Discrete Mathematics and Theoretical Computer Science DMTCS vol. **22**:1, 2020, #10

New schemes for simplifying binary constraint satisfaction problems

Wady Naanaa

National Engineering School of Tunis,
University of Tunis El Manar, Le Belvédère 1002 Tunis, Tunisia

received 22*nd* *Mar. 2018*, *revised* 22*nd* *May 2019, 23rd Oct. 2019*, *accepted* 9*th* *Feb. 2020*.

Finding a solution to a Constraint Satisfaction Problem (CSP) is known to be an NP-hard task. This has motivated the multitude of works that have been devoted to developing techniques that simplify CSP instances before or during their resolution.

The present work proposes rigidly enforced schemes for simplifying binary CSPs that allow the narrowing of value domains, either via value merging or via value suppression. The proposed schemes can be viewed as parametrized generalizations of two widely studied CSP simplification techniques, namely, value merging and neighbourhood substitutability. Besides, we show that both schemes may be strengthened in order to allow variable elimination, which may result in more significant simplifications. This work contributes also to the theory of tractable CSPs by identifying new tractable classes of binary CSP.

Keywords: Constraint satisfaction problems, value merging, variable elimination, tractable CSP

1 Introduction

Constraint Satisfaction Problem (CSP) is a generic problem which is well suited to the encoding of many difficult combinatorial problems Dechter (2003); Rossi et al. (2006); Tsang (1993). A CSP is defined by a finite set of variables and a finite set of constraints over these variables. Every variable is associated with a finite domain containing the values that may be assigned to that variable. The role of the constraints is to specify the permissible combinations of values, i.e., those that can be simultaneously assigned to subsets of the variables. In the specific case where every constraint involves, at most, two variables, we obtain a *binary* CSP. A solution is an assignment of a value to every variable that satisfies all the constraints. Finding a solution to a CSP instance or proving that none exist is known to be an NP-hard task. So, in absence of a polynomial solution algorithm, CSP solvers are usually enhanced by polynomial-time filtering algorithms. These algorithms may introduce significant simplifications on the problems to be solved without changing their solution sets, and then are often viewed as part of the solution process. Filtering algorithms proceed by establishing a limited form of (local) consistency that may, in some cases, guarantee the consistency of the whole problem. Problems that can be solved by establishing limited local consistencies are thereby solvable by polynomial algorithms. If, in addition, these problems are recognizable in polynomial time then they are said to be *tractable*.

ISSN 1365–8050 © 2020 by the author(s) Distributed under a Creative Commons Attribution 4.0 International License

Tractable problems may be arranged in *tractable classes* based on the specific features that make them tractable. Thus, there are two main types of problem tractability: *structural* tractability and *relational* tractability. Structural tractability is obtained by restricting the *constraint hyper-graph*, whose vertices are the problem variables and hyper-edges are the constraint scopes, to have some specific feature. A survey on the tractability in constraint satisfaction problems can be found in Carbonnel and Cooper (2016). The class of problems whose underlying hyper-graphs have a bounded hyper-tree width is one of the best known structural tractable class Gottlob et al. (2000). In turn, relational tractability is obtained by restricting the allowable constraint relations. The class of CSP limited to *max-closed* relations is one of the already identified relational tractable classes Jeavons and Cooper (1995). More recent works studied a new kind of tractability, called *hybrid* tractability Cooper et al. (2010); Cooper and Zivny (2017); El Mouelhi et al. (2013); Naanaa (2013); van Beek and Dechter (1997); Zhang and Yap (2003, 2006). The properties used in expressing hybrid tractability are neither purely structural nor purely relational. This allowed to derive new tractable problems whose specificity cannot be captured by exclusively structural or relational properties.

Among the multitude of works on hybrid tractability, the one presented in Cooper et al. (2010) has initiated a lot of efforts and developments, all aiming to simplify CSPs. The idea is centred on a specific ordered pattern called *broken triangle*, which when forbidden from appearing in the deep structure of a binary CSP instance makes it solvable in polynomial time. The set of all binary CSP instances not allowing a broken triangle as substructure were gathered in a binary CSP class called BTP. Furthermore, the recognition of a BTP instance can also be done in polynomial time, which completes the proof that BTP is a tractable binary CSP class. Even more practical, it has been shown in Cohen et al. (2015) that binary CSP instances that are not in the BTP class may nevertheless be simplified via the elimination of variables that are not involved in broken triangles.

Soon many other works followed in an attempt to extend the BTP class. k-BTP, a parametrized version of BTP Cooper et al. (2015) and *Extendable-Triple Property* (ETP), which is based on a pattern that generalizes the broken triangle Jégou and Terrioux (2015), are two extensions of BTP. Both are less restrictive than BTP but, in compensation, they require a higher level of local consistency to guarantee tractability. Unfortunately, enforcing the required level of local consistency on a given binary CSP instance that satisfies the k-BTP or the ETP may lead to the loss of these properties. For example, the tractability of a binary CSP that satisfies the 3-BTP or ETP requires a local consistency level, called *strong path consistency*, from the outset. This combined condition could be as restrictive as the condition underlying the BTP class. In Naanaa (2013), the author introduced the notion of CSP with bounded (directional) rank, which has proven to have a link to BTP. Indeed, it has been shown in Cooper et al. (2015) that the notion of directional rank $k-1$ strictly generalizes k-BTP. Yet another super-class of BTP is the variant called weak BTP (WBTP) Naanaa (2016). Contrary to k-BTP and ETP, WBTP requires no pre-established local consistency. In turn, WBTP inspired the work presented in Cooper et al. (2016b) in which the authors proposed a parametrized version of WBTP referred to as m-wBTP.

A huge advance was made with BTP when the authors of Cooper et al. (2016a) discovered that this property can be used as a condition to enable the merging of values inside the domains of variables. The advantage of merging values is obvious since the resulting CSP instances may only have smaller value domains. Moreover, the opportunities of value merging based on BTP are rather frequent as it has been experimentally shown in Cooper et al. (2016b). Since then, CSP reduction based on forbidding patterns has continued to be developed and many patterns, other than the broken triangle, were discovered and used, especially for variable elimination Carbonnel et al. (2018); Cohen et al. (2015); El Mouelhi (2017).

CSP simplification may also be achieved via removing values, or combinations of values, which could eliminate some, but not all, solutions. Value removal was explored in many works, giving rise to notions like neighbourhood value interchangeability and substitutability Freuder (1991), conditional interchangeability and substitutability Zhang and Freuder (2004), and value removability Bordeaux et al. (2008). These notions have resulted in as many filtering algorithms, which were successfully applied as preprocessing steps.

The present paper proposes two schemes designed to make simplifications on binary CSP. The first scheme may be seen as an enhanced value merging scheme that generalizes the one based on the BTP Cooper et al. (2016a). The goal is always to narrow the domains of variables. From this contribution, we derived a strengthened condition that turned the value merging scheme into a mean to eliminate variables. We rigorously specify this condition and propose an algorithm that identify all variables that can be eliminated from any binary CSP instance. Furthermore, the resulting variable elimination scheme allowed the discovery of new hybrid tractable classes of binary CSP. Our second contribution consists in a powerful value removal scheme that may be viewed as a parametrized version of neighbourhood substitutability, a widely studied CSP reduction technique that was proposed in Freuder (1991).

The paper is organized as follows: the next section recalls some definitions and notation of constraint satisfaction problems. Section 3 describes the first CSP simplification scheme, in which we proceed by value merging. In Section 4, we show that value merging may lead to variable elimination and even to a polynomial solution process, in favourable cases. Next, a new value removal scheme is detailed in Section 5. Finally, Section 6 is a brief conclusion.

2 Preliminaires

Let us begin by a formal definition of the constraint satisfaction problem:

Definition 1 *A constraint satisfaction problem (CSP) is defined by an ordered pair (X, C) where:*

- *X is a finite set of variables.*
- *C is a finite set of constraints. Every constraint is a pair (σ, R), where*
  - *σ is a sequence of variables providing the* scope *of the constraint and*
  - *R is a $|\sigma|$-ary* relation *containing the $|\sigma|$-tuples allowed by the constraint.*

The *arity* of a constraint is the size of its scope. The arity of a problem is the maximum arity over its constraints. A *binary* CSP is a CSP having arity two. We assume that there is, at most, one constraint, for a fixed scope. A constraint (σ, R) could therefore be indexed by its own scope and written as R_σ, for conciseness. A variable x must be assigned a value from its domain, which is the unary relation defining the unique unary constraint whose scope is limited to x, that is R_x. A *sub-domain* of a variable x is any subset of R_x. If a pair of variables x and y are not connected by a binary constraint in a binary CSP instance, one may assume that they are connected by the appropriate universal binary constraint, that is, the one defined by the complete binary relation $R_x \times R_y$. A *unary* assignment is an ordered pair $\langle x, v\rangle$ suggesting that variable x is assigned value v. A *partial assignment*, or simply *assignment*, is a set of unary assignments that cannot contain two unary assignments of the same variable. A *complete* assignment is a $|X|$-set of unary assignments, which assigns a value to every variable. An assignment A satisfies a unary constraint R_x if and only if v is in R_x whenever $\langle x, v\rangle$ is in A. Similarly, an assignment A satisfies a binary constraint $R_{x,y}$ if and only if (v, w) is in $R_{x,y}$ whenever both $\langle x, v\rangle$ and $\langle y, w\rangle$ are in A. An assignment A is *consistent* if and only if it satisfies all the constraints. A *solution* is a complete and consistent assignment. If a problem has, at least, one solution then it is said to be *consistent* otherwise it is *inconsistent*. In its most general version, the CSP is NP-hard, since there is no polynomial-time solution algorithm for general CSPs, unless P=NP. Nonetheless, CSP solving can be made faster by removing some value combinations whose removal has no effect on the solution set. Such removals achieve a limited form of consistency, called *local consistency*, which allows an easy calculation of consistent assignments of small sizes. *Arc consistency* is probably the most used local consistency level. In an arc consistent binary CSP instance, every value of every variable is compatible with, at least, one value of each of the other variables. More formally, $\forall x, y \in X$ and $\forall v \in R_x$, $\exists w \in R_y$ such that $(v, w) \in R_{x,y}$. Otherwise the non arc consistent instance can be efficiently transformed into an equivalent arc consistent one by simply removing not supported values Bessière et al. (2005).

The focus of this paper is to contribute to the improvement of binary CSP solving via new simplification schemes which will be detailed, in turn, in the following sections.

In the rest of the paper and for conciseness of notations, a union of the form $S \cup \{e\}$ will be often abbreviated to $S \cup e$. Also, the notation $\binom{S}{n}$ will be used to designate the set of all n-sized subsets of S.

3 Sub-domain merging

The first of the simplification schemes that we propose for binary CSP is based on value merging. The motivation is to reduce the size of value domains by replacing certain sub-domains by single values. The proposed value merging scheme is based on the following two definitions:

Definition 2 *Let D_x be a sub-domain of a variable x and let A be an assignment that does not affect a value to x. We say that D_x* supports *A if, for every $\langle y, w\rangle \in A$, there is $v \in D_x$ such that $(v, w) \in R_{x,y}$.*

In words, a sub-domain D_x supports an assignment A if and only if every unary assignment of A can be consistently extended to x via a value of D_x.

