Strengthening Neighbourhood Substitution

Martin C. Cooper(✉)

ANITI, IRIT, University of Toulouse III, Toulouse, France
cooper@irit.fr

Abstract. Domain reduction is an essential tool for solving the constraint satisfaction problem (CSP). In the binary CSP, neighbourhood substitution consists in eliminating a value if there exists another value which can be substituted for it in each constraint. We show that the notion of neighbourhood substitution can be strengthened in two distinct ways without increasing time complexity. We also show the theoretical result that, unlike neighbourhood substitution, finding an optimal sequence of these new operations is NP-hard.

1 Introduction

Domain reduction is classical in constraint satisfaction. Indeed, eliminating inconsistent values by what is now known as arc consistency [27] predates the first formulation of the constraint satisfaction problem [23]. Maintaining arc consistency, which consists in eliminating values that can be proved inconsistent by examining a single constraint together with the current domains of the other variables, is ubiquitous in constraint solvers [1]. In binary CSPs, various algorithms have been proposed for enforcing arc consistency in $O(ed^2)$ time, where d denotes maximum domain size and e the number of constraints [3,24]. Generic constraints on a number of variables which is unbounded are known as global constraints. Arc consistency can be efficiently enforced for many types of global constraints [18]. This has led to the development of efficient solvers providing a rich modelling language. Stronger notions of consistency have been proposed for domain reduction which lead to more eliminations but at greater computational cost [1,2,28].

In parallel, other research has explored methods that preserve satisfiability of the CSP instance but do not preserve the set of solutions. When searching for a single solution, all but one branch of the explored search tree leads to a dead-end, and so any method for faster detection of unsatisfiability is clearly useful. An important example of such methods is the addition of symmetry-breaking constraints [4,17]. In this paper we concentrate on domain-reduction methods. One family of satisfiability-preserving domain-reduction operations is value merging. For example, two values can be merged if the so-called broken

This work was partially funded by ANITI, funded by the French program "Investing for the Future – PIA3" under Grant agreement n^o ANR-19-PI3A-0004.

H. Simonis (Ed.): CP 2020, LNCS 12333, pp. 126–142, 2020.
https://doi.org/10.1007/978-3-030-58475-7_8

triangle (BT) pattern does not occur on these two values [11]. Other value-merging rules have been proposed which allow less merging than BT-merging but at a lower cost [22] or more merging at a greater cost [12,25]. Another family of satisfiability-preserving domain-reduction operations are based on the elimination of values that are not essential to obtain a solution [15]. The basic operation in this family which corresponds most closely to arc consistency is neighbourhood substitution: a value b can be eliminated from a domain if there is another value a in the same domain such that b can be replaced by a in each tuple in each constraint relation (reduced to the current domains of the other variables) [14]. In binary CSPs, neighbourhood substitution can be applied until convergence in $O(ed^3)$ time [7]. In this paper, we study notions of substitutability which are strictly stronger than neighbourhood substitutability but which can be applied in the same $O(ed^3)$ time complexity. We say that one elimination rule R_1 is stronger than (subsumes) another rule R_2 if any value in a non-trivial instance (an instance with more than one variable) that can be eliminated by R_2 can also be eliminated by R_1, and is strictly stronger (strictly subsumes) if there is also at least one non-trivial instance in which R_1 can eliminate a value that R_2 cannot. Two rules are incomparable if neither is stronger than the other.

To illustrate the strength of the new notions of substitutability that we introduce in this paper, consider the instances shown in Fig. 1. These instances are all globally consistent (each variable-value assignment occurs in a solution) and neighbourhood substitution is not powerful enough to eliminate any values. In this paper, we introduce three novel value-elimination rules, defined in Sect. 2: SS, CNS and SCSS. We will show that snake substitution (SS) allows us to reduce all domains to singletons in the instance in Fig. 1(a). Using the notation $\mathcal{D}(x_i)$ for the domain of the variable x_i, conditioned neighbourhood-substitution (CNS), allows us to eliminate value 0 from $\mathcal{D}(x_2)$ and value 2 from $\mathcal{D}(x_3)$ in the instance shown in Fig. 1(b), reducing the constraint between x_2 and x_3 to a null constraint (the complete relation $\mathcal{D}(x_2) \times \mathcal{D}(x_3)$). Snake-conditioned snake-substitution (SCSS) subsumes both SS and CNS and allows us to reduce all domains to singletons in the instance in Fig. 1(c) (as well as in the instances in Fig. 1(a), (b)).

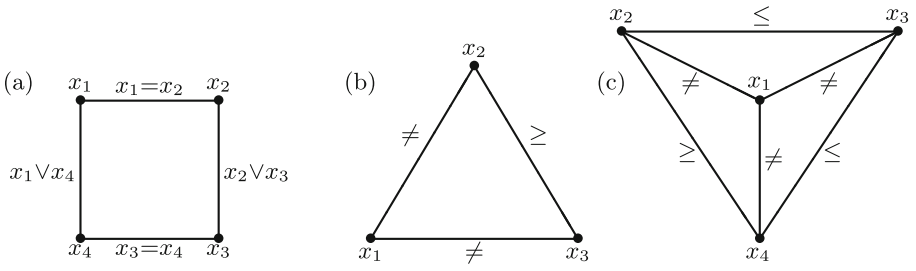

Fig. 1. (a) A 4-variable CSP instance over boolean domains; (b) a 3-variable CSP instance over domains $\{0, 1, 2\}$ with constraints $x_1 \neq x_2$, $x_1 \neq x_3$ and $x_2 \geq x_3$; (c) A 4-variable CSP instance over domain $\{0, 1, 2, 3\}$ with constraints $x_1 \neq x_2$, $x_1 \neq x_3$, $x_1 \neq x_4$, $x_2 \leq x_3$, $x_2 \geq x_4$ and $x_4 \leq x_3$.

In Sect. 2 we define the substitution operations SS, CNS and SCSS. In Sect. 3 we prove the validity of these three substitution operations, in the sense that they define satisfiability-preserving value-elimination rules. In Sect. 4 we explain in detail the examples in Fig. 1 and we give other examples from the semantic labelling of line drawings. Section 5 discusses the complexity of applying these value-elimination rules until convergence: the time complexity of SS and CNS is no greater than neighbourhood substitution (NS) even though these rules are strictly stronger. However, unlike NS, finding an optimal sequence of value eliminations by SS or CNS is NP-hard: this is shown in Sect. 6.

