compared to a general-arity broken triangle (Fig. 17) are that there is now a variable ordering $<$, with $y < x$ for all variables $y \in Y \cup Z$, and the two dashed lines now represent nogoods $u \cup \{\langle x, a\rangle\}$ and $t \cup \{\langle x, b\rangle\}$ which possibly involve assignments to variables $w > x$.

We will show that any instance I satisfying the DGABTP can be solved in polynomial time by repeatedly alternating the following two operations: (i) merge all values in the last remaining variable (according to the order $<$); (ii) eliminate this variable when its domain becomes a singleton. We will give the two operations (merging and variable-elimination) and show that both operations preserve satisfiability and that neither of them can introduce DGA broken triangles. Moreover, as for GABT-freeness, the DGABTP can be tested in polynomial time for a given order whether constraints are given as tables of satisfying assignments or as nogoods. Indeed, in the former case, using items (3) and (4) in Definition 21 we can restrict the search for a DGA broken triangle to pairs of tuples satisfying some constraint (there must be a constraint with scope

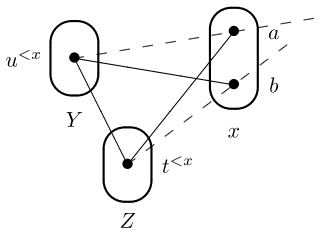

Fig. 18. Illustration of a directional general-arity broken triangle.

$Vars(t' \cup \{x\})$ since there is a nogood on these variables by item (5), and similarly for u'). This is sufficient to define a tractable class.

9.2. Merging

Let x be the last variable according to the variable order $<$. When values a, b in the domain of variable x do not belong to any DGA broken triangle, we can replace a, b by a new value c to produce an instance I' with the new set of nogoods given by Definition 12. Since x is the last variable in the ordering $<$, DGA broken triangles on $a, b \in \mathcal{D}(x)$ are GA broken triangles (and vice versa). Thus, from Proposition 13 we can deduce that satisfiability is preserved by this merging operation. What remains to be shown is that merging two values in the domain of the last variable cannot introduce the forbidden pattern.

Lemma 22. *Merging two values a, b into a value c in the domain of the last variable x (according to a DGABTP variable order $<$) in an instance I cannot introduce a directional general-arity broken triangle (DGABT) in the resulting instance I'.*

Proof. We first claim that this operation cannot introduce a DGABT on a variable $y < x$. Indeed, assume there is a DGABT on $d, e \in \mathcal{D}(y)$ in I', that is, that there are tuples v, w such that

1. $v^{<y}$ and $w^{<y}$ are non-empty
2. $\text{Good}(I', v^{<y} \cup w^{<y}) \wedge \text{Good}(I', v^{<y} \cup \{\langle y, d\rangle\}) \wedge \text{Good}(I', w^{<y} \cup \{\langle y, e\rangle\})$
3. $\exists v'\ Vars(v') = Vars(v) \ \wedge\ (v')^{<y} = v^{<y} \ \wedge\ v' \cup \{\langle y, d\rangle\} \notin \text{NoGoods}(I')$
4. $\exists w'\ Vars(w') = Vars(w) \ \wedge\ (w')^{<y} = w^{<y} \ \wedge\ w' \cup \{\langle y, e\rangle\} \notin \text{NoGoods}(I')$
5. $v \cup \{\langle y, e\rangle\} \in \text{NoGoods}(I') \ \wedge\ w \cup \{\langle y, d\rangle\} \in \text{NoGoods}(I')$

If v' contains the assignment $\langle x, c\rangle$ then, by construction of $\text{NoGoods}(I')$ (Definition 12), $\exists v'' \in \{(v' \setminus \langle x, c\rangle) \cup \{\langle x, a\rangle\}, (v' \setminus \langle x, c\rangle) \cup \{\langle x, b\rangle\}\}$ such that $v'' \cup \{\langle y, d\rangle\} \notin \text{NoGoods}(I)$. If v' does not contain $\langle x, c\rangle$ then let $v'' = v'$. Define w'' in a similar way. Now considering the last item, if v contains $\langle x, c\rangle$ then by construction of $\text{NoGoods}(I')$ there is v''' assigning a or b to x and otherwise equal to v, such that $v''' \cup \{\langle y, e\rangle\}$ was in $\text{NoGoods}(I)$, and if $v \not\ni \langle x, c\rangle$ we let $v''' = v$. We define w''' similarly. Then:

1. $(v''')^{<y} = v^{<y}$ and $(w''')^{<y} = w^{<y}$ are non-empty
2. $\text{Good}(I, (v''')^{<y} \cup (w''')^{<y}) \ \wedge\ \text{Good}(I, (v''')^{<y} \cup \{\langle y, d\rangle\}) \ \wedge\ \text{Good}(I,\ (w''')^{<y} \cup \{\langle y, e\rangle\})$ (since x is the last variable, $(v''')^{<y} = v^{<y}$ and $(w''')^{<y} = w^{<y}$)
3. $Vars(v'') = Vars(v''') \ \wedge\ (v'')^{<y} = (v''')^{<y} \ \wedge\ v'' \cup \{\langle y, d\rangle\} \notin \text{NoGoods}(I)$
4. $Vars(w'') = Vars(w''') \ \wedge\ (w'')^{<y} = (w''')^{<y} \ \wedge\ w'' \cup \{\langle y, e\rangle\} \notin \text{NoGoods}(I))$
5. $v''' \cup \{\langle y, e\rangle\} \in \text{NoGoods}(I) \ \wedge\ w''' \cup \{\langle y, d\rangle\} \in \text{NoGoods}(I)$

that is, there was a DGABT on d, e in I, contradicting our assumption.

We now show that a broken triangle cannot be introduced on x. Observe that since x is the last variable, for all tuples t not containing an assignment to x, $t^{<x} = t$ holds. We use this tacitly in the rest of the proof. Suppose for a contradiction that I contained no DGABT, but that after merging $a, b \in D(x)$ in I to produce the instance I', in which a, b have been replaced by a new value c, we have a DGABT on c, d. Then there is a pair of non-empty tuples t, u (containing no assignments to variable x) satisfying in particular the following conditions:

(1) $\text{Good}(I', t \cup u)$
(2) $\text{Good}(I', t \cup \{\langle x, c\rangle\})$
(3) $\text{Good}(I', u \cup \{\langle x, d\rangle\})$
(4) $t \cup \{\langle x, d\rangle\} \in \text{NoGoods}(I')$
(5) $u \cup \{\langle x, c\rangle\} \in \text{NoGoods}(I')$

We show that there was a DGABT in I either on a, d, on b, d or on a, b.

Since merging only affects tuples containing $\langle x, a\rangle$ or $\langle x, b\rangle$, (1) implies that Good$(I, t \cup u)$ and hence Good$(I, t \cup u')$ for all $u' \subseteq u$. Similarly, (3) implies that Good$(I, u \cup \{\langle x, d\rangle\})$ and hence Good$(I, u' \cup \{\langle x, d\rangle\})$ for all $u' \subseteq u$. Similarly, (4) implies that $t \cup \{\langle x, d\rangle\} \in$ NoGoods(I).

There are three possible cases to consider:

(a) Good$(I, t \cup \{\langle x, a\rangle\})$,
(b) Good$(I, t \cup \{\langle x, b\rangle\})$,
(c) $\exists t_1, t_2 \subseteq t$ such that $t_1 \cup \{\langle x, a\rangle\}$, $t_2 \cup \{\langle x, b\rangle\} \in$ NoGoods(I).