Definition 3 *Let (X, C) be a binary CSP instance and let $x \in X$. A sub-domain D_x is m-*mergeable*, for some integer m, $|D_x| \leq m < |X|$, if and only if, whenever D_x supports a m-sized consistent assignment A, there is $v \in D_x$ such that $A \cup \langle x, v\rangle$ is consistent.*

In words, a sub-domain D_x is m-mergeable, for some integer m, $|D_x| \leq m < |X|$, if and only if every m-sized consistent assignment supported by D_x can be consistently extended to x via a value of D_x. The integer m in the above definition will be referred to as the *merging parameter*. Note that every one-element sub-domain is trivially m-mergeable, for all $1 \leq m < |X|$. Conversely, for a fixed m, every sub-domain containing more than m values is not m-mergeable.

EXAMPLE 1. The graph depicted in Figure 1 is known as the micro-structure graph Jégou (1993). In such a graph, the vertices represent the values of the illustrated binary CSP instance. A pair of vertices are connected by an edge if and only if the associated values are mutually consistent.

The dashed ellipses are used to diagram domains and sub-domains. The values appearing inside a same dashed ellipse are assumed to be in the same domain (or sub-domain).

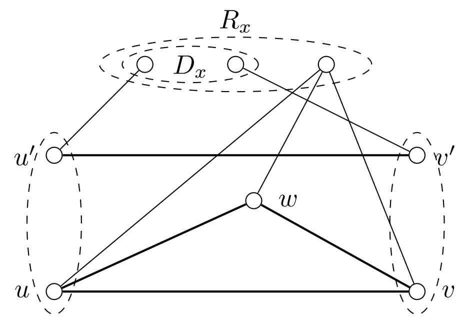

Fig. 1: R_x and D_x are 3-mergeable, but neither R_x nor D_x is 2-mergeable.

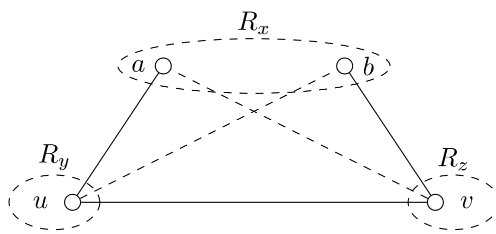

Fig. 2: A broken triangle (a, u, v, b) having a and b as end points. Inconsistent values are related by dashed edges.

R_x supports the 3-sized consistent assignment that can be built from values u, v and w (which will be designated by A_3) since each of these values is consistent with a value of R_x. Conversely, D_x does not support A_3. In accordance with Definition 3, both R_x and D_x are 3-mergeable. Indeed, A_3 can be consistently extended to x via one of the values of R_x (the one in the right of R_x). In turn, D_x is 3-mergeable because it does not support A_3. In contrast, D_x is not 2-mergeable because of the consistent assignment that can be formed from u' and v', which is supported by D_x but cannot be consistently extended via a value of D_x. Finally, R_x is trivially not 2-mergeable because of its size.

As it has been mentioned in the introduction, the authors of Cooper et al. (2016a) identified a pattern, called *broken triangle*, whose absence from a problem micro-structure allows value merging. A broken triangle involves four values $a, b \in R_x$, $u \in R_y$ and $v \in R_z$ such that $(a, u) \in R_{x,y}$, $(u, v) \in R_{y,z}$, $(v, b) \in R_{z,x}$, $(a, v) \notin R_{x,z}$ and $(b, u) \notin R_{x,y}$ (see Figure 2). In what follows, such a broken triangle will be designated by the quadruplet (a, u, v, b) and a and b will be designated by the *end points* of the broken triangle. It has been shown in Cooper et al. (2016a) that any two values, within a same domain, that are not the end points of a broken triangle can be merged while preserving problem consistency.

Two-mergeable value pairs correspond exactly to the value pairs that can be merged by means of BTP. This can be deduced by observing that the value pairs that form 2-mergrable sub-domains are those that are not the end points of a broken triangle. As a generalization of BTP-based value merging, the authors of Cooper et al. (2016b) proposed m-wBTP. This is a parametrized weak variant of BTP, which is defined as follows:

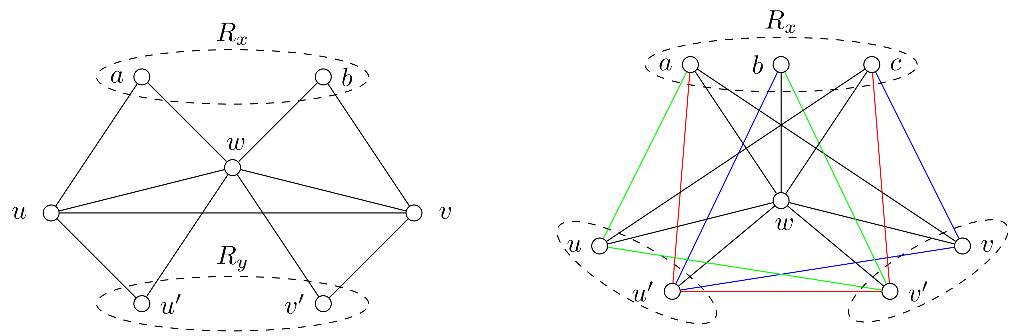

Fig. 3: (Left) The pair $\{a, b\}$ is not 3-mergeable, but it satisfies 1-wBTP. (Right) R_x is a 3-mergeable domain. However, each of the three value pairs that can be formed from R_x is not 1-wBTP.

Definition 4 *A pair of values $a, b \in R_x$ satisfies m-wBTP, where $m \leq |X| - 3$ if, for each broken triangle (a, u, v, b) with $u \in R_y$ and $v \in R_z$, there is a set of $r \leq m$ variables $Y \subseteq X \backslash \{x, y, z\}$ such that, for every assignment A that assigns values to all the variables of Y, if $A \cup \{\langle y, u \rangle, \langle z, v \rangle\}$ is consistent then there is $\langle t, w \rangle \in A$ such that $(w, a), (w, b) \notin R_{t,x}$.*

By inspecting the definition of m-mergeable sub-domains and that of m-wBTP, we notice that the former involves $m + 1$ variables, while the latter involves $m + 3$ variables. The notion of m-mergeable sub-domains should, therefore, be compared with $(m - 2)$-wBTP. However, m-mergeable sub-domains may be of any size not exceeding m, while m-wBTP allows merging pairs of values only, regardless of the value of m. In compensation, identifying a m-sized m-mergeable sub-domain requires $O(d^{m-2})$ times more steps than identifying a $(m - 2)$-wBTP value pair. Moreover, we can find non m-mergeable pairs of values that can be merged by $(m - 2)$-wBTP, as it will be shown in the following example.

EXAMPLE 2. In Figure 3-left, $R_x = \{a, b\}$ is not 3-mergeable, because of the 3-sized consistent assignment that can be formed from u, v and w, which supports R_x but cannot be consistently extended to variable x by any of the value of R_x. In contrast, due to variable y, R_x satisfies 1-wBTP although it is involved in a broken triangle: (a, u, v, b) (see Definition 4).

In Figure 3-right, $R_x = \{a, b, c\}$ is 3-mergeable since the three consistent assignments that can be formed from $\{u', v, w\}$, $\{u', v', w\}$ and $\{u, v', w\}$ can be consistently extended to x using a, b and c, respectively. In contrast, each of the three value pairs that can be formed from R_x does not satisfy 1-wBTP. For instance, the pair $\{a, c\}$, which is involved in the red broken triangle (a, u', v', c), cannot satisfy 1-wBTP via the variable that contains w in its domain. Similarly, the two other value pairs of R_x, which are involved in the blue and the green broken triangles, cannot satisfy 1-wBTP. We can also verify that the three value pairs that can be obtained from R_x are not 3-mergeable. For example, $\{a, c\}$ supports the consistent assignment that can be formed from $\{u', v', w\}$, but this latter assignment cannot be consistently extended by a or c. This shows that merging larger sub-domains could be advantageous.

Proposition 1 *In a binary CSP instance with variable set X, if a sub-domain is m-mergeable, for some integer m then it is m'-mergeable, for every m', $m \leq m' < |X|$.*

Proof: Suppose, for a sake of contradiction, that a sub-domain, say D_x, is m-mergeable but not m'-mergeable, for some m', $m < m' < |X|$. This implies that D_x supports a m'-sized consistent assignment A but A cannot be consistently extended to x by any value of D_x. Since A is consistent, we deduce that, for every $v \in D_x$, there exists $\langle y, \bar{v} \rangle \in A$ such that $(v, \bar{v}) \notin R_{x,y}$. Consider, therefore, $\bar{A}$ the subset of A composed by the $\langle y, \bar{v} \rangle$'s that satisfy this latter assertion, for the various values of D_x. Observe that $|\bar{A}| \leq |D_x| \leq m < m' = |A|$. It follows that $\bar{A} \subset A$ and then $\bar{A}$ can be completed by some unary assignments from A to obtain a m-sized consistent assignment $\bar{A}_m \subset A$. Note that D_x supports $\bar{A}_m$ since D_x supports A and $\bar{A}_m \subset A$. Moreover, $\bar{A}_m$ cannot be consistently extended to x by some value of D_x because $\bar{A}_m$ includes $\bar{A}$. This means that D_x is not m-mergeable and contradicts the hypothesis. $\Box$

The above proposition states that it is more appropriate to give priority to small values of m while searching for m-mergeable sub-domains. On the other hand, for a fixed m, a m-mergeable sub-domain may admit non m-mergeable sub-domains as proper subsets. Conversely, a sub-domain can be non m-mergeable while all its proper subsets are m-mergeable. These situations are illustrated in the following example.

EXAMPLE 3. The sub-domain D_x, of Figure 4-left, is not 2-mergeable because it supports the 2-sized consistent assignment that can be formed from u and u', but this latter assignment cannot be consistently extended by any value of D_x. D_x is, however, 3-mergeable since the only 3-sized consistent assignment, i.e. the one that can be formed from values v, v', u', is not supported by D_x.

In the CSP instance depicted in the middle of Figure 4, sub-domain D'_x and all its 2-sized subsets are not 3-mergeable. Indeed, D'_x supports the consistent assignment that can be formed from u, v and w, but this assignment cannot be consistently extended by any value of D'_x. The same situation occurs with every 2-sized subset of D'_x. This implies that D'_x and all its 2-sized subsets are not 3-mergeable. Also, all 2-sized subsets of D'_x are not 2-mergeable. For instance, the sub-domain composed of the two left-most values of D'_x does not consistently extend the assignment that can be formed from u and v. However, these two values support this assignment. The same holds for the two other 2-sized subsets of D'_x. This implies that all the 2-sized sub-domains are not 2-mergeable.

