New schemes for simplifying binary constraint satisfaction problems

Wady Naanaa

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
wady.naanaa@fsm.rnu.tn

received , *revised* , *accepted* .

Finding a solution to a Constraint Satisfaction Problem (CSP) is known to be an NP-complete 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, 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 a new tractable class of binary CSP.

1 Introduction

Constraint Satisfaction Problem (CSP) is a generic problem which is well suited to the encoding of many difficult combinatorial problems [12, 23, 24]. 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-complete 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*.

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. The class of problems whose underlying hyper-graphs have a bounded hyper-tree width is one of the best known structural tractable class [15]. 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 [19]. More recent works studied a new kind of tractability, called *hybrid* tractability [9, 11, 14, 21, 25, 27, 28]. 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 of relational properties.

Among the multitude of works on hybrid tractability, the one presented in [9] has initiated a lot of efforts and developments, all aiming to simplify CSPs. The idea is centred on a specific 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 [5] 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 [10] and *Extendable-Triple Property* (ETP), which is based on a pattern that generalizes the broken triangle [20], 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 [21], 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 [10] 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) [22]. Contrary to k-BTP and ETP, WBTP requires no pre-established local consistency. In turn, WBTP inspired the work presented in [8] 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 [7] 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 [8]. 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 [6, 4, 13].

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 [16], conditional interchangeability and substitutability [26], and value removability [3]. These notions have resulted in as many filtering algorithms, which were applied as preprocessing steps or integrated into CSP solvers in order to accelerate CSP solving.

The present paper proposes two schemes designed to make simplifications on binary CSP. The first scheme can be seen as an enhanced value merging scheme that generalizes the one based on the BTP [7]. 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 a new hybrid tractable class of binary CSP. Our second contribution consists in a powerful value removal scheme that can be viewed as a parametrized version of neighbourhood substitutability, a widely studied CSP reduction technique that was proposed in [16].

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 (x, v) 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 (x, v) 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 (x, v) and (y, w) 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-complete, which means that 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.

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 notation, a union of the form $S \cup \{e\}$ will be often abbreviated to $S \cup e$.

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 $(y, w) \in A$, there exists $v \in D_x$ such that $\{(x, v), (y, w)\}$ is consistent.*

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 P be a binary CSP instance on variable set X and let x be 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 exists $v \in D_x$ such that $A \cup (x, v)$ 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. Note also that 2-mergeable sub-domains correspond exactly to the value pairs that can be merged by means of the BTP [7].

EXAMPLE 1. In the graph depicted in Figure 1, the vertices represent some values of a binary CSP instance. The dashed ellipses are used to diagram sub-domains. We assume that u, u', v, v' and w belong all to domains of distinct variables. A pair of vertices are connected by an edge if and only if the associated values are consistent. D_x supports the 3-sized consistent assignment that uses values u, v and w since each of these values is consistent with a value of D_x. Similarly, D'_x supports the 2-sized consistent assignment that uses values u and u' since each of these values is consistent with a value of D'_x. In accordance with Definition 3, both D_x and D'_x are 3-mergeable. In contrast, D'_x is not 2-mergeable because of the consistent assignment using u and u'. Finally, D_x is trivially not 2-mergeable because of its size.

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

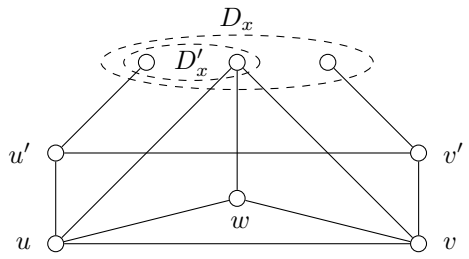

Fig. 1: Sub-domains D_x and D'_x are 3-mergeable, but neither D_x nor D'_x is 2-mergeable.

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 $(y, \bar{v}) \in A$ such that $(v, \bar{v}) \notin R_{x,y}$. Consider, therefore, $\bar{A}$ the subset of A composed by the $(y, \bar{v})$'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-mergeble and contradicts the hypothesis. □

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, an 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. This latter situations is illustrated in the following example.