Case (a): By Definition 12 of the creation of nogoods during merging, (5) implies that $\exists u' \subseteq u$ such that $u' \cup \{\langle x, a\rangle\} \in$ NoGoods(I). We know that u' is non-empty since $u' \cup \{\langle x, a\rangle\} \in$ NoGoods(I) but Good$(I, t \cup \{\langle x, a\rangle\})$ (and hence Good$(I, \{\langle x, a\rangle\})$). We have Good$(I, t \cup u')$, Good$(I, t \cup \{\langle x, a\rangle\})$ (and hence $t \cup \{\langle x, a\rangle\} \notin$ NoGoods(I)), Good$(I, u' \cup \{\langle x, d\rangle\})$ (and hence $u' \cup \{\langle x, d\rangle\} \notin$ NoGoods(I)), $t \cup \{\langle x, d\rangle\} \in$ NoGoods(I), $u' \cup \{\langle x, a\rangle\} \in$ NoGoods(I) and hence there was a DGABT on a, d in I.

Case (b): Symmetrically to case (a), there was a DGABT on b, d in I.

Case (c): We claim that Good$(I, t_1 \cup \{\langle x, b\rangle\})$. If not, then we would have $\exists t_3 \subseteq t_1$ such that $t_3 \cup \{\langle x, b\rangle\} \in$ NoGoods(I) which would imply $t_1 \cup \{\langle x, c\rangle\} \in$ NoGoods(I') which is impossible since, by (2) above, we have Good$(I', t \cup \{\langle x, c\rangle\})$. By a symmetrical argument, we can deduce Good$(I, t_2 \cup \{\langle x, a\rangle\})$. Since Good$(I, t \cup u)$ and $t_1, t_2 \subseteq t$, we have Good$(I, t_1 \cup t_2)$. Since $t_1 \cup \{\langle x, a\rangle\} \in$ NoGoods(I) and Good$(I, t_2 \cup \{\langle x, a\rangle\})$ (and hence Good$(I, \{\langle x, a\rangle\})$), we must have $t_1 \neq \emptyset$. By a symmetric argument, $t_2 \neq \emptyset$. We therefore have non-empty tuples t_1, t_2 such that Good$(I, t_1 \cup t_2)$, Good$(I, t_1 \cup \{\langle x, b\rangle\})$ (and hence $t_1 \cup \{\langle x, b\rangle\} \notin$ NoGoods(I)), Good$(I, t_2 \cup \{\langle x, a\rangle\})$ (and hence $t_2 \cup \{\langle x, a\rangle\} \notin$ NoGoods(I)), $t_1 \cup \{\langle x, a\rangle\} \in$ NoGoods(I), $t_2 \cup \{\langle x, b\rangle\} \in$ NoGoods(I) and hence we have a DGABT in I on a, b.

Since in each of the three possible cases, we produced a contradiction, this completes the proof. □

9.3. Tractability of DGABTP for a known variable ordering

We are now in a position to give a new tractable class of general-arity CSP instances based on the DGABTP.

Theorem 23. *A CSP instance I satisfying the DGABTP on a given variable ordering can be solved in polynomial time.*

Proof. Suppose that I satisfies the DGABTP for variable ordering $<$ and that x is the last variable according to this ordering. Lemma 22 tells us that DGA broken triangles cannot be introduced by merging all elements in $\mathcal{D}(x)$ to form a singleton domain $\{a\}$. At this point it may be that $\{\langle x, a\rangle\}$ is a nogood. In this case the instance is clearly unsatisfiable and the algorithm halts returning this result. If not then we simply delete $\langle x, a\rangle$ from all nogoods in which it occurs. This operation of variable elimination clearly preserves satisfiability. It is polynomial time to recursively apply this merging and variable elimination algorithm until a nogood corresponding to a singleton domain is discovered or until all variables have been eliminated (in which case I is satisfiable).