Finally, in Figure 4-right, D''_x is not 3-mergeable because of the 3-sized consistent assignment that can be formed from u, v and w. This latter assignment is supported by D''_x but cannot be consistently extended by any value of D''_x. In contrast, every 2-sized subset of D''_x is 3-mergeable, because it does not support the consistent assignment that can be formed from u, v and w. Unfortunately, merging any value pair of D''_x results in a non 3-mergeable value pairs.

We now focus on how to benefit from m-mergeable sub-domains to simplify CSPs. We first define a *m-unmergeable binary CSP instance* as being a binary CSP instance in which all sub-domains having size two or more are not m-mergeable. Clearly, in a m-unmergeable CSP instance, no sub-domains can be reduced in size by m-merging. On the other hand, any binary CSP instance can be transformed into a m-unmergeable CSP instance whose consistency is closely related to the consistency of the original instance. Such a m-unmergeable instance can be obtained by applying merging operations on m-mergeable sub-domains, until no non-singleton m-mergeable sub-domains are left. The advantage of the resulting m-unmergeable instance is that it would have smaller value domains than those of the original instance. It has been shown in Cooper et al. (2016a) that the size of the resulting instances depend on the order following which the merging operations were applied. Unfortunately, for $m = 2$, determining the "optimal"

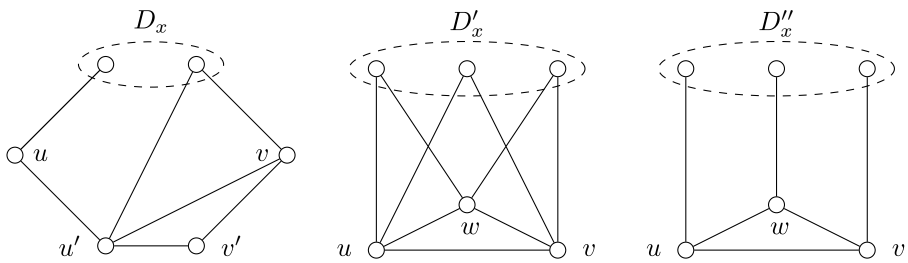

Fig. 4: Sub-domain D_x is 3-mergeable but not 2-mergeable. D'_x, as well as all its 2-sized subsets are 3-unmergeable. D''_x is 3-unmergeable but all its 2-sized subsets are 3-mergeable.

ordering, that is, the one that maximizes the number of merged values, is an NP-hard problem Cooper et al. (2016a), and the task should be more difficult for $m > 2$.

Below, we prove that every solution of any binary CSP instance reduced by merging operations can be transformed into a solution of the initial instance and *vice versa*. To show this, we begin by formally defining the CSP that results from a single merging operation. Note however that, in what follows, the values added by merging operations will be designated by lower-case bold-face letters.

Definition 5 *Let P be a binary CSP instance and let D_x be a sub-domain of a variable x. The merging of D_x results in the CSP instance P' obtained from P by only modifying the constraints containing x in their scopes as follows:*

- $R'_x = (R_x \backslash D_x) \ \cup \ \{\mathbf{v}\}$
- $R'_{x,y} = R_{x,y} \cup \{(\mathbf{v}, w) : \exists v \in D_x, \ (v, w) \in R_{x,y}\}$

where $\mathbf{v}$ is a new value.

A binary CSP instance P differs very little from its one-step merging reductions. Indeed, assume that the merging is performed on a sub-domain of a variable x. Then the constraints of P not having x in their scopes are identical to the corresponding constraints in any one-step merging reductions of P. More importantly, we show that if the merging operation described by Definition 5 is applied on a m-mergeable sub-domain then the consistency of the resulting CSP instance is closely related to that of the original instance.

Theorem 2 *Let P be a binary CSP instance and let P' be a CSP instance obtained from P by merging a m-mergeable sub-domain. Then P is consistent if and only if P' is consistent.*

Proof: Assume that the merging operation that allowed the transition from P to P' is a m-merging operation that was performed on a sub-domain of variable x. Thus, D_x will denote the sub-domain of x that contains the merged values and $\mathbf{v}$ will denote the value introduced in P' as suggested by Definition 5. Recall also that P and P' differ only with regard to the constraints having x in their scopes. As a consequence, any partial assignment, that does not affect a value to x, is consistent w.r.t. the constraints of P if and only if it is consistent w.r.t. the constraints of P'. This latter equivalence will be intensively used in the remainder of the proof.

$\Rightarrow$ Assume that $A \cup \langle x, v\rangle$ is a solution of P and proceed to deduce a solution for P'. Since A is a partial solution of P not assigning a value to x, it is also a partial solution of P'. To show that A can be consistently extended to form a solution of P', we distinguish two cases:

– $v \notin D_x$: According to Definition 5, this implies that $v \in R'_x$, which means that $A \cup \langle x, v\rangle$ satisfies the unique unary constraint of P' that has x as scope. So, let us turn to binary constraints. Unary assignment $\langle x, v\rangle$ is consistent, w.r.t. the constraints of P, with every unary assignment $\langle y, w\rangle \in A$ because $A \cup \langle x, v\rangle$ is a solution of P. It follows that $(v, w) \in R_{x,y}$, for all $\langle y, w\rangle \in A$, and since $v \notin D_x$, we obtain, by Definition 5, that $(v, w) \in R'_{x,y}$, for all $\langle y, w\rangle \in A$. This means that $\langle x, v\rangle$ is consistent, w.r.t. the constraints of P', with all the elements of A. It follows that $A \cup \langle x, v\rangle$ is also a solution of P'.

– $v \in D_x$: which means that v is one of the merged values. Let $A' = A \cup \langle x, \mathbf{v}\rangle$, where $\mathbf{v}$ is the new value introduced in P'. We prove that A' is a solution of P'. Note that A' trivially satisfies the unique unary constraint of P' on variable x because, according to Definition 5, $\mathbf{v}$ is in R'_x. Moreover, other than the value assigned to x, A' and A are the same and A is a partial solution of P'. It follows that A' satisfies all the binary constraints of P' not involving x. Consider therefore any binary constraint, $R'_{x,y}$, of P' that has x in its scope and show that A' satisfies such a constraint as well. Let w be the value assigned by A to y. We have therefore $\langle y, w\rangle \in A$. Note that $\langle y, w\rangle$ is also in A'. Since $A \cup \langle x, v\rangle$ is a solution of P, we must have $(v, w) \in R_{x,y}$. It follows from Definition 5 and $v \in D_x$ that $(\mathbf{v}, w) \in R'_{x,y}$, which implies that A' satisfies all the binary constraints of P' having x in their scopes. This completes the proof that A' satisfies all the constraints of P', which means that A' is a solution of P'.

$\Leftarrow$ Assume that $A' \cup \langle x, v'\rangle$ is a solution of P' and proceed to deduce a solution for P. We distinguish two cases:

– $v' = \mathbf{v}$: we prove that there exists $v \in D_x$ such that $A = A' \cup \langle x, v\rangle$ is a solution of P. Note that A' is already a partial solution of P since P and P' are identical with regard to the constraints not involving x in their scopes. For the sake of contradiction, suppose that there is no $v \in D_x$ such that $A = A' \cup \langle x, v\rangle$ is a solution of P. This implies that A violates a binary constraint involving x whatever the choice of $v \in D_x$. So, for every $v \in D_x$, there must exist $\langle y, w\rangle \in A'$ such that $(v, w) \notin R_{x,y}$. Let us denote by S' a minimal subset of A' such that, for every $v \in D_x$, there exists $\langle y, w\rangle \in S'$ and $(v, w) \notin R_{x,y}$. Note that S' is inconsistent with every value of D_x. Moreover, we have $|S'| \leq |D_x| \leq m$. On the other hand, $A' \cup \langle x, \mathbf{v}\rangle$ is a solution of P', which implies that $(\mathbf{v}, w) \in R'_{x,y}$, for all $\langle y, w\rangle \in A'$. By Definition 5, we deduce that, for every $\langle y, w\rangle \in A'$, there exists $v \in D_x$ such that $(v, w) \in R_{x,y}$. This means that D_x supports A' in P. Moreover, we have $m \leq |X| - 1 = |A'|$. It follows that D_x supports every m-subset M' of A'. All these M''s are consistent with regard to the constraints of P since A' is a partial solution of P. According to Definition 3, for every M', there must exist $v \in D_x$ such that $M' \cup \langle x, v\rangle$ is consistent with regard to the constraints of P. But among these M''s, there is necessarily one, say $\bar{M}'$, which includes S'. This is because the M''s are all the m-sized subsets of A' and S' is a subset of A' with m elements or less. The contradiction follows from the fact that $\bar{M}'$ is consistent with a value of D_x and, at the same time, it includes a subset, S', which is inconsistent with all the values of D_x.

– $v' \neq \mathbf{v}$: this implies that $v' \in R_x$, which means that $A' \cup \langle x, v'\rangle$ satisfies all the unary constraints of P. On the other hand, $A' \cup \langle x, v'\rangle$ satisfies all the binary constraints of P not involving x in their scopes. It remains to show that $A' \cup \langle x, v' \rangle$ satisfies also all the constraint of P that have x in their scopes. This amount to showing that $(v', w') \in R_{x,y}$, for all $\langle y, w' \rangle \in A'$. We start from the fact that $A' \cup \langle x, v' \rangle$ is a solution of P', from which we deduce that $(v', w') \in R'_{x,y}$, for all $\langle y, w' \rangle \in A'$. Then we use the second point of Definition 5, with $v' \neq \mathbf{v}$, to deduce that $(v', w') \in R_{x,y}$, for all $\langle y, w' \rangle \in A'$. Thus, $A' \cup \langle x, v' \rangle$ satisfies all the constraints of P, which means that it is a solution of P.

□

Theorem 2 expresses a tight consistency relationship between a binary CSP instance and its merging-based reductions. This relationship holds true regardless of the merging parameter value and the size of merged sub-domains. In fact, even by fixing the merging parameter to a specific value, merging operations can be attempted on sub-domains having various sizes. The choice of sub-domains that we would attempt to merge is a crucial issue for obtaining significant domain reductions. The authors of Cooper et al. (2016a), who proposed a merging scheme which corresponds to the specific case $m = 2$, showed that distinct 2-merging sequences may result in different reduced problems. Indeed, the order following which sub-domains are merged is deterministic for the problem that will be obtained at the end of the merging process.

A natural way to get around the choice of the sub-domains that we would attempt to merge could be achieved by limiting the merging parameter to small values. As a consequence and by referring to Definition 3, we deduce that only sub-domains whose size does not exceed the merging parameter could be merged. This limitation is motivated by the fact that the merging parameter has a huge influence on the computational complexity of the merging algorithm. Indeed, by referring to Definition 3, one can easily see that the time complexity of any optimal merging algorithm is bounded below by $\Omega(n^m d^m)$, where n is the number of variables, d is the size of the largest value domain and m is the merging parameter. Another issue that shall be considered concerns the merging strategy that will be followed in case where there are many mergeable sub-domains within the same domain. Such sub-domains may have different sizes and, most crucially, may overlap. This can complicate the merging process because it is not always possible to simultaneously merge all mergeable sub-domains.