2 Definitions

We study binary constraint satisfaction problems.

A *binary CSP instance* $I = (X, \mathcal{D}, R)$ comprises

- a set X of n variables $x_1, \ldots, x_n$,
- a domain $\mathcal{D}(x_i)$ for each variable x_i $(i = 1, \ldots, n)$, and
- a binary constraint relation R_{ij} for each pair of distinct variables x_i, x_j $(i, j \in \{1, \ldots, n\})$

For notational convenience, we assume that there is exactly one binary relation R_{ij} for each pair of variables. Thus, if x_i and x_j do not constrain each other, then we consider that there is a *trivial constraint* between them with $R_{ij} = \mathcal{D}(x_i) \times \mathcal{D}(x_j)$. Furthermore, R_{ji} (viewed as a boolean matrix) is always the transpose of R_{ij}. A *solution* to I is an n-tuple $s = \langle s_1, \ldots, s_n \rangle$ such that $\forall i \in \{1, \ldots, n\}$, $s_i \in \mathcal{D}(x_i)$ and for each distinct $i, j \in \{1, \ldots, n\}$, $(s_i, s_j) \in R_{ij}$.

We say that $v_i \in D(x_i)$ has a *support* at variable x_j if $\exists v_j \in D(x_j)$ such that $(v_i, v_j) \in R_{ij}$. A binary CSP instance I is *arc consistent (AC)* if for all pairs of distinct variables x_i, x_j, each $v_i \in D(x_i)$ has a support at x_j [21].

In the following we assume that we have a binary CSP instance $I = (X, \mathcal{D}, R)$ over n variables and, for clarity of presentation, we write $j \neq i$ as a shorthand for $j \in \{1, \ldots, n\} \backslash \{i\}$. We use the notation $b \xrightarrow{ij} a$ for

$$\forall c \in \mathcal{D}(x_j),\ (b, c) \in R_{ij} \Rightarrow (a, c) \in R_{ij}$$

(i.e. a can be substituted for b in any tuple $(b, c) \in R_{ij}$).

Definition 1 [14]**.** *Given two values $a, b \in \mathcal{D}(x_i)$, b is* neighbourhood substitutable (NS) *by a if $\forall j \neq i$, $b \xrightarrow{ij} a$.*

It is well known and indeed fairly obvious that eliminating a neighbourhood substitutable value does not change the satisfiability of a binary CSP instance. We will now define stronger notions of substitutability. The proofs that these are indeed valid value-elimination rules are not directly obvious and hence are delayed until Sect. 3. We use the notation $b \overset{ik}{\rightsquigarrow} a$ for

$$\forall d \in \mathcal{D}(x_k),\ (b, d) \in R_{ik} \ \Rightarrow\ \exists e \in \mathcal{D}(x_k)((a, e) \in R_{ik} \ \wedge\ \forall \ell \notin \{i, k\}, d \xrightarrow{k\ell} e).$$

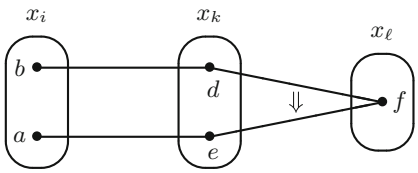

Fig. 2. An illustration of the definition of $b \overset{ik}{\rightsquigarrow} a$.

This is illustrated in Fig. 2, in which ovals represent domains, bullets represent values, a line joining two values means that these two values are compatible (so, for example, $(a, e) \in R_{ik}$), and the $\Downarrow$ means that $(d, f) \in R_{k\ell} \Rightarrow (e, f) \in R_{k\ell}$. Since e in this definition is a function of i, k, a and d, if necessary, we will write $e(i, k, a, d)$ instead of e. In other words, the notation $b \overset{ik}{\rightsquigarrow} a$ means that a can be substituted for b in any tuple $(b, d) \in R_{ik}$ provided we also replace d by $e(i, k, a, d)$. It is clear that $b \xrightarrow{ik} a$ implies $b \overset{ik}{\rightsquigarrow} a$ since it suffices to set $e(i, k, a, d) = d$ since, trivially, $d \xrightarrow{k\ell} d$ for all $\ell \notin \{i, k\}$. In Fig. 1(a), the value $0 \in \mathcal{D}(x_1)$ is snake substitutable by 1: we have $0 \overset{12}{\rightsquigarrow} 1$ by taking $e(1, 2, 1, 0) = 1$ (where the arguments of $e(i, k, a, d)$ are as shown in Fig. 2), since $(1, 1) \in R_{12}$ and $0 \xrightarrow{23} 1$; and $0 \overset{14}{\rightsquigarrow} 1$ since $0 \xrightarrow{14} 1$. Indeed, by a similar argument, the value 0 is snake substitutable by 1 in each domain.

Definition 2. *Given two values $a, b \in \mathcal{D}(x_i)$, b is* snake substitutable (SS) *by a if $\forall k \neq i$, $b \overset{ik}{\rightsquigarrow} a$.*

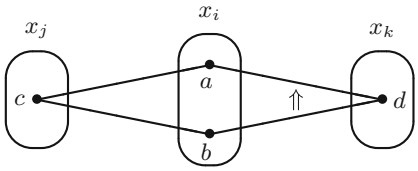

Fig. 3. An illustration of the definition of conditioned neighbourhood-substitutability of b by a (conditioned by x_j).

In the following two definitions, b can be eliminated from $\mathcal{D}(x_i)$ because it can be substituted by some other value in $\mathcal{D}(x_i)$, but this value is a function of the value assigned to another variable x_j. Definition 3 is illustrated in Fig. 3.

Definition 3. *Given $b \in \mathcal{D}(x_i)$, b is* conditioned neighbourhood-substitutable (CNS) *if for some $j \neq i$, $\forall c \in \mathcal{D}(x_j)$ with $(b, c) \in R_{ij}$, $\exists a \in \mathcal{D}(x_i) \backslash \{b\}$ such that $((a, c) \in R_{ij} \ \wedge \ \forall k \notin \{i, j\}, b \xrightarrow{ik} a)$.*

A CNS value $b \in \mathcal{D}(x_i)$ is substitutable by a value $a \in \mathcal{D}(x_i)$ where a is a function of the value c assigned to some other variable x_j. In Fig. 1(b), the value $0 \in \mathcal{D}(x_2)$ is conditioned neighbourhood-substitutable (CNS) with x_1 as the conditioning variable (i.e. $j = 1$ in Definition 3): for the assignments of 0 or 1 to x_1, we can take $a = 2$ since $0 \xrightarrow{23} 2$, and for the assignment 2 to x_1, we can take $a = 1$ since $0 \xrightarrow{23} 1$. By a symmetrical argument, the value $2 \in \mathcal{D}(x_3)$ is CNS, again with x_1 as the conditioning variable. We can note that in the resulting CSP instance, after eliminating 0 from $\mathcal{D}(x_2)$ and 2 from $\mathcal{D}(x_3)$, all domains can be reduced to singletons by applying snake substitutability.