EXAMPLE 2. The sub-domain of Figure 2-left, D_x, is not 2-mergeable because it supports the 2-sized consistent assignment that can be formed by u and u' but this latter assignment cannot be consistently extended by a 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 graph depicted in the middle of Figure 2, sub-domain D'_x and all its 2-sized subsets are neither 3-mergeable nor 2-mergeable. Finally, in Figure 2-right, D''_x is not 3-mergeable but all its proper subsets are 3-mergeable.

We now focus on how to benefit from m-mergeable sub-domains to simply 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. In addition, we prove that every solution of the reduced instance 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.

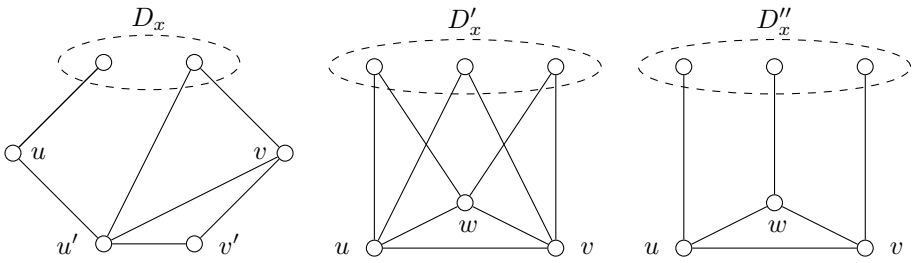

Fig. 2: Sub-domain D_x is 3-mergeable but not 2-mergeable. D'_x, as well as all its 2-sized subsets are 3-unmergeable. D''_x is 3-unmergeable but all its 2-sized subsets are 3-mergeable.

Definition 4 *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 4 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 4. 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 $A \cup (x, v)$ 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 4, this implies that $v \in R'_x$, which means that $A \cup (x, v)$ satisfies the unique unary constraint of P' that has x as scope. So, let us turn to binary constraints. Unary

assignment (x, v) is consistent, w.r.t. the constraints of P, with every unary assignment $(y, w) \in A$ because $A \cup (x, v)$ is a solution of P. It follows that $(v, w) \in R_{x,y}$, for all $(y, w) \in A$, and since $v \notin D_x$, we obtain, by Definition 4, that $(v, w) \in R'_{x,y}$, for all $(y, w) \in A$. This means that (x, v) is consistent, w.r.t. the constraints of P', with all the elements of A. It follows that $A \cup (x, v)$ is also a solution of P'.

– $v \in D_x$: which means that v is one of the merged values. Let $A' = A \cup (x, \mathbf{v})$, 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 4, $\mathbf{v}$ 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 $(y, w) \in A$. Note that (y, w) is also in A'. Since $A \cup (x, v)$ is a solution of P, we must have $(v, w) \in R_{x,y}$. It follows from Definition 4 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 $A' \cup (x, v')$ 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 $A = A' \cup (x, v)$ 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 $A = A' \cup (x, v)$ 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 $(y, w) \in A'$ 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 $(y, w) \in S'$ 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, $A' \cup (x, \mathbf{v})$ is a solution of P', which implies that $(\mathbf{v}, w) \in R'_{x,y}$, for all $(y, w) \in A'$. By Definition 4, we deduce that, for every $(y, w) \in A'$, 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 $M' \cup (x, v)$ 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 $A' \cup (x, v')$ satisfies all the unary constraints of P. On the other hand, $A' \cup (x, v')$ satisfies all the binary constraints of P not involving x in their scopes. It remains to show that $A' \cup (x, v')$ 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 $(y, w') \in A'$. We start from the fact that $A' \cup (x, v')$ is a solution of P', from which we deduce that $(v', w') \in R'_{x,y}$, for all $(y, w') \in A'$. Then we use the second point of Definition 4, with $v' \neq \mathbf{v}$, to deduce that $(v', w') \in R_{x,y}$, for all $(y, w') \in A'$. Thus, $A' \cup (x, v')$ 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 [7], 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 is varied 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 4.