We propose to apply merging operations by fixing the merging parameter, m, to a specific value, then the sub-domains are merged by increasing sizes. So, the size, s, of the sub-domains that the algorithm would attempt to merge varies from 2 to m. All s-sized and m-mergeable sub-domains are, therefore, identified, in turn, in every domain. Among these sub-domains, the merging algorithm selects a large subset whose members are pairwise disjoint. This large subset of disjoint sub-domains can be obtained, respectively approximated, by standard maximum set packing algorithms, respectively, maximum set packing heuristics. Note that all disjoint sub-domains can be merged simultaneously. The merging of the selected sub-domains can, therefore, take place in accordance with Definition 5.

The steps of the proposed sub-domain merging algorithm are detailed in Algorithm 1. The algorithm calls Boolean function Mergeable (see Algorithm 2), which determines whether a given sub-domain is m-mergeable or not. Looking at the pseudo-code of this function, we can see that its outer loop iterates $O(n^m d^m)$ times, which corresponds to the size of the m-sized consistent assignments set $\mathcal{A}^m$. The *support* test can be performed in $O(m^2)$, because $|A| = m$ and $|D_x| \leq m$. For the same reason, the inner loop also can be executed in $O(m^2)$ steps. So, the time complexity of function Mergeable is $O(m^2 n^m d^m)$.

The outer loop of the main algorithm (see Line 4 of Algorithm 1) can be repeated up to $O(nd)$ times, because in order for this loop to keep iterating, at least, one bi-valued sub-domain must be merged. The two nested for-loops can call function Mergeable up to $O(n^2 d^{m+1})$ times. On the other hand, the main algorithm needs to repeatedly calculate set packings (see Line 9). For $m = 2$, this can be achieved, in polynomial time, by any efficient maximum matching algorithm Gabow (1976). In contrast, for $m \geq 3$, the problem becomes NP-hard. An approximate solution can, however, be obtained in time linear in the input size by the algorithm proposed in Halldórsson et al. (2000), therefore, in $O(md^m)$. Since $m \leq n$, we deduce that the overall time complexity of the sub-domain merging algorithm is $O(m^2 n^{m+2} d^{2m+1})$.

Algorithm 1: MergeCSP(m, X, C)

```
1  𝒜 ← {⟨x, v⟩ : x ∈ X ∧ v ∈ R_x}
2  𝒜^m ← {A ∈ (𝒜 choose m) : A is consistent}
3  mrg ← true
4  while mrg do
5      mrg ← false
6      for s ← 2 to m do
7          for x ∈ X do
8              D_x ← {D_x ∈ (R_x choose s) : Mergeable(x, D_x, 𝒜^m, C)}
9              pck ← MaxSetPack(D_x)
10             for D_x ∈ pck do
11                 Merge(D_x, C)
12                 mrg ← true
```

Algorithm 2: Mergeable$(x, D_x, \mathcal{A}^m, C)$

```
for A ∈ 𝒜^m do
    if Supports(D_x, A, C) then
        mrg ← false
        for v ∈ D_x do
            if IsConsistent(x, v, A, C) then
                mrg ← true
                break
        if not mrg then return false
return true
```

4 Variable elimination

In this section, we address the issue of simplifying binary CSP instances by eliminating variables. We show that variables whose sub-domains of a fixed size are all mergeable can be eliminated with the guarantee that a solution of the initial instance can be polynomially deduced from any solution of the simplified

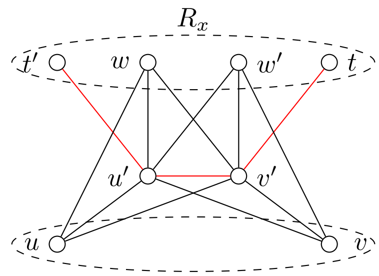

Fig. 5: A fragment of a binary CSP instance involving a 3-mergeable variable (x).

instance. However, prior to that, let us recall when could variables be eliminated as suggested in Cohen et al. (2015).

Definition 6 *A variable x can be* eliminated *from a binary CSP instance P having variable set X if, whenever there is a partial solution that assigns values to $X \backslash \{x\}$, there is a solution for P.*

In connection with the previous section and by an abuse of terminology, we define the notion of *m-mergeable variables*. Roughly speaking, a m-mergeable variable is a variable whose domain is comprised of m-mergeable m-sized sub-domains. More formally, m-mergeable variables are defined as follows:

Definition 7 *We say that a variable x is m-*mergeable*, for some integer m, $1 \leq m < |X|$, if every $\min(|R_x|, m)$-sized sub-domain of x is m-mergeable.*

Recall that the notion of m-mergeable sub-domain, to which it is referred in the above definition, is the one introduced in Definition 3.

EXAMPLE 4. The graph depicted in Figure 5 illustrates a fragment of a binary CSP instance which contains a 3-mergeable variable (x). To check this, we examine the four 3-sized sub-domain of x, in order to verify that all of them are 3-mergeable. The four sub-domains must be checked against the two 3-sized consistent assignments that can be respectively formed from u, u', v' and v, v', u', which will be referred to by A_u and A_v. We see from Figure 5 that A_u, resp. A_v, can be consistently extended to x using w, resp., w'. This implies that the two sub-domains $\{w, w', t\}$ and $\{w, w', t'\}$, are 3-mergeable. Moreover, $\{t, t', w\}$, which supports A_u but not A_v, consistently extends A_u via w. Similarly, $\{t, t', w'\}$, which supports A_v but not A_u, consistently extends A_v via w'. It follows that all the 3-sized sub-domains of R_x are 3-mergeable, which implies that x is a 3-mergeable variable. Note, however, that R_x contains a non 2-mergeable sub-domain, which is $\{t, t'\}$. Then x is not 2-mergeable.

We now focus on how to benefit from m-mergeable variables to simply CSPs. As a first step, we show that applying a single merging operation (see Definition 5) into the domain of a m-mergeable variable results in a CSP instance in which the processed variable remains m-mergeable.

Lemma 3 *Let x be a m-mergeable variable in a binary CSP instance. The merging of any* $\min(|R_x|, m)$-*sized sub-domain of x results in a CSP instance in which x remains m-mergeable.*

Proof: Let P be a binary CSP instance and let x be a m-mergeable variable of P. Denote by P' the instance resulting from merging a $\min(|R_x|, m)$-sized sub-domain of x, say D_x, and by $\mathbf{v}$ the value added to R'_x, the domain of x in P', as specified in Definition 5.

Suppose, for a sake of a contradiction, that x is not m-mergeable in P'. According to Definition 6, this implies that the domain of x in P', R'_x, contains a $\min(|R'_x|, m)$-sized sub-domain, say D'_x, which is not m-mergeable. According to Definition 3, D'_x must support a m-sized consistent assignment A' of P' while A' cannot be consistently extended to x by any value of D'_x. Thus, for every $v' \in D'_x$, there must exist $\langle y, \bar{v}' \rangle \in A'$ such that $(v', \bar{v}') \notin R'_{x,y}$. At this stage, we distinguish two cases, depending on the membership of $\mathbf{v}$ to D'_x:

- $\mathbf{v} \in D'_x$: in which case there must exist $\langle y, \bar{\mathbf{v}} \rangle \in A'$ such that $(\mathbf{v}, \bar{\mathbf{v}}) \notin R'_{x,y}$. From Definition 5, we deduce that $(v, \bar{\mathbf{v}}) \notin R_{x,y}$, for all $v \in D_x$. Let $D = D_x \cup D'_x \backslash \{\mathbf{v}\}$. Observe that $D \subseteq R_x$, which means that D is a sub-domain of x in P. We also deduce that A' cannot be consistently extended to x, by any of the values of D, to form a consistent assignment of P. Moreover, since D'_x supports A', any subset of A' which is not supported by $D'_x \backslash \{\mathbf{v}\}$ must be supported by $\{\mathbf{v}\}$. From Definition 5, we deduce that the subsets of A' that are not supported by $D'_x \backslash \{\mathbf{v}\}$ must be supported by D_x. It follows that D supports A'. On the other hand, we have $\min(|R_x|, m) \leq |D| \leq |R_x|$ since $D_x \subseteq D$, $|D_x| = \min(|R_x|, m)$ and $D \subseteq R_x$. And, since D supports A' and $|A'| = m$, there must exist $D' \subseteq D$, with $|D'| = \min(|R_x|, m)$ such that D' supports A'. Moreover, $D' \subseteq D$ implies that A' cannot be consistently extended to x, by any of the values of D', to form a consistent assignment of P. Thus, D' is not a m-mergeable $\min(|R_x|, m)$-sized sub-domain of x in P, which means that x is not m-mergeable in P and contradicts the hypothesis.

- $\mathbf{v} \notin D'_x$: which implies that $D'_x \subsetneq R'_x$ and then D'_x is a m-sized sub-domain of x in P. On the other hand, A' is a m-sized consistent assignment of P since P and P' differ only by the constraints involving x. In addition, because D'_x is a sub-domain of x in both P and P', the binary relations involving x are the same in P and P' if we limit the domain of x to D'_x. It follows that, D'_x supports A' in P as soon as it supports A' in P'. Moreover, A' cannot be consistently extended to x in P by a value of D'_x as soon as it cannot be consistently extended to x in P' by a value of D'_x. By Definition 3, this implies that D'_x, which has size m, is not m-mergeable in P. According to Definition 7, this means that x is not m-mergeable in P, which contradicts the hypothesis.

$\square$

It is clear that applying a merging operation on a sub-domain having size two or more narrows the domain of the processed variable. Moreover, according to Lemma 3, sub-domain merging can be applied many times inside the domain of a same variable when it comes to a m-mergeable variable. This would result in a variable with a one-element domain. Such a variable could be easily eliminated from the problem at hand. To show this, we begin by formally defining the CSP that results from the elimination of a single variable.

Definition 8 *Let P=(X, C) be a binary CSP instance and let x be in X. The elimination of x from P results in the CSP instance denoted by $P_{\backslash x}$=$(X_{\backslash x}, C_{\backslash x})$ and obtained from P as follows:*

- $X_{\backslash x} = X \backslash \{x\}$
- $C_{\backslash x} = \{S_\sigma : \exists R_\sigma \in C \ \wedge \ x \notin \sigma\}$, *with*

(i) $S_y = \{w \in R_y : \exists v \in R_x,\ (v,w) \in R_{x,y}\}$, *for all* $S_y \in C_{\backslash x}$.

(ii) $S_{y,z} = R_{y,z}$, *for all* $S_{y,z} \in C_{\backslash x}$.

Note that $P_{\backslash x}$ can be computed from P in $O(nd^2)$ steps. These steps are precisely needed to calculate the value domains of $P_{\backslash x}$.