Observe that CNS subsumes arc consistency; if a value $b \in \mathcal{D}(x_i)$ has no support c in $\mathcal{D}(x_j)$, then b is trivially CNS (conditioned by the variable x_j). It is easy to see from their definitions that SS and CNS both subsume NS (in instances with more than one variable), but that neither NS nor SS subsume arc consistency.

We now integrate the notion of snake substitutability in two ways in the definition of CNS: the value d (see Fig. 3) assigned to a variable $k \notin \{i, j\}$ may be replaced by a value e (as in the definition of $b \overset{ik}{\rightsquigarrow} a$, above), but the value c (see Fig. 3) assigned to the conditioning variable x_j may also be replaced by a value g. This is illustrated in Fig. 4.

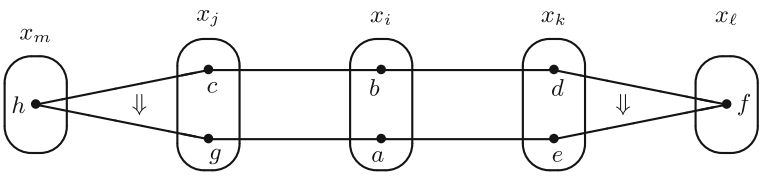

Fig. 4. An illustration of snake-conditioned snake-substitutability of b by a.

Definition 4. *A value* $b \in \mathcal{D}(x_i)$ *is* snake-conditioned snake-substitutable (SCSS) *if for some* $j \neq i$, $\forall c \in \mathcal{D}(x_j)$ *with* $(b, c) \in R_{ij}$, $\exists a \in \mathcal{D}(x_i) \backslash \{b\}$ *such that* $(\forall k \notin \{i, j\}, b \overset{ik}{\rightsquigarrow} a \ \wedge \ (\exists g \in \mathcal{D}(x_j)((a, g) \in R_{ij} \ \wedge \ \forall m \notin \{i, j\}, c \xrightarrow{jm} g)))$.

In Fig. 1(c), the value $3 \in \mathcal{D}(x_1)$ is snake-conditioned snake-substitutable (SCSS) with x_2 as the conditioning variable: for the assignment of 0 or 2 to x_2, we can take $a = 1$ since $3 \overset{13}{\rightsquigarrow} 1$ (taking $e(1, 3, 1, d) = 3$ for $d = 0, 1, 2$) and $3 \overset{14}{\rightsquigarrow} 1$ (taking $e(1, 4, 1, d) = 0$ for $d = 0, 1, 2$), and for the assignment of 1 to x_2, we can take $a = 2$ since $3 \overset{13}{\rightsquigarrow} 2$ (again taking $e(1, 3, 2, d) = 3$ for $d = 0, 1, 2$) and $3 \overset{14}{\rightsquigarrow} 2$ (again taking $e(1, 4, 2, d) = 0$ for $d = 0, 1, 2$). By similar arguments, all domains can be reduced to singletons following the SCSS elimination of values in the following order: 0 from $\mathcal{D}(x_1)$, 0, 1 and 2 from $\mathcal{D}(x_3)$, 0, 1 and 2 from $\mathcal{D}(x_2)$, 1, 2 and 3 from $\mathcal{D}(x_4)$ and 2 from $\mathcal{D}(x_1)$.

We can see that SCSS subsumes CNS by setting $g = c$ in Definition 4 and by recalling that $b \xrightarrow{ik} a$ implies that $b \overset{ik}{\rightsquigarrow} a$. It is a bit more subtle to see that SCSS subsumes SS: if b is snake substitutable by some value a, it suffices to choose a in Definition 4 to be this value (which is thus constant, i.e. not dependent on the value of c), then the snake substitutability of b by a implies that $b \overset{ik}{\rightsquigarrow} a$ for all $k \neq i, j$ and $b \overset{ij}{\rightsquigarrow} a$, which in turn implies that $(a, g) \in R_{ij} \wedge \forall m \notin \{i, j\}, c \xrightarrow{jm} g$ for $g = e(i, j, a, c)$; thus b is snake-conditioned snake-substitutable.

3 Value Elimination

It is well-known that NS is a valid value-elimination property, in the sense that if $b \in \mathcal{D}(x_i)$ is neighbourhood substitutable by a then b can be eliminated from $\mathcal{D}(x_i)$ without changing the satisfiability of the CSP instance [14]. In this section we show that SCSS is a valid value-elimination property. Since SS and CNS are subsumed by SCSS, it follows immediately that SS and CNS are also valid value-elimination properties.

Theorem 1. *In a binary CSP instance I, if $b \in \mathcal{D}(x_i)$ is snake-conditioned snake-substitutable then b can be eliminated from $\mathcal{D}(x_i)$ without changing the satisfiability of the instance.*

Proof. By Definition 4, for some $j \neq i$, $\forall c \in \mathcal{D}(x_j)$ with $(b, c) \in R_{ij}$, $\exists a \in \mathcal{D}(x_i) \backslash \{b\}$ such that

$$\forall k \notin \{i, j\}, b \overset{ik}{\rightsquigarrow} a \tag{1}$$

$$\wedge \quad \exists g \in \mathcal{D}(x_j)((a, g) \in R_{ij} \ \wedge \ \forall m \notin \{i, j\}, c \xrightarrow{jm} g). \tag{2}$$