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 [17]. 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 [18], 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)

```
A ← {(x, v) : x ∈ X ∧ v ∈ R_x}
A^m ← {A ∈ (A choose m) : A is consistent}
mrg ← true
while merged do
    mrg ← false
    for s ← 2 to m do
        for x ∈ X do
            D_x ← {D_x ∈ (R_x choose s) : Mergeable(x, D_x, A^m, C)}
            pck ←MaxSetPack(D_x)
            for D_x ∈ pck do
                Merge(D_x, C)
                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
```

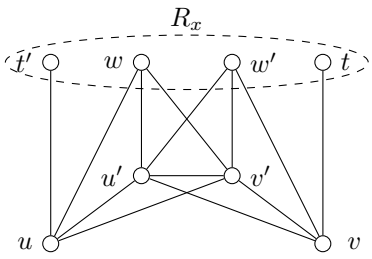

Fig. 3: A fragment of a binary CSP instance involving a 3-mergeable variable (x).

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 instance. However, prior to that, let us recall when could variables be eliminated as suggested in [5].

Definition 5 *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 *m-mergeable variables* as follows:

Definition 6 *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 3. The graph depicted in Figure 3 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'. Note here that all these values are assumed to belong to distinct domains. We see from Figure 3 that u, u', v', resp. v, v', u', can be consistently extended to x using w, resp., w'. The domain of x, R_x, can, therefore, be reduced to a singleton by means of a two-step merging. The first step merges the three left-most values of R_x, which form a 3-mergeable sub-domain. The second step consists in merging the two-valued domain resulting from the first merging. Note, however, that R_x contains non 2-mergeable sub-domains, which are $\{w, t\}$ and $\{w', t'\}$.

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 4) 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 et let x be a m-mergeable variable of P. Denote by P' the instance resulting from the merging of 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 4.

Suppose, for a sake of a contradiction, that x is not m-mergeable in P'. According to Definition 5, this implies that the domain of x in P', R'_x, contains a $\min(|R'_x|, m)$-sized sub-domain, say D'_x, chich 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 $(y, \bar{v}') \in A'$ 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 $(y, \bar{\mathbf{v}}) \in A'$ such that $(\mathbf{v}, \bar{\mathbf{v}}) \notin R'_{x,y}$. From Definition 4, 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 4, 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 exists $D' \subseteq D$, 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 6, this means that x is not m-mergeable in P, which contradicts the hypothesis.

□

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 7 *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*

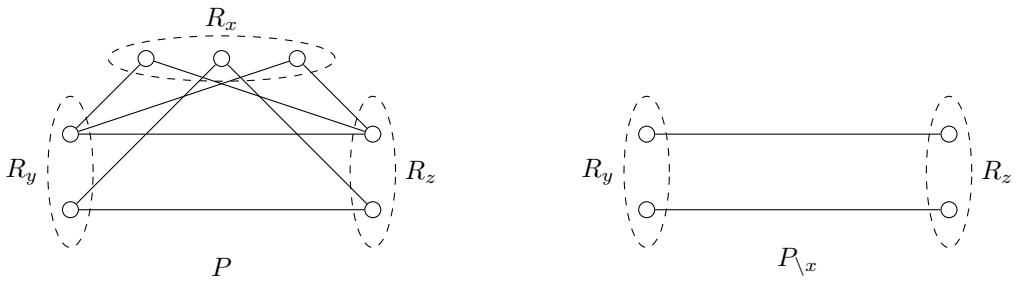

Fig. 4: A binary CSP instance whose variables are all 2-mergeable (Left). The CSP instance resulting from eliminating variable x (Right).

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

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

EXAMPLE 2. Figure 4 depicts a small binary CSP instance in which all the variables are 2-mergeable. It is therefore possible to remove any of the three variables. In the resulting 2-variable CSP, there is no 2-mergeable variables because of the condition imposing that the merging parameter must be smaller than the size of the variable set.

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 7. In contrast, the binary constraints remaining after a single-variable elimination are unchanged, as it can be seen from point (ii) of Definition 7. 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 $P_{\setminus x} = P'_{\setminus x}$.*

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 3. Recall also that P and P' differ only with regard to the constraints having x in their scopes.