Any single-variable elimination results in an instance that may differ from the original one at unary constraints level, as it can be seen from point (i) of Definition 8. In contrast, the binary constraints remaining after a single-variable elimination are unchanged, as it can be seen from point (ii) of Definition 8. More importantly, if a single-variable elimination is applied on a m-mergeable variable, for some integer $m \geq 2$, then the consistency of the resulting instance is closely related to that of the original instance. To show this, we proceed in two steps. First, we prove that sub-domain merging does not affect single variable elimination in the sense suggested by the following lemma.

Lemma 4 *Let P be a binary CSP instance and let P' be an instance obtained from P by merging a sub-domain of a variable x. Then, we have $P_{\backslash x} = P'_{\backslash x}$.*

Proof: Assume that the merging operation that allowed the transition from P=(X, C) to P'=(X, C') was performed on a sub-domain D_x of variable x. Denote by $\mathbf{v}$ the value introduced in P' as suggested by Definition 5. Recall also that P and P' differ only with regard to the constraints having x in their scopes.

According to Definition 8, $P_{\backslash x}$ and $P'_{\backslash x}$ have the same variable set, that is $X\backslash\{x\}$. Again, according to Definition 8, the binary constraints of $P_{\backslash x}$ and those of $P'_{\backslash x}$ are the same. It remains to check that the unary constraints, i.e. the domains, of $P_{\backslash x}$ and $P'_{\backslash x}$ are the same.

In what follows, we use R_σ and R'_σ to denote the constraints of P and P' respectively and we use S_σ and S'_σ to denote the constraints of $P_{\backslash x}$ and $P'_{\backslash x}$ respectively.

Suppose that $S_y \neq S'_y$, for some $y \in X\backslash\{x\}$, and proceed to get a contradiction. $S_y \neq S'_y$ implies that either $S_y\backslash S'_y \neq \varnothing$ or $S'_y\backslash S_y \neq \varnothing$. If $S_y\backslash S'_y \neq \varnothing$ then there exists $w \in S_y\backslash S'_y$. From Definition 8, we deduce the followings:
On the one hand, $w \in S_y$ implies that

$$\exists v \in R_x,\ (v,w) \in R_{x,y} \tag{1}$$

On the other hand $w \notin S'_y$ implies that

$$\forall v' \in R'_x,\ (v',w) \notin R'_{x,y} \tag{2}$$

From Definition 5 and (2), we obtain

$$(\forall v \in R_x\backslash D_x,\ (v,w) \notin R'_{x,y}) \quad \wedge \quad (\mathbf{v},w) \notin R'_{x,y}$$

Again according to Definition 5, this implies that

$$(\forall v \in R_x\backslash D_x,\ (v,w) \notin R_{x,y}) \quad \wedge \quad (\forall v \in D_x,\ (v,w) \notin R_{x,y})$$

This is equivalent to

$$\forall v \in R_x,\ (v,w) \notin R_{x,y}$$

which is in contradiction with (1).

Suppose now that there exists $w' \in S'_y \backslash S_y$. From Definition 8, we deduce the followings:
On the one hand, $w' \in S'_y$ implies that

$$\exists v' \in R'_x,\ (v', w') \in R'_{x,y} \tag{3}$$

On the other hand $w' \notin S_y$ implies that

$$\forall v \in R_x,\ (v, w') \notin R_{x,y} \tag{4}$$

From Definition 5 and (3), we obtain

$$(\exists v \in R_x \backslash D_x,\ (v, w') \in R'_{x,y}) \quad \vee \quad (\mathbf{v}, w') \in R'_{x,y}$$

Again according to Definition 5, this implies that

$$(\exists v \in R_x \backslash D_x,\ (v, w') \in R_{x,y}) \quad \vee \quad (\exists v \in D_x,\ (v, w') \in R_{x,y})$$

This is equivalent to

$$\exists v \in R_x,\ (v, w') \in R_{x,y}$$

which is in contradiction with (4).

We conclude that both $S_y \backslash S'_y$ and $S'_y \backslash S_y$ are empty, for all $y \in X \backslash \{x\}$, which means that $S_y = S'_y$, for all $y \in X \backslash \{x\}$. □

Next, we show that a variable with one-element domain can be easily eliminated.

Lemma 5 *Let P be a binary CSP instance and let x be a variable whose domain is a singleton. Then P is consistent if and only if $P_{\backslash x}$ is consistent.*

Proof: We prove the two senses in turn.
$\Rightarrow$ Assume that $A \cup \langle x, v \rangle$ is a solution of P and show that A is a solution of $P_{\backslash x}$. First, observe that A satisfies all the binary constraints of $P_{\backslash x}$, since these constraints are not changed while transforming P into $P_{\backslash x}$. It remains to check A against the unary constraints of $P_{\backslash x}$. Since $A \cup \langle x, v \rangle$ is a solution of P, we have $(v, w) \in R_{x,y}$, for all $\langle y, w \rangle \in A$, where $R_{x,y}$ denotes a binary constraint of P. According to Definition 8-(i), this implies that $w \in S_y$, for every $y \in X_{\backslash x}$, where S_y denotes the domain of y in $P_{\backslash x}$. This means that A satisfies all the unary constraints of $P_{\backslash x}$, and then A is a solution of $P_{\backslash x}$.
$\Leftarrow$ Let A be a solution of $P_{\backslash x}$. Let us show that A can be extended by a value of R_x, the domain of x in P, to form a solution of P. Since A is a solution of $P_{\backslash x}$, for every $\langle y, w \rangle \in A$, we have $w \in S_y$, where S_y denotes the domain of y in $P_{\backslash x}$. According to Definition 8, this implies that, for every $\langle y, w \rangle \in A$, there exits $v \in R_y$, such that $(v, w) \in R_{x,y}$, where R_y and $R_{x,y}$ denote constraints of P. But the domain of x in P is a singleton. It follows that, for all $\langle y, w \rangle \in A$, we have $(v, w) \in R_{x,y}$, where v is the unique value of x. This implies that $A \cup \langle x, v \rangle$ is a solution of P. □

Theorem 6 *Let P be a binary CSP instance and let x be a m-mergeable variable of P, for some $m \geq 2$. Then P is consistent if and only if $P_{\backslash x}$ is consistent.*

Proof: If the domain of x is empty then P is inconsistent. Moreover, in accordance with Definition 8, all the domains of $P_{\setminus x}$ will be empty, which means that $P_{\setminus x}$ is inconsistent.

Otherwise, according to Lemma 3, any $\min(|R_x|, m)$-sized sub-domain of x can be merged to obtain a reduced CSP instance in which x remains m-mergeable. Moreover, since $m \geq 2$, the merged domain will have a strictly smaller size than its initial size, unless the domain of x is already a singleton. Merging operations can be repeated until the domain of x becomes a singleton. Denote by P' the resulting instance. By Theorem 2, P is consistent if and only if P' is consistent. Consider therefore $P'_{\setminus x}$, the instance obtained from P' by eliminating variable x in accordance with Definition 8. According to Lemma 5, P is consistent if and only if $P'_{\setminus x}$ is consistent. On the other hand, by Lemma 4, we have $P'_{\setminus x} = P_{\setminus x}$. It follows that P is consistent if and only if $P_{\setminus x}$. □

The practical interest of Theorem 6 is reflected in a variable elimination algorithm (see Algorithm 3), which identifies and eliminates m-mergeable variables. The algorithm begins by computing the set of unary assignments, $\mathcal{A}$, and then, the set of m-sized consistent assignments, $\mathcal{A}^m$. As mentioned in the previous section, these two sets can respectively be computed in $O(nd)$ and $O(n^m d^m)$ steps. The core of the variable elimination algorithm is a Boolean function called *Eliminable*, which determines if a variable x is m-mergeable or not by simply testing the m-mergeability of all its $\min(|R_x|, m)$-sized sub-domains. By observing that variable suppression can only occur $O(n)$ times, we deduce that the call to function *Eliminable* can be performed $O(n^2)$ times. In turn, function *Eliminable* can perform $O(d^m)$ calls to function *Mergeable* (see Algorithm 2). The worst-case time complexity of this latter function has already been bounded by $O(m^2 n^m d^m)$ in Section 3. It follows that the overall time complexity of the variable elimination algorithm is $O(m^2 n^{m+2} d^{2m})$.

Next, we show that the property of being a m-mergeable variable is preserved by variable elimination, however under certain conditions related to domain size. To begin with, we prove that variable elimination preserves m-mergeable sub-domains.

Lemma 7 *Let P be a binary CSP instance containing a variable x. Then any sub-domain of any variable of $P_{\setminus x}$ is m-mergeable, with $m \leq |X| - 2$, whenever it is m-mergeable in P.*

Proof: We prove the contrapositive. Suppose that D_y, a sub-domain of a variable $y \neq x$, is not m-mergeable in $P_{\setminus x}$ and show that D_y is not m-mergeable in P. By Definition 3, if D_y is not m-mergeable in $P_{\setminus x}$ then there must exist a m-sized consistent assignment A of $P_{\setminus x}$ such that D_y supports A, but A cannot be extended by a value of D_y to form a consistent assignment of $P_{\setminus x}$. On the other hand, by Definition 8, every unary constraint of $P_{\setminus x}$ is a subset of the constraint having the same scope in P. Moreover, every binary constraint of $P_{\setminus x}$ is identical to the binary constraint having the same scope in P (see Definition 8-(ii)). It follows that D_y is a sub-domain of P, A is a m-sized consistent assignment of P and D_y supports A in P. This implies that A cannot be extended by a value of D_y to form a consistent assignment of P. We deduce that D_y is not m-mergeable in P, which proves the lemma. □

Theorem 8 *Let P be a binary CSP instance on variable set X and let x be a m-mergeable variable of P, with $m \leq |X| - 2$. Then any other m-mergeable variable of P is either m-mergeable in $P_{\setminus x}$ or its domain, in $P_{\setminus x}$, is reduced to less than m values.*

Proof: In the whole proof, the constraints of P and $P_{\setminus x}$ will be denoted by R_σ and S_σ, respectively.