We will only apply this definition for fixed i, j, and for fixed values a and c, so we can consider g as a constant (even though it is actually a function of i, j, a, c). Let $s = \langle s_1, \ldots, s_n \rangle$ be a solution to I with $s_i = b$. It suffices to show that there is another solution $t = \langle t_1, \ldots, t_n \rangle$ with $t_i \neq b$. Consider $c = s_j$. Since s is a solution, we know that $(b, c) = (s_i, s_j) \in R_{ij}$. Thus, according to the above definition of SCSS, there is a value $a \in \mathcal{D}(x_i)$ that can replace b (conditioned by the assignment $x_j = c = s_j$) in the sense that (1) and (2) are satisfied. Now, for each $k \notin \{i, j\}$, $b \overset{ik}{\rightsquigarrow} a$, i.e.

$$\forall d \in \mathcal{D}(x_k), \ (b, d) \in R_{ik} \ \Rightarrow \ \exists e \in \mathcal{D}(x_k)((a, e) \in R_{ik} \ \wedge \ \forall \ell \notin \{i, k\}, d \xrightarrow{k\ell} e).$$

Recall that e is a function of i, k, a and d. But we will only consider fixed i, a and a unique value of d dependant on k, so we will write $e(k)$ for brevity. Indeed, setting $d = s_k$ we can deduce from $(b, d) = (s_i, s_k) \in R_{ik}$ (since s is a solution) that

$$\forall k \neq i, j, \quad \exists e(k) \in \mathcal{D}(x_k)((a, e(k)) \in R_{ik} \ \wedge \ \forall \ell \notin \{i, k\}, s_k \xrightarrow{k\ell} e(k)). \tag{3}$$

Define the n-tuple t as follows:

$$t_r = \begin{cases} a & \text{if } r = i \\ s_r & \text{if } r \neq i \ \wedge \ (a, s_r) \in R_{ir} \\ g & \text{if } r = j \ \wedge \ (a, s_r) \notin R_{ir} \\ e(r) & \text{if } r \neq i, j \ \wedge \ (a, s_r) \notin R_{ir} \end{cases}$$

Clearly $t_i \neq b$ and $t_r \in \mathcal{D}(x_r)$ for all $r \in \{1, \ldots, n\}$. To prove that t is a solution, it remains to show that all binary constraints are satisfied, i.e. that $(t_k, t_r) \in R_{kr}$ for all distinct $k, r \in \{1, \ldots, n\}$. There are three cases: (1) $k = i$, $r \neq i$, (2) $k = j$, $r \neq i, j$, (3) $k, r \neq i, j$.

(1) There are three subcases: (a) $r = j$ and $(a, s_j) \notin R_{ij}$, (b) $r \neq i$ and $(a, s_r) \in R_{ir}$, (c) $r \neq i, j$ and $(a, s_r) \notin R_{ir}$. In case (a), $t_i = a$ and $t_j = g$, so from Eq. 2, we have $(t_i, t_r) = (a, g) \in R_{ij}$. In case (b), $t_i = a$ and $t_r = s_r$ and so, trivially, $(t_i, t_r) = (a, s_r) \in R_{ir}$. In case (c), $t_i = a$ and $t_r = e(r)$, so from Eq. 3, we have $(t_i, t_r) = (a, e(r)) \in R_{ir}$.

(2) There are four subcases: (a) $(a, s_r) \in R_{ir}$ and $(a, s_j) \in R_{ij}$, (b) $(a, s_r) \notin R_{ir}$ and $(a, s_j) \in R_{ij}$, (c) $(a, s_r) \in R_{ir}$ and $(a, s_j) \notin R_{ij}$, (d) $(a, s_r) \notin R_{ir}$ and $(a, s_j) \notin R_{ij}$. In case (a), $t_j = s_j$ and $t_r = s_r$, so $(t_j, t_r) \in R_{jr}$ since s is a solution. In case (b), $t_j = s_j$ and $t_r = e(r)$; setting $k = r$, $\ell = j$ in Eq. 3, we have $(t_j, t_r) = (s_j, e(r)) \in R_{jr}$ since $(s_j, s_r) \in R_{jr}$. In case (c), $t_j = g$ and $t_r = s_r$; setting $c = s_j$ and $m = r$ in Eq. 2 we can deduce that $(t_j, t_r) = (g, s_r) \in R_{jr}$ since $(s_j, s_r) \in R_{jr}$. In case (d), $t_j = g$ and $t_r = e(r)$. By the same argument as in case 2(b), we know that $(s_j, e(r)) \in R_{jr}$, and then setting $c = s_j$ and $m = r$ in Eq. 2, we can deduce that $(t_j, t_r) = (g, e(r)) \in R_{jr}$.

(3) There are three essentially distinct subcases: (a) $(a, s_r) \in R_{ir}$ and $(a, s_k) \in R_{ik}$, (b) $(a, s_r) \notin R_{ir}$ and $(a, s_k) \in R_{ik}$, (c) $(a, s_r) \notin R_{ir}$ and $(a, s_k) \notin R_{ik}$. In cases (a) and (b) we can deduce $(t_k, t_r) \in R_{kr}$ by the same arguments as in cases 2(a) and 2(b), above. In case (c), $t_k = e(k)$ and $t_r = e(k)$. Setting $\ell = r$ in Eq. 3, we have $s_k \xrightarrow{kr} e(k)$ from which we can deduce that $(e(k), s_r) \in R_{kr}$ since $(s_k, s_r) \in R_{kr}$. Reversing the roles of k and r in Eq. 3 (which is possible since they are distinct and both different to i and j), we also have that $s_r \xrightarrow{rk} e(r)$. We can then deduce that $(t_k, t_r) = (e(k), e(r)) \in R_{kr}$ since we have just shown that $(e(k), s_r) \in R_{kr}$.

We have thus shown that any solution s with $s_i = b$ can be transformed into another solution t that does not assign the value b to x_i and hence that the elimination of b from $\mathcal{D}(x_i)$ preserves satisfiability.

Corollary 1. *In a binary CSP instance I, if $b \in \mathcal{D}(x_i)$ is snake-substitutable or conditioned neighbourhood substitutable, then b can be eliminated from $\mathcal{D}(x_i)$ without changing the satisfiability of the instance.*

4 Examples

We have already illustrated the power of SS, CNS and SCSS using the examples given in Fig. 1.