According to Definition 7, $P_{\setminus x}$ and $P'_{\setminus x}$ have the same variable set, that is $X\setminus\{x\}$. Again, according to Definition 7, the binary constraints of $P_{\bar{x}}$ and those of $P'_{\setminus x}$ are the same. It remains to check that the unary constraints, i.e. the domains, of $P_{\setminus x}$ and $P'_{\setminus x}$ 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 $P_{\setminus x}$ and $P'_{\setminus x}$ respectively.

Suppose that $S_y \neq S'_y$, for some $y \in X\setminus\{x\}$, and proceed to get a contradiction. $S_y \neq S'_y$ implies that either $S_y\setminus S'_y \neq \varnothing$ or $S'_y\setminus S_y \neq \varnothing$. It $S_y\setminus S'_y \neq \varnothing$ then there exits $w \in S_y\setminus S'_y$. From Definition 7, 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 3 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 3, 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 exits $w' \in S'_y \backslash S_y$. From Definition 7, 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 3 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 3, 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 eliminated in the sense of Definition 5.

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 $A \cup \{(x, v)\}$ is a solution of P and show that A is a solution of $P_{\bar{x}}$. First, observe that A satisfies all the binary constraints of $P_{\setminus x}$, since these constraints are not changed while transforming P into $P_{\bar{x}}$. It remains to check A against the unary constraints of $P_{\bar{x}}$. Since $A \cup \{(x, v)\}$ is a solution of P, we have $(v, w) \in R_{x,y}$, for all $(y, w) \in A$, where $R_{x,y}$ denotes a binary constraint of P. According to Definition 7-(i), this implies that $w \in S_y$, for every $y \in X_{\setminus x}$, where S_y denotes the domain of y in $P_{\bar{x}}$. This means that A satisfies all the unary constraints of $P_{\setminus x}$, and then A is a solution of $P_{\setminus x}$.
$\Leftarrow$ Let A be a solution of $P_{\setminus x}$. 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 $P_{\setminus x}$, for every $(y, w) \in A$, we have $w \in S_y$, where S_y denotes the domain of y in $P_{\setminus x}$. According to Definition 7, this implies that, for every $(y, w) \in A$, 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 $(y, w) \in A$, we have $(v, w) \in R_{x,y}$, where v is the unique value of x. This implies that $A \cup \{(x, v)\}$ 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 $P_{\setminus x}$ is consistent.*

Proof: If the domain of x is empty then P is inconsistent. Moreover, in accordance with Definition 7, 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 7. 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 assignment, $\mathbf{A}$, 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 an 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.

Algorithm 3: ElimVariable(m, X, C)

```
A ← {(x, v) : x ∈ X ∧ v ∈ R_x}
A^m ← {A ∈ (A choose m) : A is consistent}
elim ← true
while elim and m < |X| do
    elim ← false
    for x ∈ X do
        if Eliminable(x, m, A^m, X, C) then
            X ← X\{x}
            C ← {R_σ ∈ C : x ∉ σ}
            for y ∈ X do
                R_y ← {w ∈ R_y : ∃v ∈ R_x, (v, w) ∈ R_{x,y}}
            if EmptyDomain(C) then
                elim ← false
                break
            // suppressing the m-assignments that use removed values
            for A ∈ A^m do
                for (x, v) ∈ A do
                    if v ∉ R_x then
                        A^m ← A^m\{A}
                        break
            elim ← true
```

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

```
if |R_x| ≤ m then
    return Mergeable(x, R_x, A^m, C)
else
    for D_x ∈ (R_x choose m) do
        if not Mergeable(x, D_x, A^m, C) then
            return false
    return true
```

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

Proof: We prove the contrapositive. Suppose that D_y is a sub-domain of $P_{\setminus x}$ which is not m-mergeable and show that D_y is not m-mergeable in P. By Definition 3, D_y not m-mergeable in $P_{\setminus x}$ implies that there exists 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 7, every constraint of $P_{\bar{x}}$ is a subset of the constraint having the same scope in P. 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. Again, by Definition 7, every binary constraint of $P_{\bar{x}}$ is identical to the corresponding binary constraint 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.