Algorithm 3: ElimVariable(m, X, C)

$\mathcal{A} \leftarrow \{\langle x, v\rangle : x \in X \ \wedge \ v \in R_x\}$
$\mathcal{A}^m \leftarrow \{A \in \binom{\mathcal{A}}{m} : A$ is consistent$\}$
elim $\leftarrow$ `true`
while elim **and** $m < |X|$ **do**
    elim $\leftarrow$ `false`
    for $x \in X$ **do**
        if Eliminable$(x, m, \mathcal{A}^m, X, C)$ **then**
            $X \leftarrow X \backslash \{x\}$
            $C \leftarrow \{R_\sigma \in C : x \notin \sigma\}$
            for $y \in X$ **do**
                $R_y \leftarrow \{w \in R_y : \exists v \in R_x,\ (v, w) \in R_{x,y}\}$
            if EmptyDomain(C) **then**
                elim $\leftarrow$ `false`
                break
            `// suppressing the` m`-assignments that use removed values`
            for $A \in \mathcal{A}^m$ **do**
                for $\langle x, v\rangle \in A$ **do**
                    if $v \notin R_x$ **then**
                        $\mathcal{A}^m \leftarrow \mathcal{A}^m \backslash \{A\}$
                        break
            elim $\leftarrow$ `true`

Algorithm 4: Eliminable$(x, m, \mathcal{A}^m, C)$

if $|R_x| \leq m$ **then**
    return Mergeable$(x, R_x, \mathcal{A}^m, C)$
else
    for $D_x \in \binom{R_x}{m}$ **do**
        if not Mergeable$(x, D_x, \mathcal{A}^m, C)$ **then**
            return `false`
    return `true`

Let y be m-mergeable variable of P other than x. We prove that y remains m-mergeable in $P_{\setminus x}$ as long as its domain contains m values or more, i.e. $|S_y| \geq m$. First, observe that $|S_y| \geq m$ implies $|R_y| \geq m$. In accordance with Definition 7, we have to prove that every m-sized sub-domain of y is m-mergeable in $P_{\setminus x}$. We know that y is m-mergeable in P then, by Lemma 7, we deduce that every $\min(|R_y|, m)$-sized sub-domain of y is m-mergeable in $P_{\setminus x}$. It follows from $|R_y| \geq m$ that every m-sized sub-domain of y is m-mergeable in $P_{\setminus x}$. Moreover, we have $m \leq |X| - 2$ and then $m \leq |X_{\setminus x}| - 1$. According to Definition 7, this means that y is m-mergeable in $P_{\setminus x}$. □

In what follows we identify a new tractable binary CSP class based on 3-mergeable variables and another binary CSP class whose time complexity depends mainly of the merging parameter (m).

Theorem 9 *A binary CSP instance in which all the variables are* 3*-mergeable can be solved in* $O(n^2d^2 + \max(d^3, n^3))$*, where* n *is the number of variables and* d *is the size of the largest value domain.*

Proof: Let P be a binary CSP instance in which all the variables are 3-mergeable. According to Definition 7, P must contain more than three variables. Consider, therefore, the sequence of CSP instances defined as follows: Initially, we take $P^{(0)} = P$. The subsequent elements of the sequence are obtained by first checking whether $P^{(k)}$ admits a 3-mergeable variable or not. If yes, then a 3-mergeable variable, say x_k, is eliminated to obtain $P^{(k+1)} = P^{(k)}_{\setminus x_k}$. According to Theorem 6, $P^{(k)}$ is consistent if and only if $P^{(k+1)}$ is consistent. Otherwise, according to Theorem 8, we have two cases: either $P^{(k)}$ contains no more than three variables, or the variables of $P^{(k)}$ are all bi-valued[(i)]. In both cases, the variable elimination process ends with a residual instance which is easy to solve, as it will be explained below. Once we have obtained a solution for the residual instance, it is possible to deduce a solution for the original instance as suggested by Theorems 6.

We now turn to the time complexity of the overall solution process. This amounts to bounding the time cost of the following three stages: (1) The construction of the problem sequence described above, (2) Solving the residual instance, (3) Deducing a solution for the original instance. Constructing the problem sequence amounts to performing $O(n)$ single-variable eliminations. This can be done in $O(n^2d^2)$ steps, if we assume that the unmodified constraints are not duplicated when performing a single-variable elimination. If the residual instance is composed of no more than three variables then a solution can be obtained, by executing an exhaustive search, in $O(d^3)$ steps. Otherwise, the variables of the residual instance are all bi-valued, and then the instance can be solved, by establishing strong path consistency, in $O(n^3)$ steps Dechter (2003). Finally, extending a solution of the residual instance to a solution for the original instance can be done in $O(n^2d)$ steps. It follows that the overall time complexity of solving binary CSP with 3-mergeable variables only is $O(n^2d^2 + \max(d^3, n^3))$. □

Theorem 9 suggests that binary CSPs instances with only 3-mergeable variables can be solved in polynomial time. Moreover, such instances can be recognized in $O(n^4d^6)$ by checking every variable against the 3-mergeable property. Thus, binary CSPs with 3-mergeable variables is a tractable class of binary CSPs. This is a hybrid class, because the conditions that characterize 3-mergeable variables are neither purely structural nor purely relational. Unfortunately, the result obtained for binary CSPs involving 3-mergeable variables only does not hold for $m > 3$. The reason is that, for $m > 3$, the variable elimination process described in the proof of Theorem 9 may ends with an instance that contains $(m-1)$-valued variables and the CSP that includes such instances is known to be NP-complete. However, consider a case

[(i)] A bi-valued variable is a variable with no more than two possible values.

where the variable elimination process does not modify the domains of the initially m-mergeable variables. For example, this occurs with arc consistent binary CSP instances, whose m-mergeable variables can be eliminated in the sense of Definition 6.

Lemma 10 *Let P be an arc consistent binary CSP instance on variable set X and let x be a m-mergeable variable, with $m \leq |X| - 2$. Then any other m-mergeable variable of P remains m-mergeable in $P_{\setminus x}$.*

Proof: Let $y \neq x$ be a m-mergeable variable of P. This implies that every $\min(|R_y|, m)$-sized sub-domain of R_y is m-mergeable in P. On the other hand, thanks to arc consistency, we deduce from Definition 8-(i) that the domain of y in $P_{\setminus x}$ is the same as in P, that is R_y. It follows, by Lemma 7, that every $\min(|R_y|, m)$-sized sub-domain of R_y in $P_{\setminus x}$ is m-mergeable. The result follows from Definition 7. □

Theorem 11 *An arc consistent binary CSP instance on n m-mergeable variables can be solved in $O(m^2 d^m + n^2 d^2)$ steps, where d is the size of the largest value domain.*

Proof: We proceed as in the proof of Theorem 9. So, let P be a binary CSP instance on n m-mergeable variables. Note that, according to Definition 7, we must have $m < n$. Let us define the binary CSP instances $P^{(0)}, \ldots, P^{(n-m)}$ as follows:

$$P^{(0)} = P \qquad \text{and} \qquad P^{(k+1)} = P^{(k)}_{\setminus x_k}, \quad k : 0, \ldots, n-m-1 \tag{5}$$

where x_k is a m-mergeable variable of $P^{(k)}$.

First, we show that the arc consistency of $P^{(k)}$ entails that of $P^{(k+1)}$. Indeed, by Definition 8-(i), the domains of the variables of $P^{(k+1)}$ are the same as in $P^{(k)}$ when this latter is arc consistent. And since we started from an arc consistent instance, that is $P^{(0)}$, we deduce that all the instances defined by (5) are arc consistent. Then, by Lemma 10, $P^{(k)}$ admits a m-mergeable variable as long as it contains more that m variables. By observing that that $P^{(k)}$ contains $n - k$ variables, we deduce that every one of the instances $P^{(0)}, \ldots, P^{(n-m)}$ contains a m-mergeable variable. This means that these instances can be constructed according to (5). Moreover, according to Theorem 6, $P^{(k)}$ is consistent if and only if $P^{(k+1)}$ is consistent. The consistency of P is, therefore, equivalent to that of $P^{(n-m)}$. And since $P^{(n-m)}$ contains m variables, an $O(m^2 d^m)$ exhaustive search can be performed to get a solution for this instance. If such a solution exists then it can be extended to a solution for P, as suggested by Theorem 6, in $O(n^2 d^2)$ steps. Otherwise, P is inconsistent. □

EXAMPLE 5. We return to the CSP instance depicted in Figure 3-right. We have seen in Example 2 that R_x is 3-mergeable, while the three value pairs that can be formed from R_x are not 1-wBTP. In El Mouelhi (2017), the author defined m-fBTP, a more restrictive form of m-wBTP which has the advantage of allowing variable elimination. It has been proven that if a value pair satisfies m-fBTP then it satisfies m-wBTP (see Proposition 2 of El Mouelhi (2017)). This implies that the three value pairs of R_x are not 1-fBTP. We deduce that variable x is 3-mergeable but not 1-fBTP, which means that x can be eliminated via 3-merging but not via 1-fBTP.

5 Value removal

Another contribution of this paper consists in a powerful value removal scheme that can be viewed as a generalization of neighbourhood substitutability Freuder (1991). As for existing filtering schemes, the goal is to narrow the domains of the variables by suppressing values whose removal does not affect the consistency of the problem at hand.

Given a CSP instance P, let us first characterize the values whose removal preserves the consistency of P. To this end, we use the notation $P|x{\neq}v$ to designate the instance obtained from P by removing value v from the domain of variable x. First, recall the sufficient and necessary condition for removing values Bordeaux et al. (2004):

Definition 9 *We say that a value v can be* removed *from the domain of a variable x in a binary CSP instance P if, whenever there is a solution for P, there is a solution for $P|x{\neq}v$.*

Given a binary CSP instance P with variable set X, let us denote by $\mathcal{A}$ the set of all unary assignments that can be formed from the variables of P and their respective domains. We have therefore:

$$\mathcal{A} = \{\langle x, v\rangle : x \in X \ \wedge \ v \in R_x\} \tag{6}$$

For any integer r, $1 \leq r < |X|$, denote by $\mathcal{A}^r(x, v)$ the set of all r-sized consistent assignments of P that can be consistently extended to x with $v \in R_x$, that is

$$\mathcal{A}^r(x, v) = \{A \in \binom{\mathcal{A}}{r} : A \cup \langle x, v\rangle \text{ is consistent}\}$$

Note that, to be in $\mathcal{A}^r(x, v)$, an r-sized consistent assignment must not assign a value to x.

Definition 10 *Consider a binary CSP instance (X, C) and let $x \in X$. A value $v \in R_x$ is r-*removable*, for some integer r, $1 \leq r < |X|$, if there is a sub-domain $D_x \subseteq R_x \backslash \{v\}$, with $|D_x| \leq r$, such that $\mathcal{A}^r(x, v) \subseteq \bigcup_{w \in D_x} \mathcal{A}^r(x, w)$.*

The integer r intervening in the above definition will be referred to as the *removing parameter*. We can easily verify that 1-removable values correspond to neighbourhood substitutable values Freuder (1991). Indeed, if we adapt the definition of neighbourhood substitutability to our notation, we get the following: a value v of a variable x is neighbourhood substitutable to another value w of x if and only if $\mathcal{A}^1(x, v) \subseteq \mathcal{A}^1(x, w)$.

EXAMPLE 6. In the 3-variable binary CSP instance depicted in Figure 6, there are three 2-removable values, that are those represented by gray-filled vertices. The removal of these values results in the CSP instance depicted in the right-hand side of Figure 6. It is worth mentioning that, by inspecting every 2-sized sub-domain of the initial instance, we notice that there is none which is 2-mergeable. This illustrates a situation where the notion of r-removable values may simplify binary CSP instances that cannot be simplified by the notion of r-mergeable values.

The following theorem provides the main result of this section. It relates the notion of r-removable values introduced in Definition 10 to the values that can be removed from the domains of a binary CSP in accordance with Definition 9.