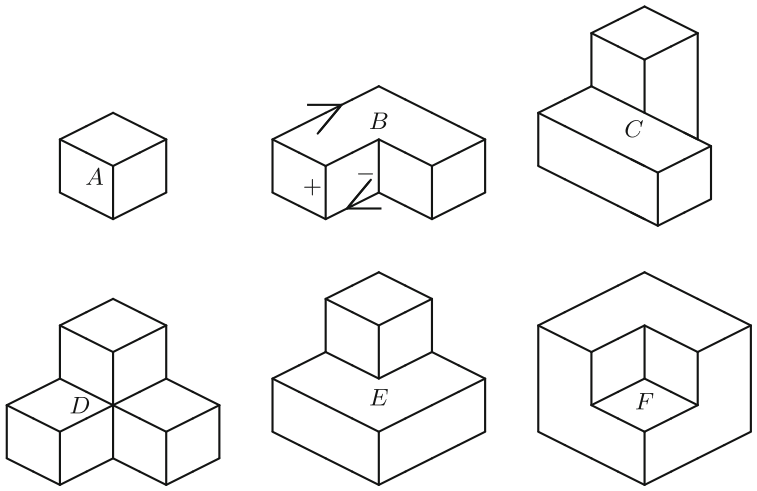

Fig. 5. The six different types of trihedral vertices: A, B, C, D, E, F.

To give a non-numerical example, we considered the impact of SS and CNS in the classic problem of labelling line-drawings of polyhedral scenes composed of objects with trihedral vertices [5,19,27]. There are six types of trihedral vertices: A, B, C, D, E and F, shown in Fig. 5. The aim is to assign each line in the drawing a semantic label among four possibilities: convex (+), concave (−) or occluding ($\leftarrow$ or $\rightarrow$ depending whether the occluding surface is above or below the line). Some lines in the top middle drawing in Fig. 5 have been labelled to illustrate the meaning of these labels. This problem can be expressed as a binary CSP by treating the junctions as variables. The domains of variables are given by the catalogue of physically realisable labellings of the corresponding junction according to its type. This catalogue of junction labellings is obtained by considering the six vertex types viewed from all possible viewpoints [5,19]. For example, there are 6 possible labellings of an L-junction, 8 for a T-junction, 5 for a Y-junction and 3 for a W-junction [9]. There is a constraint between any two junctions joined by a line: this line must have the same semantic label at both ends. We can also apply binary constraints between distant junctions: the 2Reg constraint limits the possible labellings of junctions such as A and D in Fig. 6, since two non-colinear lines, such as AB and CD, which separate the same two regions cannot both be concave [8,9].

The drawing shown in Fig. 6 is ambiguous. Any of lines AB, BC or CD could be projections of concave edges (meaning that the two blocks on the left side

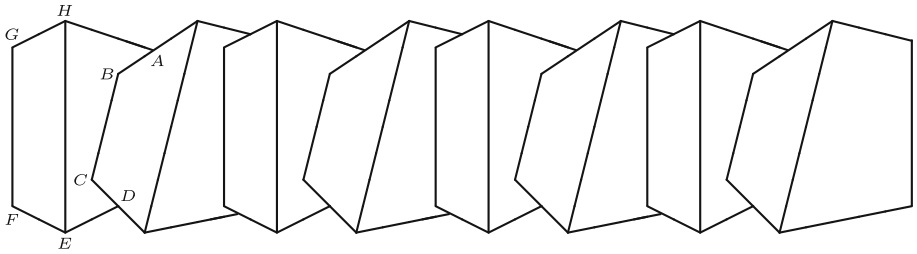

Fig. 6. An example from a family of line drawings whose exponential number of labellings is reduced to one by snake substitution.

of the figure are part of the same object) or all three could be projections of occluding edges (meaning that these two blocks are, in fact, separate objects). The drawing shown in Fig. 6 is an example of a family of line drawings. In this figure there are four copies of the basic structure, but there is a clear generalisation to drawings containing n copies of the basic structure. The ambiguity that we have pointed out above gives rise to an exponential number of valid labellings for this family of drawings. However, after applying arc consistency and snake substitution until convergence, each domain is a singleton in this family of line drawings. We illustrate this by giving one example of a snake substitution. After arc consistency has been established, the labelling $(-, +, -)$ for junction E in Fig. 6 is snake substitutable by $(\leftarrow, +, \leftarrow)$: snake substitutability follows from the fact that the labelling $(-, +, -)$ for E can be replaced by $(\leftarrow, +, \leftarrow)$ in any global labelling, provided the labelling $(\uparrow, -)$ for F is also replaced by $(\uparrow, \leftarrow)$ and the labelling $(\leftarrow, -, \leftarrow)$ for D is also replaced by $(\leftarrow, \leftarrow, \leftarrow)$.

Of course, there are line drawings where snake substitution is much less effective than in Fig. 6. Nevertheless, in the six drawings in Fig. 5, which are a representative sample of simple line drawings, 22 of the 73 junctions have their domains reduced to singletons by arc consistency alone and a further 20 junctions have their domains reduced to singletons when both arc consistency and snake substitution are applied. This can be compared with neighbourhood substitution which eliminates no domain values in this sample of six drawings. It should be mentioned that we found no examples where conditioned neighbourhood substitution could lead to the elimination of labellings in the line-drawing labelling problem.

5 Complexity

In a binary CSP instance $(X, \mathcal{D}, R)$, we say that two variables $x_i, x_j \in X$ constrain each other if there is a non-trivial constraint between them (i.e. $R_{ij} \neq \mathcal{D}(x_i) \times \mathcal{D}(x_j)$). Let $E \subseteq \{1, \ldots, n\} \times \{1, \ldots, n\}$ denote the set of pairs $\{i, j\}$ such that x_i, x_j constrain each other. We use d to denote the maximum size of the domains $\mathcal{D}(x_i)$ and $e = |E|$ to denote the number of non-trivial binary constraints. We have designed algorithms for applying CNS, SS and SCSS

until convergence using classical propagation techniques and data structures. The proofs of the following results can be found in the long version of this paper [10].