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 6, 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 6, this means that y is m-mergeable in $P_{\setminus x}$. □

We conclude this section by a corollary that identifies a new tractable binary CSP class based on 3-mergeable variables.

Corollary 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 6, 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 variables 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.

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

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 [12]. 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 $O(n^2d^2 + \max(d^3, n^3))$. □

Corollary 9 suggests that CSPs instances with 3-mergeable variables only 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.

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 [16]. 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 [2]:

Definition 8 *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 $\mathbf{A}$ the set of all unary assignments that can be formed from the variables of P and their respective domains. We have therefore:

$$\mathbf{A} = \{(x, v) : x \in X \ \wedge \ v \in R_x\} \tag{5}$$

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

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

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

Definition 9 *Consider a binary CSP instance with variable set X and let x be in X. A value v in R_x is* r-removable*, for some integer r, $1 \leq r < |X|$, if there exists 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 [16]. 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 $\mathcal{A}^1(x,v) \subseteq \mathcal{A}^1(x,w)$.

The following theorem provides the main result of this section. It relates the notion of r-removable values introduced in Definition 9 to the values that can be removed from the domains of a binary CSP in accordance with Definition 8.

Theorem 10 *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

$$\mathcal{A}^r(x,v) \quad \subseteq \quad \bigcup_{w \in D_x} \mathcal{A}^r(x,w) \tag{6}$$

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

- $\Rightarrow$ Let $A \cup (x,u)$ be a solution of P. It is clear that, if $u \neq v$ then $A \cup (x,u)$ is also a solution of $P|x{\neq}v$. So, assume henceforth that $u = v$, which implies that $A \cup (x,u)$ 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 $A \cup (x,w)$ is inconsistent, for every $w \in D_x$. But A is consistent, as well as every unary assignment $(x,w),\ w \in D_x$. It follows that, for every $w \in D_x$, there exists $(y,\bar{w}) \in A$ which is not consistent with (x,w). 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
$$\bar{A}^r \notin \bigcup_{w \in D_x} \mathcal{A}^r(x,w) \tag{7}$$
because $\bar{A}^r$ includes $\bar{A}$ and $\bar{A}$ is inconsistent with (x,w), for every $w \in D_x$. On the other hand, $\bar{A}^r \subseteq A$ and $A \cup (x,v)$ is consistent. This implies that $\bar{A}^r \cup (x,v)$ is consistent, and then $\bar{A}^r \in \mathcal{A}^r(x,v)$. This is in contradiction with (6) and (7).
- $\Leftarrow$ 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.

□

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 proposed algorithm are detailed in Algorithm 5. This latter can be viewed as a r-parametrized version of the algorithm proposed in [1], 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 10 *An assignment* A separates *a value* $v \in R_x$ *from a sub-domain* $D_x \subseteq R_x \backslash \{v\}$ *if* $A \cup (x, v)$ *is consistent but not* $A \cup (x, w)$*, for all* $w \in D_x$*.*

Bearing in mind Definition 9, it can be deduced from Definition 10 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.

- $\mathbf{A}$ is an array containing all unary assignments that can be obtained from the variables and their respective domains as indicated in (5). 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 $\mathbf{A}$. $\mathcal{A}^r$ 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 $(x, v) \in A$. We need $O(nd)$ assignment lists, one for every $(x, v) \in \mathbf{A}$, 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 (x, v) is inserted in the removed value list, rmvList, in order to propagate the effect of this removal. Whenever a pair (x, v) is eliminated, every assignment containing that pair becomes inconsistent. The propagation begins, therefore, by checking whether the values separated by r-assignments containing (x, v) 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 $\mathbf{A}$ 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 [7] and [16]. 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 a new hybrid tractable class of binary CSPs.

References

[1] 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.

[2] Lucas Bordeaux, Marco Cadoli, and Toni Mancini. Exploiting fixable, removable, and implied values in constraint satisfaction problems. In *Logic for Programming, Artificial Intelligence, and Reasoning, 11th International Conference, LPAR 2004, Montevideo, Uruguay, March 14-18, 2005, Proceedings*, pages 270–284, 2004.