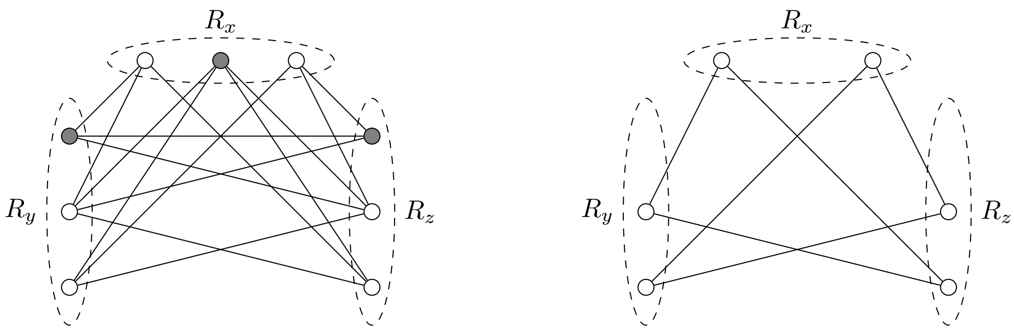

Fig. 6: A binary CSP instance including three 2-removable values, those represented by gray-filled vertices.

Theorem 12 *Let v be a value in the domain of a variable x in a binary CSP instance. If v is r-removable, for some integer r, then v can be removed from the domain of x.*

Proof: Assume that v is an r-removable value of variable x, which means that there exists a sub-domain $D_x \subseteq R_x \backslash \{v\}$, with $|D_x| \leq r$ such that

$$\mathcal{A}^r(x,v) \quad \subseteq \quad \bigcup_{w \in D_x} \mathcal{A}^r(x,w) \tag{7}$$

Then, let us show that P is consistent if and only if $P|x{\neq}v$ is consistent.

- $\Rightarrow$ Let $A \cup \langle x,u \rangle$ be a solution of P. It is clear that, if $u \neq v$ then $A \cup \langle x,u \rangle$ is also a solution of $P|x{\neq}v$. So, assume henceforth that $u = v$, which implies that $A \cup \langle x,u \rangle$ is not a solution of $P|x{\neq}v$. Let us show that A can be consistently extended to x by a value of D_x to form a solution of $P|x{\neq}v$. Suppose, for the sake of contradiction, that the converse is true. This implies that $A \cup \langle x,w \rangle$ is inconsistent, for every $w \in D_x$. But A is consistent, as well as every unary assignment $\langle x,w \rangle$, $w \in D_x$. It follows that, for every $w \in D_x$, there exists $\langle y,\bar{w} \rangle \in A$ which is not consistent with $\langle x,w \rangle$. Consider, therefore, $\bar{A}$ the subset of A that uses the values $\bar{w}$'s that satisfy the latter assertion. Observe that $|\bar{A}| \leq |D_x| \leq r$. In addition, since $\bar{A} \subseteq A$ and $r \leq |A| = |X| - 1$, we deduce that $\bar{A}$ can be completed by some elements of A to obtain a r-sized consistent assignment $\bar{A}^r \subseteq A$. Observe that
$$\bar{A}^r \notin \bigcup_{w \in D_x} \mathcal{A}^r(x,w) \tag{8}$$
because $\bar{A}^r$ includes $\bar{A}$ and $\bar{A}$ is inconsistent with $\langle x,w \rangle$, for every $w \in D_x$. On the other hand, $\bar{A}^r \subseteq A$ and $A \cup \langle x,v \rangle$ is consistent. This implies that $\bar{A}^r \cup \langle x,v \rangle$ is consistent, and then $\bar{A}^r \in \mathcal{A}^r(x,v)$. This is in contradiction with (7) and (8).

- $\Leftarrow$ From the definition of $P|x{\neq}v$, one can easily deduce that every solution of $P|x{\neq}v$ is also a solution of P.

$\square$

Based on the notion of r-removable values, we designed an algorithm that removes all the value that can be removed from the domains of binary CSP instances without affecting problem consistency. The steps of the proposed algorithm are detailed in Algorithm 5. This latter can be viewed as a r-parametrized version of the algorithm proposed in Bellicha et al. (1997), which can only eliminate 1-removable values. Our algorithm is based on the *separability* relationship, which was originally defined between pairs of simple values. In our case, the separability relationship need to be adapted in order to allow identifying r-removable values. Henceforth, the goal is to separate a single value from a r-sized sub-domain coming from the same domain.

Definition 11 *An assignment A* separates *a value $v \in R_x$ from a sub-domain $D_x \subseteq R_x \backslash \{v\}$ if $A \cup \langle x, v\rangle$ is consistent but not $A \cup \langle x, w\rangle$, for all $w \in D_x$.*

Bearing in mind Definition 10, it can be deduced from Definition 11 that if a value $v \in R_x$, does not admit a r-sized consistent assignment that separates it from an r-sized sub-domain of x then v is r-removable.

We propose a filtering algorithm whose worst case time complexity is $O(n^{r+1}d^{2r+1})$. To prove this complexity, we begin with a description of the data structures that we have used.

- $\mathcal{A}$ is an array containing all unary assignments that can be obtained from the variables and their respective domains as indicated in (6). Clearly, this array has a $O(nd)$ space complexity.
- $\mathcal{A}^r$ is the set of all r-sized consistent assignments that can be formed from the elements of $\mathcal{A}$. $\mathcal{A}^r$ can be implemented as a linked list whose space complexity is $O(n^r d^r)$.
- assgLst$[x, v]$ (for assignment list) is a linked list dedicated to the storage of every r-sized assignment $A \in \mathcal{A}^r$ such that $\langle x, v\rangle \in A$. We need $O(nd)$ assignment lists, one for every $\langle x, v\rangle \in \mathcal{A}$, each of which may contain up to $O(n^{r-1}d^{r-1})$ elements. Thus, the overall space complexity of the assignment lists is $O(n^r d^r)$.
- rmvLst is a linked list dedicated to the storage of the variable-value pairs that have been identified as r-removable. In the worst case, rmvLst may contain all variable-value pairs of the instance, that is, $O(nd)$ pairs.
- sepLst$[A]$: (for separation list) is a linked list that stores triples of the form (x, v, D_x), with $x \in X$, $v \in R_x$ and $D_x \subseteq R_x \backslash \{v\}$, such that A is the first element in list $\mathcal{A}^r$ that separates v from D_x. Note that there are $O(nd^{r+1})$ distinct triples. We need as many separation lists as there are elements in $\mathcal{A}^r$, that is, $O(n^r d^r)$. A crucial property of these lists is that they are pairwise disjoint. This implies that the total storage space required for all separation lists is $O(n^r d^r + nd^{r+1})$.

These data structures are used by Algorithm 5 as follows: Whenever a value v is removed from the domain of a variable x, the pair $\langle x, v\rangle$ is inserted in the removed value list, rmvList, in order to propagate the effect of this removal. Whenever a pair $\langle x, v\rangle$ is eliminated, every assignment containing that pair becomes inconsistent. The propagation begins, therefore, by checking whether the values separated by r-assignments containing $\langle x, v\rangle$ still have other separators. Observe that such values may become r-removable in cases where there is no r-assignment left that separates them from some r-sized sub-domains of the same variable. So, the nested loops beginning at Line 21 are executed in order to determine which values had actually become r-removable. If any, these values are removed from the domains to which they belong and are inserted in, turn, in the removed value list.

To prove the time-complexity mentioned above, we proceed to a careful examination of the steps of Algorithm 5. First of all, we assume that the removing parameter, that is, r is $O(1)$. Array $\mathcal{A}$ can be built in $O(nd)$ and list $\mathcal{A}^r$ in $O(n^r d^r)$. The first for loop of the algorithm can iterate $O(nd)$ times in order to initialize nd empty assignment lists. The nested loops beginning at Line 5 repeat the list insertion which is inside the two loops $O(n^r d^r)$ times, because $|A| = r$ and r is $O(1)$.

In order to evaluate the time complexity of the block composed by the three nested loops beginning at Line 11, we first calculate the time complexity of function GetSeparator (see Algorithm 6). Taking into account the size of $\mathcal{A}^r$ and the fact that $|D_x| = r = O(1)$, we deduce that GetSeparator runs in $O(n^r d^r)$ steps. Looking at the conditions of the three nested loops, we deduce that the whole block runs in $O(n^{r+1} d^{2r+1})$.

The time-complexity of the last block of Algorithm 5, that is the one beginning at Line 21, can be evaluated by focusing on the call to function GetSeparator (see Line 29). The three first parameters of the call are a variable $x \in X$, a value $v \in R_x$ and a r-sized subset of R_x. Note that, for these parameters, there are $O(nd^{r+1})$ different triples. The fourth parameter, $\mathcal{A}$, is the address of the cell of list $\mathcal{A}^r$ from which the current call to GetSeparator will start the search for a new separator. The use of cell addresses ensures that, for every triple (x, v, D_x), the cells of list $\mathcal{A}^r$ can only be examined once. Taking into account the size of $\mathcal{A}^r$, which is $O(n^r d^r)$, we deduce that $O(n^{r+1} d^{2r+1})$ steps are needed to process all the triples. By comparing the time complexity of the three blocks, we deduce that the overall algorithm runs in $O(n^{r+1} d^{2r+1})$.

6 Conclusion

This paper presented new schemes whose aim is to simplify constraint satisfaction problems. The proposed schemes proceed by merging many values at a time, i.e. sub-domains, or by suppressing values. The proposed contributions are parametrized versions of the value merging technique and the neighbourhood substitutability technique respectively proposed in Cooper et al. (2016a) and Freuder (1991). For this reason, our schemes can be viewed as generalizations of the two mentioned above. Moreover, we showed that the proposed variable elimination scheme allowed the identification of CSP instances than can be recognized and solved in polynomial time, thus giving rise to new hybrid tractable classes of binary CSPs.

References

L. Barto and M. Kozik. Constraint satisfaction problems solvable by local consistency methods. *J. ACM*, 61(1):3:1–3:19, 2014.

L. Barto, M. Kozik, and R. Willard. Near unanimity constraints have bounded pathwidth duality. In *LICS*, pages 125–134, 2012.

A. Bellicha, C. Capelle, M. Habib, T. Kôkény, and M.-C. Vilarem. Exploitation de la relation de substituabilité pour la réduction des CSP. *Revue d'Intelligence Artificielle*, 11(3):249–281, 1997.

C. Bessière, J.-C. Régin, R. H. C. Yap, and Y. Zhang. An optimal coarse-grained arc consistency algorithm. *Artif. Intell.*, 165(2):165–185, 2005.