Theorem 2. *Value eliminations by snake substitution or conditioned neighbourhood substitution can be applied until convergence in $O(ed^3)$ time and $O(ed^2)$ space.*

Theorem 3. *It is possible to verify in $O(ed^3)$ time and $O(ed^2)$ space whether or not any value eliminations by SCSS can be performed on a binary CSP instance. Value eliminations by SCSS can then be applied until convergence in $O(end^5)$ time and $O(ed^2)$ space.*

We now investigate the interaction between arc consistency and the substitution operations we have introduced in this paper. It is well known that arc consistency eliminations can provoke new eliminations by neighbourhood substitution (NS) but that NS eliminations cannot destroy arc consistency [7]. It follows that arc consistency eliminations can provoke new eliminations by SS, CNS and SCSS (since these notions subsume NS). It is easily seen from Definition 3 that eliminations by CNS cannot destroy arc consistency, since each value c for x_j which had b as a support at x_i has another support a at x_i and this value a is also a support for all values d for other variable x_k which had b as a support at x_i. We therefore establish arc consistency before looking for eliminations by any form of substitution. Nonetheless, unlike CNS, eliminations by SS (or SCSS) can provoke new eliminations by arc consistency; however, these eliminations cannot themselves propagate. To see this, suppose that $b \in \mathcal{D}(x_i)$ is eliminated since it is snake-substitutable by a. If b is the only support of $d \in \mathcal{D}(x_k)$ at x_i, then d can then be eliminated by arc consistency. However, the elimination of d cannot provoke any new eliminations by arc consistency. To see this, recall that, by Definition 2 of SS, there is a value $e \in \mathcal{D}(x_k)$ such that for all $\ell \neq i, k$, for all $f \in \mathcal{D}(x_\ell)$, if d was a support for f at x_k then so was e (as illustrated in Fig. 2). Furthermore, since b was the only support for d at x_i, no other value in $\mathcal{D}(x_i)$ can lose its support when d is eliminated from $\mathcal{D}(x_k)$. In conclusion, the algorithm for applying SS has to apply this limited form of arc-consistency (without propagation) whereas the algorithm to apply CNS does not need to test for arc consistency since we assume that it has already been established. Furthermore, since AC is, in fact, subsumed by SCSS we do not explicitly need to test for it in the algorithm to apply SCSS.

We now consider the interaction between neighbourhood substitution and CNS. Recall that CNS subsumes neighbourhood substitution. It is also clear from Definition 3 of CNS that eliminating values by neighbourhood substitution cannot prevent elimination of other values by CNS. However, the converse is not true: eliminations by CNS can prevent eliminations of other values by NS. To see this, consider a 2-variable instance with constraint $(x_1 = x_2) \vee (x_2 = 0)$ and domains $\mathcal{D}(x_1) = \{1, \ldots, d-1\}$, $\mathcal{D}(x_2) = \{0, \ldots, d-1\}$. The value $0 \in \mathcal{D}(x_2)$ can be eliminated by CNS (conditioned by the variable x_1) since $\forall c \in \mathcal{D}(x_1)$, $\exists a = c \in \mathcal{D}(x_2) \backslash \{0\}$ such that $(a, c) \in R_{12}$. After eliminating 0 from $\mathcal{D}(x_2)$, no further eliminations are possible by CNS or neighbourhood substitution. However, in

the original instance we could have eliminated all elements of $\mathcal{D}(x_2)$ except 0 by neighbourhood substitution. Thus, in our algorithm to apply CNS, we give priority to eliminations by NS.

In this section we have seen that it is possible to apply CNS and SS until convergence in $O(ed^3)$ time and that it is possible to check SCSS in $O(ed^3)$ time. Thus, the complexity of applying the value-elimination rules CNS, SS and SCSS is comparable to the $O(ed^3)$ time complexity of applying neighbourhood substitution (NS) [7]. This is interesting because (in instances with more than one variable) CNS, SS and SCSS all strictly subsume NS.

6 Optimal Sequences of Eliminations

It is known that applying different sequences of neighbourhood operations until convergence produces isomorphic instances [7]. This is not the case for CNS, SS or SCSS. Indeed, as we show in this section, the problems of maximising the number of value-eliminations by CNS, SS or SCSS are all NP-hard. These intractability results do not detract from the utility of these operations, since any number of value eliminations reduces search-space size regardless of whether or not this number is optimal.

Theorem 4. *Finding the longest sequence of CNS value-eliminations or SCSS value-eliminations is NP-hard.*

Proof. We prove this by giving a polynomial reduction from the set cover problem [20], the well-known NP-complete problem which, given sets $S_1, \ldots, S_m \subseteq U$ and an integer k, consists in determining whether there are k sets $S_{i_1}, \ldots, S_{i_k}$ which cover U (i.e. such that $S_{i_1} \cup \ldots \cup S_{i_k} = U$). We can assume that $S_1 \cup \ldots \cup S_m = U$ and $k < m$, otherwise the problem is trivially solvable. Given sets $S_1, \ldots, S_m \subseteq U$, we create a 2-variable CSP instance with $\mathcal{D}(x_1) = \{1, \ldots, m\}$, $\mathcal{D}(x_2) = U$ and $R_{12} = \{(i, u) \mid u \in S_i\}$. We can eliminate value i from $\mathcal{D}(x_1)$ by CNS (with, of course, x_2 as the conditioning variable) if and only if $S_1, \ldots, S_{i-1}, S_{i+1}, \ldots, S_m$ cover U. Indeed, we can continue eliminating elements from $\mathcal{D}(x_1)$ by CNS provided the sets S_j ($j \in \mathcal{D}(x_1)$) still cover U. Clearly, maximising the number of eliminations from $\mathcal{D}(x_1)$ by CNS is equivalent to minimising the size of the cover. To prevent any eliminations from the domain of x_2 by CNS, we add variables x_3 and x_4 with domains $\{1, \ldots, m\}$, together with the three equality constraints $x_2 = x_3$, $x_3 = x_4$ and $x_4 = x_2$. To complete the proof for CNS, it is sufficient to observe that this reduction is polynomial.

It is easily verified that in this instance, CNS and SCSS are equivalent. Hence, this proof also shows that finding the longest sequence of SCSS value-eliminations is NP-hard.

In the proof of the following theorem, we need the following notion: we say that a sequence of value-eliminations by snake-substitution (SS) is *convergent* if no more SS value-eliminations are possible after this sequence of eliminations is applied.

Theorem 5. *Finding a longest sequence of snake-substitution value-eliminations is NP-hard.*

Proof. It suffices to demonstrate a polynomial reduction from the problem Max 2-Sat which is known to be NP-hard [16]. Consider an instance I_{2SAT} of Max 2-Sat with variables $X_1, \ldots, X_N$ and M binary clauses: the goal is to find a truth assignment to these variables which maximises the number of satisfied clauses. We will construct a binary CSP instance I_{CSP} on $O(N+M)$ variables, each with domain of size at most four, such that the convergent sequences S of SS value-eliminations in I_{CSP} correspond to truth assignments to $X_1, \ldots, X_N$ and the length of S is $\alpha N + \beta m$ where α, β are constants and m is the number of clauses of I_{2SAT} satisfied by the corresponding truth assignment.