[3] Lucas Bordeaux, Marco Cadoli, and Toni Mancini. A unifying framework for structural properties of csps: Definitions, complexity, tractability. *J. Artif. Intell. Res.*, 32:607–629, 2008.

[4] C. Carbonnel, D. A. Cohen, M. C. Cooper, and S. Zivny. On singleton arc consistency for natural CSPs defined by forbidden patterns. *CoRR*, abs/1704.06215, 2017.

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

[6] David A. Cohen, Martin C. Cooper, Guillaume Escamocher, and Stanislav Zivny. Variable and value elimination in binary constraint satisfaction via forbidden patterns. *J. Comput. Syst. Sci.*, 81(7):1127–1143, 2015.

Algorithm 5: REMOVEVALUEFROMCSP(X, C, r)

$\mathbf{A} \leftarrow \{(x, v) : x \in X \ \wedge \ v \in R_x\}$
$\mathcal{A}^r \leftarrow \{A \in \binom{\mathbf{A}}{r} : A \text{ is consistent}\}$
`// Initializing lists`
for $(x, v) \in \mathbf{A}$ **do** assgLst$[x, v] \leftarrow \varnothing$
for $A \in \mathcal{A}^r$ **do**
    sepLst$[A] \leftarrow \varnothing$
    for $(x, v) \in A$ **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 $\cup \ (x, v)$
            else
                sepLst$[A] \leftarrow$ sepLst$[A] \cup (x, v, D_x)$
while rmvLst $\neq \varnothing$ **do**
    $(y, w) \leftarrow$ 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 $\cup \ (x, v)$
                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
```


[7] Martin C. Cooper, Aymeric Duchein, Achref El Mouelhi, Guillaume Escamocher, Cyril Terrioux, and Bruno Zanuttini. Broken triangles: From value merging to a tractable class of general-arity constraint satisfaction problems. *Artif. Intell.*, 234:196–218, 2016.

[8] Martin C. Cooper, Achref El Mouelhi, and Cyril Terrioux. Extending broken triangles and enhanced value-merging. In *Principles and Practice of Constraint Programming - 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings*, pages 173–188, 2016.

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

[10] Martin C. Cooper, Philippe Jégou, and Cyril Terrioux. A microstructure-based family of tractable classes for CSPs. In *Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings*, pages 74–88, 2015.

[11] Martin C. Cooper and Stanislav 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.

[12] Rina Dechter. *Constraint processing*. Elsevier Morgan Kaufmann, 2003.

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

[14] Achref El Mouelhi, Philippe Jégou, and Cyril Terrioux. A hybrid tractable class for non-binary CSPs. In *2013 IEEE 25th International Conference on Tools with Artificial Intelligence, Herndon, VA, USA, November 4-6, 2013*, pages 947–954, 2013.

[15] Eugene C. Freuder. A sufficient condition for backtrack-bounded search. *J. ACM*, 32(4):755–761, October 1985.

[16] Eugene 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, 1991, Volume 1.*, pages 227–233, 1991.

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

[18] Magnús M. Halldórsson, Jan Kratochvíl, and Jan Arne Telle. Independent sets with domination constraints. *Discrete Applied Mathematics*, 99(1-3):39–54, 2000.

[19] Peter Jeavons and Martin Cooper. Tractable constraints on ordered domains. *Artif. Intell.*, 79(2):327–339, 1995.

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

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

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

[23] Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006.

[24] Edward P. K. Tsang. *Foundations of constraint satisfaction*. Computation in cognitive science. Academic Press, 1993.

[25] Peter van Beek and Rina Dechter. Constraint tightness and looseness versus local and global consistency. *J. ACM*, 44(4):549–566, 1997.

[26] Yuanlin Zhang and Eugene C. Freuder. Conditional interchangeability and substitutability. In *In 4th Intl. Workshop on Symmetry and Constraint Satisfaction Problems- SymCon'04*, pages 95–100, 2004.

[27] Yuanlin Zhang and Roland H. C. Yap. Consistency and set intersection. In *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pages 263–270, 2003.

[28] Yuanlin Zhang and Roland H. C. Yap. Set intersection and consistency in constraint networks. *J. Artif. Intell. Res.*, 27:441–464, 2006.