To complete the proof of correction of this algorithm, it only remains to show that elimination of the last variable x cannot introduce a DGA broken triangle on another variable y. For all tuples t, u and all values $c, d \in D(y)$, none of Good$(I, t^{<y} \cup u^{<y})$, Good$(I, t^{<y} \cup \{\langle y, c\rangle\})$ and Good$(I, u^{<y} \cup \{\langle y, d\rangle\})$ can become true due to the variable elimination operation described above. On the other hand it is possible that $t \cup \{\langle y, d\rangle\}$ or $u \cup \{\langle y, c\rangle\}$ becomes a nogood due to variable elimination. Without loss of generality, suppose that $t \cup \{\langle y, d\rangle\}$ becomes a nogood and that $t' \cup \{\langle y, d\rangle\}$ is not a nogood for some t' such that $Vars(t') = Vars(t)$ and $(t')^{<y} = t^{<y}$. Then by construction there was a nogood $t \cup \{\langle y, d\rangle\} \cup \{\langle x, a\rangle\}$ before the variable x (with singleton domain $\{a\}$) was eliminated, and $t' \cup \{\langle y, d\rangle\} \cup \{\langle x, a\rangle\}$ was not a nogood. But then there was a DGA broken triangle (given by tuples $t \cup \{\langle x, a\rangle\}$, u on values $c, d \in D(y)$) before elimination of x. This contradiction shows that variable elimination cannot introduce DGA broken triangles. □

9.4. Finding a DGABTP variable ordering is NP-hard

An important question is the tractability of the recognition problem of the class DGABTP when the variable order is not given, i.e. testing the existence of a variable ordering for which a given instance satisfies the DGABTP. In the case of binary CSP, this test can be performed in polynomial time [7]. Unfortunately, as the following theorem shows, the problem becomes NP-complete in the general-arity case.

When a DGABTP ordering exists, there is at least one variable x such that all pairs of values $a, b \in \mathcal{D}(x)$ are GABT-free. In fact there may be several such variables which are all candidates for being the last variable in the DGABTP ordering. For any such variable x, after merging all values in the domain $\mathcal{D}(x)$ so that it becomes a singleton $\{a\}$, we can eliminate x from the instance, by deleting $\langle x, a\rangle$ from all nogoods, without changing its satisfiability. It is at this moment that DGABTs may be introduced on other variables. In the binary case, we can eliminate all such variables without the risk of introducing broken triangles. This is because deleting $\langle x, a\rangle$ from a binary nogood, such as $\{\langle x, a\rangle, \langle y, b\rangle\}$, produces the unary nogood $\langle y, b\rangle$ corresponding to the elimination of b from $\mathcal{D}(y)$ and the DGABTP cannot be destroyed by such domain reductions. In

the general-arity case, on the other hand, we cannot use such a greedy algorithm since the elimination of such a variable x may destroy the DGABTP for the as-yet-unknown variable ordering $<$ if x is not the last variable according to $<$.

Theorem 24. *Testing the existence of a variable ordering for which a CSP instance satisfies the DGABTP is NP-complete (even if the arity of constraints is at most 5).*

Proof. The problem is in NP since verifying the DGABTP is polytime for a given order, so it suffices to give a polynomial-time reduction from the well-known NP-complete problem 3SAT. Let I_{3SAT} be an instance of 3SAT with variables $X_1, \ldots, X_N$ and clauses $C_1, \ldots, C_M$. We will create a CSP instance I_{CSP} which has a DGABTP variable-ordering if and only if I_{3SAT} is satisfiable. For each variable X_i of I_{3SAT}, we add two variables x_i, y_i to I_{CSP}. To complete the set of variables in I_{CSP}, we add three special variables v, w, z. We add constraints to I_{CSP} in such a way that each DGABTP ordering of its variables corresponds to a solution to I_{3SAT} (and vice versa). The role of the variable z is critical: a DGABTP ordering $>$ of the variables of I_{CSP} corresponds to a solution to I_{3SAT} in which $X_i = \texttt{true} \Leftrightarrow x_i > z$. The variables y_i are used to code $\overline{X_i}$: $y_i > z$ in a DGABTP ordering if and only if $X_i = \texttt{false}$ in the corresponding solution to I_{3SAT}. The variables v, w are necessary for our construction and will necessarily satisfy $v, w < z$ in a DGABTP ordering. Each clause $C = l_1 \vee l_2 \vee l_3$, where l_1, l_2, l_3 are literals in I_{3SAT}, is imposed in I_{CSP} by adding constraints which force one of $\overline{l_1}$, $\overline{l_2}$, $\overline{l_3}$ to be false. To give a concrete example, if $C = X_1 \vee X_2 \vee X_3$, then constraints are added to I_{CSP} to force $y_1 < z$ or $y_2 < z$ or $y_3 < z$ in a DGABTP ordering. If the clause C contains a negated variable $\overline{X_i}$ instead of X_i, it suffices to replace y_i by x_i.