Algorithm 5: REMOVEVALUEFROMCSP(X, C, r)

$\mathcal{A} \leftarrow \{\langle x, v\rangle : x \in X \ \wedge\ v \in R_x\}$
$\mathcal{A}^r \leftarrow \{A \in \binom{\mathcal{A}}{r} : A$ is consistent$\}$
`// Initializing lists`
for $\langle x, v\rangle \in \mathcal{A}$ **do** assgLst$[x, v] \leftarrow \varnothing$
for $A \in \mathcal{A}^r$ **do**
  sepLst$[A] \leftarrow \varnothing$
  for $\langle x, v\rangle \in A$ **do**
    `// Inserting the address of the cell of list` $\mathcal{A}^r$ `that contains` r`-assignment` A
    Insert(address(A), assgLst$[x, v]$)
rmvLst $\leftarrow \varnothing$
for $x \in X$ **do**
  for $D_x \in \binom{R_x}{r}$ **do**
    for $v \in R_x \backslash D_x$ **do**
      $A \leftarrow$ GetSeparator$(x, v, D_x, \mathcal{A}^r)$
      if $A =$ `nil` **then**
        $R_x \leftarrow R_x \backslash \{v\}$
        rmvLst $\leftarrow$ rmvLst $\cup\ \langle x, v\rangle$
      else
        sepLst$[A] \leftarrow$ sepLst$[A] \cup (x, v, D_x)$
while rmvLst $\neq \varnothing$ **do**
  $\langle y, w\rangle \leftarrow$ Extract(rmvLst)
  for $\mathcal{A} \in$ assgLst$[y, w]$ **do**
    $A \leftarrow$ ContentOf($\mathcal{A}$)
    while sepLst$[A] \neq \varnothing$ **do**
      $(x, v, D_x) \leftarrow$ Extract(sepLst(A))
      if $v \in R_x$ **and** $D_x \subseteq R_x$ **then**
        $A' \leftarrow$ GetSeparator$(x, v, D_x, \mathcal{A})$
        if $A' =$ `nil` **then**
          $R_x \leftarrow R_x \backslash \{v\}$
          rmvLst $\leftarrow$ rmvLst $\cup\ \langle x, v\rangle$
        else
          sepLst$[A'] \leftarrow$ sepLst$[A'] \cup (x, v, D_x)$

Algorithm 6: GetSeparator$(x, v, D_x, \mathcal{A})$

```
for A ∈ 𝒜 do
    if A ∪ ⟨x, v⟩ is consistent then
        sep ← true
        for w ∈ D_x do
            if A ∪ ⟨x, w⟩ is consistent then
                sep ← false
                break
        if sep then return A
return nil
```


C. Bessière, C. Carbonnel, E. Hebrard, G. Katsirelos, and T. Walsh. Detecting and exploiting subproblem tractability. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, 2013.

L. Bordeaux, M. Cadoli, and T. Mancini. Exploiting fixable, removable, and implied values in constraint satisfaction problems. In *Proceedings of the 11th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR), Montevideo, Uruguay, March 14-18*, pages 270–284, 2004.

L. Bordeaux, M. Cadoli, and T. Mancini. A unifying framework for structural properties of csps: Definitions, complexity, tractability. *J. Artif. Intell. Res.*, 32:607–629, 2008.

A. Bulatov. Mal'tsev constraints are tractable. *Electronic Colloquium on Computational Complexity (ECCC)*, (34), 2002.

A. Bulatov, P. G. Jeavons, and A. Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM J. Comput.*, 34:720–742, 2005.

C. Carbonnel and M. C. Cooper. Tractability in constraint satisfaction problems: a survey. *Constraints*, 21(2):115–144, 2016.

C. Carbonnel, D. A. Cohen, M. C. Cooper, and S. Zivny. On singleton arc consistency for csps defined by monotone patterns. In *the 35th Symposium on Theoretical Aspects of Computer Science, STACS, Caen, France, February 28 to March 3*, pages 19:1–19:15, 2018.

H. Chen, V. Dalmau, and B. Grußien. Arc consistency and friends. *J. Log. Comput.*, 23(1):87–108, 2013.

D. A. Cohen and P. G. Jeavons. The complexity of constraint languages. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, pages 245–280. Elsevier, 2006.

D. A. Cohen, M. C. Cooper, G. Escamocher, and S. Zivny. Variable elimination in binary CSP via forbidden patterns. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI), Beijing, China, August 3-9*, pages 517–523, 2013.

D. A. Cohen, M. C. Cooper, G. Escamocher, and S. Zivny. Variable and value elimination in binary constraint satisfaction via forbidden patterns. *J. Comput. Syst. Sci.*, 81(7):1127–1143, 2015.

M. C. Cooper. An optimal k-consistency algorithm. *Artif. Intell.*, 41(1):89–95, 1989.

M. C. Cooper and S. Zivny. Hybrid tractable classes of constraint problems. In *The Constraint Satisfaction Problem: Complexity and Approximability*, pages 113–135. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

M. C. Cooper, D. A. Cohen, and P. Jeavons. Characterising tractable constraints. *Artif. Intell.*, 65(2): 347–361, 1994.

M. C. Cooper, P. G. Jeavons, and A. Z. Salamon. Generalizing constraint satisfaction on trees: Hybrid tractability and variable elimination. *Artif. Intell.*, 174(9-10):570–584, 2010.

M. C. Cooper, A. E. Mouelhi, C. Terrioux, and B. Zanuttini. On broken triangles. In *Proceedings of the 20th Conference on Principles and Practice of Constraint Programming, Lyon, France, September 8-12*, pages 9–24, 2014.

M. C. Cooper, P. Jégou, and C. Terrioux. A microstructure-based family of tractable classes for CSPs. In *Proceedings of the 21st Conference on Principles and Practice of Constraint Programming, Cork, Ireland, August 31 - September 4*, pages 74–88, 2015.

M. C. Cooper, A. Duchein, A. El Mouelhi, G. Escamocher, C. Terrioux, and B. Zanuttini. Broken triangles: From value merging to a tractable class of general-arity constraint satisfaction problems. *Artif. Intell.*, 234:196–218, 2016a.

M. C. Cooper, A. El Mouelhi, and C. Terrioux. Extending broken triangles and enhanced value-merging. In *Proceedings of the 22nd Conference on Principles and Practice of Constraint Programming, Toulouse, France, September 5-9*, pages 173–188, 2016b.

V. Dalmau. Generalized majority-minority operations are tractable. *Logical Methods in Computer Science*, 2(4), 2006.

R. Dechter. *Constraint processing*. Elsevier Morgan Kaufmann, 2003.

R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, pages 353–366, 1989.

Y. Deville, O. Barette, and P. V. Hentenryck. Constraint satisfaction over connected row convex constraints. In *Artificial Intelligence*, pages 405–411, 1997.

Y. Deville, O. Barette, and P. V. Hentenryck. Constraint satisfaction over connected row convex constraints. *Artif. Intell.*, 109(1-2):243–271, 1999.

A. El Mouelhi. A BTP-based family of variable elimination rules for binary CSPs. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, San Francisco, California, USA., February 4-9*, pages 3871–3877, 2017.

A. El Mouelhi, P. Jégou, and C. Terrioux. A hybrid tractable class for non-binary CSPs. In *Proceedings of the 25th International Conference on Tools with Artificial Intelligence, Herndon, VA, USA, November 4-6*, pages 947–954, 2013.

T. Feder and M. Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory. *SIAM J. Comput.*, 28(1):57–104, 1998.

E. C. Freuder. A sufficient condition for backtrack-free search. *J. ACM*, 29(1):24–32, Jan. 1982. ISSN 0004-5411.

E. C. Freuder. A sufficient condition for backtrack-bounded search. *J. ACM*, 32(4):755–761, Oct. 1985. ISSN 0004-5411.

E. C. Freuder. Eliminating interchangeable values in constraint satisfaction problems. In *Proceedings of the 9th National Conference on Artificial Intelligence, Anaheim, CA, USA, July 14-19, Volume 1*, pages 227–233, 1991.

H. N. Gabow. An efficient implementation of edmonds' algorithm for maximum matching on graphs. *J. ACM*, 23(2):221–234, 1976.

G. Gottlob, N. Leone, and F. Scarcello. A comparison of structural CSP decomposition methods. *Artif. Intell.*, 124(2):243–282, 2000.

M. J. Green and D. A. Cohen. Domain permutation reduction for constraint satisfaction problems. *Artif. Intell.*, 172(8-9):1094–1118, 2008.

M. Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. ACM*, 54(1):1–24, March 2007.

M. M. Halldórsson, J. Kratochvíl, and J. A. Telle. Independent sets with domination constraints. *Discrete Applied Mathematics*, 99(1-3):39–54, 2000.

P. G. Jeavons and M. C. Cooper. Tractable constraints on ordered domains. *Artif. Intell.*, 79(2):327–339, 1995.

P. G. Jeavons, D. A. Cohen, and M. Gyssens. Closure properties of constraints. *J. ACM*, 44(4):527–548, 1997.

P. G. Jeavons, D. A. Cohen, and M. C. Cooper. Constraints, consistency and closure. *Artif. Intell.*, 101 (1-2):251–265, 1998.

P. Jégou. On the consistency of general constraint-satisfaction problems. In *Proceedings of the 11th National Conference on Artificial Intelligence. Washington, DC, USA, July 11-15*, pages 114–119, 1993. URL `http://www.aaai.org/Library/AAAI/1993/aaai93-018.php`.

P. Jégou and C. Terrioux. The extendable-triple property: A new CSP tractable class beyond BTP. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, Austin, Texas, USA., January 25-30*, pages 3746–3754, 2015.

E. W. Kiss and M. Valeriote. On tractability and congruence distributivity. *Logical Methods in Computer Science*, 3(2), 2007.

A. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.

M. Maróti. The existence of a near-unanimity term in a finite algebra is decidable. *J. Symb. Log.*, 74(3): 1001–1014, 2009.

R. Mohr and T. C. Henderson. Arc and path consistency revisited. *Artif. Intell.*, 28(2):225–233, 1986.

W. Naanaa. Unifying and extending hybrid tractable classes of CSPs. *J. Exp. Theor. Artif. Intell.*, 25(4): 407–424, 2013.

W. Naanaa. Extending the broken triangle property tractable class of binary CSPs. In *Proceedings of the 9^{th} Hellenic Conference on Artificial Intelligence, SETN 2016, Thessaloniki, Greece, May 18-20, 2016*, pages 3:1–3:6, 2016.

B. O'Sullivan, editor. *Proceedings of the 20^{th} Conference on Principles and Practice of Constraint Programming, Lyon, France, September 8-12*, volume 8656 of *Lecture Notes in Computer Science*, 2014. Springer.

F. Rossi, editor. *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, 2013. IJCAI/AAAI.

F. Rossi, P. van Beek, and T. Walsh, editors. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006. ISBN 978-0-444-52726-4. URL `http://www.sciencedirect.com/science/bookseries/15746526/2`.

E. P. K. Tsang. *Foundations of constraint satisfaction*. Computation in cognitive science. Academic Press, 1993.

P. van Beek and R. Dechter. Constraint tightness and looseness versus local and global consistency. *J. ACM*, 44(4):549–566, 1997.

Y. Zhang and E. C. Freuder. Conditional interchangeability and substitutability. In *Fourth Intl. Workshop on Symmetry and Constraint Satisfaction Problems- SymCon'04*, pages 95–100, 2004.

Y. Zhang and R. H. C. Yap. Consistency and set intersection. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15*, pages 263–270, 2003.

Y. Zhang and R. H. C. Yap. Set intersection and consistency in constraint networks. *J. Artif. Intell. Res.*, 27:441–464, 2006.