We require four constructions (which we explain in detail below):

1. the construction in Fig. 7 simulates a Max 2-Sat literal X by a path of CSP variables joined by greater-than-or-equal-to constraints.
2. the construction in Fig. 8 simulates the relationship between a Max 2-Sat variable X and its negation $\overline{X}$.
3. the construction in Fig. 9 allows us to create multiple copies of a Max 2-Sat literal X.
4. the construction in Fig. 10 simulates a binary clause $X \vee Y$ where X, Y are Max 2-Sat literals.

In each of these figures, each oval represents a CSP variable with the bullets inside the oval representing the possible values for this variable. If there is a non-trivial constraint between two variables x_i, x_j this is represented by joining up with a line those pairs of values a, b such that $(a, b) \in R_{ij}$. Where the constraint has a compact form, such as $x_1 \geq x_2$ this is written next to the constraint. In the following, we write $b \overset{x_i}{\rightsquigarrow} a$ if $b \in \mathcal{D}(x_i)$ is snake substitutable by $a \in \mathcal{D}(x_i)$. Our constructions are such that the only value that can be eliminated from any domain by SS is the value 2.

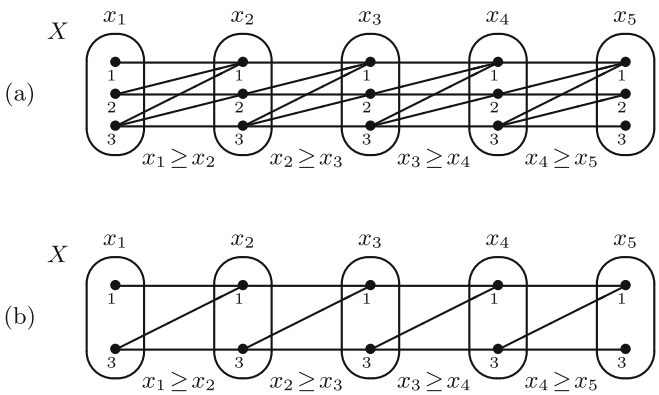

Fig. 7. A construction to simulate a Max 2-Sat variable X: (a) $X = 0$, (b) $X = 1$.

Figure 7(a) shows a path of CSP variables constrained by greater-than-or-equal-to constraints. The end variables x_1 and x_5 are constrained by other variables that, for clarity of presentation, are not shown in this figure. If value 2 is eliminated from $\mathcal{D}(x_1)$, then we have $2 \overset{x_2}{\rightsquigarrow} 3$. In fact, 2 is neighbourhood substitutable by 3. Once the value 2 is eliminated from $\mathcal{D}(x_2)$, we have $2 \overset{x_3}{\rightsquigarrow} 3$. Indeed, eliminations of the value 2 propagate so that in the end we have the situation shown in Fig. 7(b). By a symmetrical argument, the elimination of the value 2 from $\mathcal{D}(x_5)$ propagates from right to left (this time by neighbourhood substitution by 1) to again produce the situation shown in Fig. 7(b). It is easily verified that, without any eliminations from the domains $\mathcal{D}(x_1)$ or $\mathcal{D}(x_5)$, no values for the variables x_2, x_3, x_4 are snake-substitutable. Furthermore, the values 1 and 3 for the variables x_2, x_3, x_4 are not snake-substitutable even after the elimination of the value 2 from all domains. So we either have no eliminations, which we associate with the truth assignment $X = 0$ (where X is the Max 2-Sat literal corresponding to this path of variables in I_{CSP}) or the value 2 is eliminated from all domains, which we associate with the truth assignment $X = 1$.

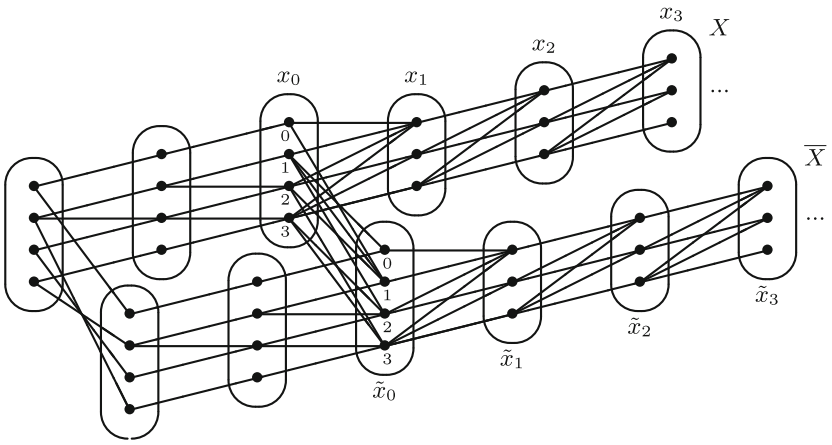

Fig. 8. A construction to simulate a Max 2-Sat variable X and its negation $\overline{X}$.

The construction in Fig. 8 joins the two path-of-CSP-variables constructions corresponding to the literals X and $\overline{X}$. This construction ensures that exactly one of X and $\overline{X}$ are assigned the value 1. It is easy (if tedious) to verify that the only snake substitutions that are possible in this construction are $2 \overset{x_0}{\rightsquigarrow} 3$ and $2 \overset{\tilde{x}_0}{\rightsquigarrow} 3$, but that after elimination of the value 2 from either of $\mathcal{D}(x_0)$ or $\mathcal{D}(\tilde{x}_0)$, the other snake substitution is no longer valid. Once, for example, 2 has been eliminated from $\mathcal{D}(x_0)$, then this elimination propagates along the path of CSP variables $(x_1, x_2, x_3, \ldots)$ corresponding to X, as shown in Fig. 7(b). By a symmetrical argument, if 2 is eliminated from $\mathcal{D}(\tilde{x}_0)$, then this elimination propagates along the path of CSP variables $(\tilde{x}_1, \tilde{x}_2, \tilde{x}_3, \ldots)$ corresponding to $\overline{X}$.

Thus, this construction simulates the assignment of a truth value to X and its complement to $\overline{X}$.