We now give in detail the necessary gadgets in I_{CSP} to enforce each of the following properties in a DGABTP ordering:

1. $v, w < z$
2. $y_i < z \Leftrightarrow x_i > z$
3. $y_i < z$ or $y_j < z$ or $y_k < z$

We introduce broken triangles in order to impose these properties. However, it is important not to inadvertently introduce other broken triangles. This can be avoided by making all pairs of assignments $\langle x, a\rangle$, $\langle x', a'\rangle$ from two different gadgets incompatible (i.e. $\{\langle x, a\rangle, \langle x', a'\rangle\} \in \text{NoGoods}(I_{CSP})$). We also assume that two gadgets which use the same variable x use distinct domain values in $\mathcal{D}(x)$. To avoid creating a trivial instance in which the gadgets disappear after establishing arc consistency, we can also add extra values in each domain which are compatible with all variable-value assignments in the gadgets.

We give the details of the three types of gadget:

1. The gadget to force $v, w < z$ in a DGABTP ordering consists of values $a_0 \in \mathcal{D}(z)$, $b_0, b_1 \in \mathcal{D}(v)$, $c_0, c_1 \in \mathcal{D}(w)$ and three nogoods $\{\langle z, a_0\rangle, \langle v, b_0\rangle\}$, $\{\langle z, a_0\rangle, \langle w, c_0\rangle\}$, $\{\langle v, b_1\rangle, \langle w, c_1\rangle\}$. The only way to satisfy the DGABTP on this triple of variables is to have $v, w < z$ since there are broken triangles on variables v and w.
2. To force $y_i < z \Leftrightarrow x_i > z$ in a DGABTP ordering we use two gadgets, the first to force $y_i > z \vee x_i > z$ and the second to force $y_i < z \vee x_i < z$.
   The first gadget is a broken triangle consisting of values $a_1, a_2 \in \mathcal{D}(z)$, $d_0 \in \mathcal{D}(x_i)$, $e_0 \in \mathcal{D}(y_i)$ and two nogoods $\{\langle z, a_1\rangle, \langle x_i, d_0\rangle\}$, $\{\langle z, a_2\rangle, \langle y_i, e_0\rangle\}$. In a DGABTP ordering we must have $y_i > z \vee x_i > z$.
   The second gadget consists of values $a_3, a_4 \in \mathcal{D}(z)$, $b_2 \in \mathcal{D}(v)$, $c_2 \in \mathcal{D}(w)$, $d_1 \in \mathcal{D}(x_i)$, $e_1 \in \mathcal{D}(y_i)$ and four nogoods $\{\langle z, a_3\rangle, \langle v, b_2\rangle, \langle x_i, d_1\rangle\}$, $\{\langle z, a_4\rangle, \langle v, b_2\rangle, \langle x_i, d_1\rangle\}$, $\{\langle z, a_4\rangle, \langle w, c_2\rangle, \langle y_i, e_1\rangle\}$, $\{\langle z, a_3\rangle, \langle w, c_2\rangle, \langle y_i, e_1\rangle\}$. We assume that we have forced $v, w < z$ using the gadget described in point (1). The tuples $t = \{\langle v, b_2\rangle, \langle x_i, d_1\rangle\}$, $u = \{\langle w, c_2\rangle, \langle y_i, e_1\rangle\}$ then form a DGA broken triangle on assignments $a_3, a_4 \in \mathcal{D}(z)$ if $x_i, y_i > z$. If either $x_i < z$ or $y_i < z$ then there is no DGA broken triangle; for example, if $x_i < z$, then we no longer have $\text{Good}(I_{CSP},\ t^{<z} \cup \{\langle z, a_3\rangle\})$ since $t^{<z} \cup \{\langle z, a_3\rangle$ is precisely the nogood $\{\langle z, a_3\rangle, \langle v, b_2\rangle, \langle x_i, d_1\rangle\}$. Thus this gadget forces $y_i < z \vee x_i < z$ in a DGABTP ordering.
3. The gadget to force $y_i < z$ or $y_j < z$ or $y_k < z$ in a DGABTP ordering consists of values $a_5, a_6 \in \mathcal{D}(z)$, $b_3 \in \mathcal{D}(v)$, $c_3 \in \mathcal{D}(w)$, $e_2 \in \mathcal{D}(y_i)$, $e_3 \in \mathcal{D}(y_j)$, $e_4 \in \mathcal{D}(y_k)$ and five nogoods: $\{\langle z, a_6\rangle, \langle v, b_3\rangle, \langle y_i, e_2\rangle, \langle y_j, e_3\rangle, \langle y_k, e_4\rangle\}$, $\{\langle z, a_5\rangle, \langle w, c_3\rangle\}$, $\{\langle z, a_5\rangle, \langle y_i, e_2\rangle\}$, $\{\langle z, a_5\rangle, \langle y_j, e_3\rangle\}$, $\{\langle z, a_5\rangle, \langle y_k, e_4\rangle\}$. The tuples $t = \{\langle v, b_3\rangle, \langle y_i, e_2\rangle, \langle y_j, e_3\rangle, \langle y_k, e_4\rangle\}$, $u = \{\langle w, c_3\rangle\}$ form a DGA broken triangle on $a_5, a_6 \in \mathcal{D}(a)$ if $y_i, y_j, y_k > z$. If $y_i < z$ or $y_j < z$ or $y_k < z$, then there is no DGA broken triangle; for example, if $y_i < z$, then we no longer have $\text{Good}(I_{CSP},\ t^{<z} \cup \{\langle z, a_5\rangle\})$ since $\{\langle z, a_5\rangle, \langle y_i, e_2\rangle\}$ is a nogood. Thus this gadget forces $y_i < z$ or $y_j < z$ or $y_k < z$ in a DGABTP ordering.

The above gadgets allow us to code I_{3SAT} as the problem of testing the existence of a DGABTP ordering in the corresponding instance I_{CSP}. To complete the proof it suffices to observe that this reduction is clearly polynomial. □

Our proof of Theorem 24 used large domains. The question still remains whether it is possible to detect in polynomial time whether a DGABTP variable ordering exists in the case of domains of bounded size, and in particular in the important case of SAT.

10. Conclusion

This paper described a novel reduction operation for binary CSP, called BTP-merging, which is strictly stronger than neighbourhood substitution. Experimental trials have shown that in several benchmark-domains, applying BTP-merging until convergence can significantly reduce the total number of variable-value assignments. We gave a general-arity version of BTP-merging and demonstrated a theoretical link with resolution in SAT. From a theoretical point of view, we then went on to define a general-arity version of the tractable class defined by the broken-triangle property for a known variable ordering. Our investigation of the interaction of BTP-merging and AllDifferent constraints has shown that the semantics of binary difference constraints can allow us to speed up the search for BTP-merges. An interesting avenue of future research is to try to take advantage of the semantics of other types of constraints to speed up the search for BTP-merges.

Acknowledgements

This research was supported by ANR Project ANR-10-BLAN-0210. Martin Cooper was also supported by EPSRC grant EP/L021226/1.

References

[1] D.A. Cohen, P.G. Jeavons, The complexity of constraint languages, in: F. Rossi, P. van Beek, T. Walsh (Eds.), Handbook of Constraint Programming, Elsevier, 2006, pp. 245–280.
[2] N. Creignou, P.G. Kolaitis, H. Vollmer (Eds.), Complexity of Constraints – An Overview of Current Research Themes [Result of a Dagstuhl Seminar], Lect. Notes Comput. Sci., vol. 5250, Springer, 2008, http://dx.doi.org/10.1007/978-3-540-92800-3.
[3] M. Grohe, The complexity of homomorphism and constraint satisfaction problems seen from the other side, J. ACM 54 (1) (2007), http://doi.acm.org/10.1145/1206035.1206036.
[4] D. Marx, Tractable hypergraph properties for constraint satisfaction and conjunctive queries, J. ACM 60 (6) (2013) 42, http://doi.acm.org/10.1145/2535926.
[5] P. Jégou, Decomposition of domains based on the micro-structure of finite constraint-satisfaction problems, in: R. Fikes, W.G. Lehnert (Eds.), Proceedings of the 11th National Conference on Artificial Intelligence, Washington, DC, USA, July 11–15, 1993, AAAI Press/The MIT Press, 1993, pp. 731–736, http://www.aaai.org/Library/AAAI/1993/aaai93-109.php.
[6] A.Z. Salamon, P.G. Jeavons, Perfect constraints are tractable, in: P.J. Stuckey (Ed.), Principles and Practice of Constraint Programming, Proceedings of the 14th International Conference, CP 2008, Sydney, Australia, September 14–18, 2008, in: Lect. Notes Comput. Sci., vol. 5202, Springer, 2008, pp. 524–528, http://dx.doi.org/10.1007/978-3-540-85958-1_35.
[7] M.C. Cooper, P.G. Jeavons, A.Z. Salamon, Generalizing constraint satisfaction on trees: hybrid tractability and variable elimination, Artif. Intell. 174 (9–10) (2010) 570–584, http://dx.doi.org/10.1016/j.artint.2010.03.002.
[8] D.A. Cohen, M.C. Cooper, P. Creed, D. Marx, A.Z. Salamon, The tractability of CSP classes defined by forbidden patterns, J. Artif. Intell. Res. 45 (2012) 47–78, http://dx.doi.org/10.1613/jair.3651.
[9] M.C. Cooper, G. Escamocher, Characterising the complexity of constraint satisfaction problems defined by 2-constraint forbidden patterns, Discrete Appl. Math. 184 (2015) 89–113, http://dx.doi.org/10.1016/j.dam.2014.10.035.
[10] M.C. Cooper, S. Živný, Tractable triangles and cross-free convexity in discrete optimisation, J. Artif. Intell. Res. 44 (2012) 455–490, http://dx.doi.org/10.1613/jair.3598.
[11] D.A. Cohen, M.C. Cooper, G. Escamocher, S. Živný, Variable and value elimination in binary constraint satisfaction via forbidden patterns, J. Comput. Syst. Sci. 81 (7) (2015) 1127–1143, http://dx.doi.org/10.1016/j.jcss.2015.02.001.
[12] M.C. Cooper, Beyond consistency and substitutability, in: O'Sullivan [31], pp. 256–271, http://dx.doi.org/10.1007/978-3-319-10428-7_20.
[13] P. Jégou, C. Terrioux, The extendable-triple property: a new CSP tractable class beyond BTP, in: B. Bonet, S. Koenig (Eds.), Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, Austin, TX, USA, January 25–30, 2015, AAAI Press, 2015, pp. 3746–3754, http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9939.
[14] M.C. Cooper, P. Jégou, C. Terrioux, A microstructure-based family of tractable classes for CSPs, in: Pesant [30], pp. 74–88, http://dx.doi.org/10.1007/978-3-319-23219-5_6.
[15] W. Naanaa, Unifying and extending hybrid tractable classes of CSPs, J. Exp. Theor. Artif. Intell. 25 (4) (2013) 407–424, http://dx.doi.org/10.1080/0952813X.2012.721138.
[16] A.E. Mouelhi, P. Jégou, C. Terrioux, A hybrid tractable class for non-binary CSPs, Constraints 20 (4) (2015) 383–413, http://dx.doi.org/10.1007/s10601-015-9185-y.
[17] J. Gao, M. Yin, J. Zhou, Hybrid tractable classes of binary quantified constraint satisfaction problems, in: W. Burgard, D. Roth (Eds.), Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, CA, USA, August 7–11, 2011, AAAI Press, 2011, http://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/view/3507.
[18] M.C. Cooper, A. El Mouelhi, C. Terrioux, B. Zanuttini, On broken triangles, in: O'Sullivan [31], pp. 9–24, http://dx.doi.org/10.1007/978-3-319-10428-7_5.
[19] M.C. Cooper, A. Duchein, G. Escamocher, Broken triangles revisited, in: Pesant [30], pp. 58–73, http://dx.doi.org/10.1007/978-3-319-23219-5_5.
[20] C. Likitvivatanavong, R.H. Yap, Eliminating redundancy in CSPs through merging and subsumption of domain values, ACM SIGAPP Appl. Comput. Rev. 13 (2) (2013).
[21] E.C. Freuder, Eliminating interchangeable values in constraint satisfaction problems, in: T.L. Dean, K. McKeown (Eds.), Proceedings of the 9th National Conference on Artificial Intelligence, Anaheim, CA, USA, July 14–19, 1991, vol. 1, AAAI Press/MIT Press, 1991, pp. 227–233, http://www.aaai.org/Library/AAAI/1991/aaai91-036.php.
[22] M.C. Cooper, Fundamental properties of neighbourhood substitution in constraint satisfaction problems, Artif. Intell. 90 (1–2) (1997) 1–24, http://dx.doi.org/10.1016/S0004-3702(96)00018-5.
[23] C. Bessière, J. Régin, Refining the basic constraint propagation algorithm, in: B. Nebel (Ed.), Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4–10, 2001, Morgan Kaufmann, 2001, pp. 309–315.
[24] D. Sabin, E.C. Freuder, Contradicting conventional wisdom in constraint satisfaction, in: A. Borning (Ed.), Principles and Practice of Constraint Programming, Proceedings of the Second International Workshop, PPCP'94, Rosario, Orcas Island, Washington, USA, May 2–4, 1994, in: Lect. Notes Comput. Sci., vol. 874, Springer, 1994, pp. 10–20, http://dx.doi.org/10.1007/3-540-58601-6_86.

[25] F. Boussemart, F. Hemery, C. Lecoutre, L. Saïs, Boosting systematic search by weighting constraints, in: R.L. de Mántaras, L. Saitta (Eds.), Proceedings of the 16th European Conference on Artificial Intelligence, ECAI'2004, Including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22–27, 2004, IOS Press, 2004, pp. 146–150.
[26] A. El Mouelhi, P. Jégou, C. Terrioux, Hidden tractable classes: from theory to practice, in: 26th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2014, Limassol, Cyprus, November 10–12, 2014, IEEE Computer Society, 2014, pp. 437–445, http://dx.doi.org/10.1109/ICTAI.2014.73.
[27] N. Eén, A. Biere, Effective preprocessing in SAT through variable and clause elimination, in: F. Bacchus, T. Walsh (Eds.), Theory and Applications of Satisfiability Testing, Proceedings of the 8th International Conference, SAT 2005, St. Andrews, UK, June 19–23, 2005, in: Lect. Notes Comput. Sci., vol. 3569, Springer, 2005, pp. 61–75, http://dx.doi.org/10.1007/11499107_5.
[28] W.-J. van Hoeve, I. Katriel, Global constraints, in: F. Rossi, P. van Beek, T. Walsh (Eds.), Handbook of Constraint Programming, Elsevier, 2006, pp. 169–208.
[29] M. Kutz, K.M. Elbassioni, I. Katriel, M. Mahajan, Simultaneous matchings: hardness and approximation, J. Comput. Syst. Sci. 74 (5) (2008) 884–897, http://dx.doi.org/10.1016/j.jcss.2008.02.001.
[30] G. Pesant (Ed.), Principles and Practice of Constraint Programming – Proceedings of the 21st International Conference, CP 2015, Cork, Ireland, August 31–September 4, 2015, Lect. Notes Comput. Sci., vol. 9255, Springer, 2015, http://dx.doi.org/10.1007/978-3-319-23219-5.
[31] B. O'Sullivan (Ed.), Principles and Practice of Constraint Programming – Proceedings of the 20th International Conference, CP 2014, Lyon, France, September 8–12, 2014, Lect. Notes Comput. Sci., vol. 8656, Springer, 2014, http://dx.doi.org/10.1007/978-3-319-10428-7.