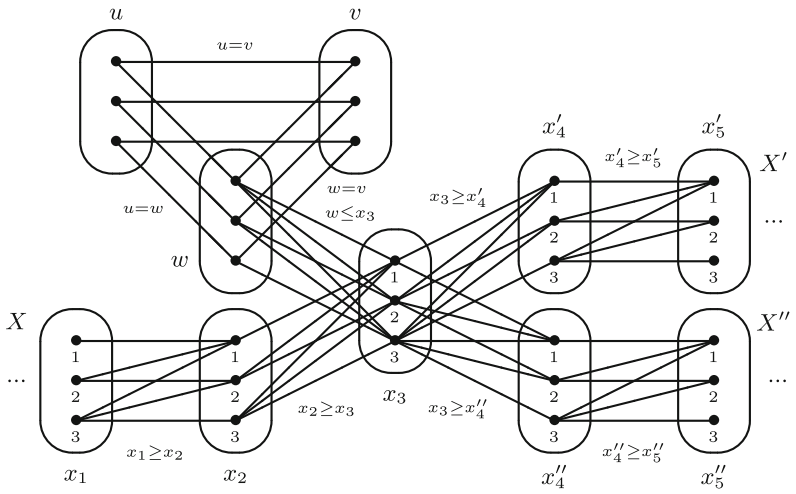

Fig. 9. A construction to create two copies X' and X'' of the Max 2-Sat variable X.

Since any literal of I_{2SAT} may occur in several clauses, we need to be able to make copies of any literal. Figure 9 shows a construction that creates two copies X', X'' of a literal X. This construction can easily be generalised to make k copies of a literal, if required, by having k identical paths of greater-than-equal-to constraints on the right of the figure all starting at the pivot variable x_3. Before any eliminations are performed, no snake substitutions are possible in this construction. However, once the value 2 has been eliminated from $\mathcal{D}(x_1)$, eliminations propagate, as in Fig. 7: the value 2 can successively be eliminated from the domains of variables x_2, x_3, x'_4, x'_5, and x''_4, x''_5. Each elimination is in fact by neighbourhood substitution, as in Fig. 7. These eliminations mean that we effectively have two copies X', X'' of the literal X. The triangle of equality constraints at the top left of this construction is is there simply to prevent propagation in the reverse direction: even if the value 2 is eliminated from the domains of x'_5, x'_4 and x''_5, x''_4 by the propagation of eliminations from the right, this cannot provoke the elimination of the value 2 from the domain of the pivot variable x_3.

Finally, the construction of Fig. 10 simulates the clause $X \vee Y$. In fact, this construction simply joins together the paths of CSP-variables corresponding to the two literals X, Y, via a variable z. It is easily verified that the elimination of the value 2 from the domain of x_1 allows the propagation of eliminations of the value 2 from the domains of x_2, z, y_2, y_1 in exactly the same way as the propagation of eliminations in Fig. 7. Similarly, the elimination of the value 2 from the domain of y_1 propagates to all other variables in the opposite order y_2,

z, x_2, x_1. Thus, if one or other of the literals X or Y in the clause is assigned 1, then the value 2 is eliminated from all domains of this construction. Eliminations can propagate back up to the pivot variable (x_3 in Fig. 9) but no further, as explained in the previous paragraph.

Putting all this together, we can see that there is a one-to-one correspondence between convergent sequences of SS value-eliminations and truth assignments to the variables of the MAX 2-SAT instance. Furthermore, the number of SS value-eliminations is maximised when this truth assignment maximises the number of satisfied clauses, since it is $\alpha N + \beta m$ where α is the number of CSP-variables in each path of greater-than-or-equal-to constraints corresponding to a literal, β is the number of CSP-variables in each clause construction and m is the number of satisfied clauses. This reduction is clearly polynomial.

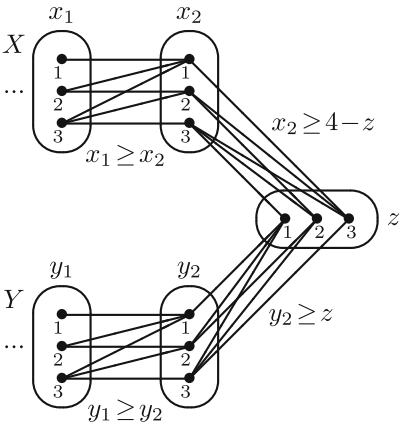

Fig. 10. A construction to simulate a MAX 2-SAT clause $X \vee Y$.

7 Discussion and Conclusion

Both snake substitutability (SS) and conditioned neighbourhood substitutability (CNS) strictly subsume neighbourhood substitution but nevertheless can be applied in the same $O(ed^3)$ time complexity. We have also given a more general notion of substitution (SCSS) subsuming both these rules that can be detected in $O(ed^3)$ time. The examples in Fig. 1 show that these three rules are strictly stronger than neighborhood substitution and that SS and CNS are incomparable.

An avenue of future research is the generalisation to valued CSPs. The notion of snake substitutability has already been generalised to binary valued CSPs and it has been shown that it is possible to test this notion in $O(ed^4)$ time if the aggregation operator is addition over the non-negative rationals [13]. However, further research is required to determine the complexity of applying this operation until convergence.

It is possible to efficiently find all (or a given number of) solutions to a CSP after applying NS: given the set of all solutions to the reduced instance, it is possible to find $K \geq 1$ solutions to the original instance I (or to determine that I does not have K solutions) in $O(K(de + n^2))$ time [7]. This also holds for CNS, since, as for NS, for each solution s found and for each value b eliminated from some domain $\mathcal{D}(x_i)$, it suffices to test each putative solution obtained by replacing s_i by b. Unfortunately, the extra strength of snake substitution (SS) is here a drawback, since, by exactly the same argument as for the $\exists 2snake$ value-elimination rule (which is a weaker version of SS) [6], we can deduce that

determining whether a binary CSP instance has two or more solutions is NP-hard, even given the set of solutions to the reduced instance after applying SS.

References

1. Bessiere, C.: Constraint propagation. In: Rossi et al. [26], pp. 29–83. https://doi.org/10.1016/S1574-6526(06)80007-6
2. Bessière, C., Debruyne, R.: Optimal and suboptimal singleton arc consistency algorithms. In: Kaelbling, L.P., Saffiotti, A. (eds.) IJCAI-2005, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, pp. 54–59. Professional Book Center (2005). http://ijcai.org/Proceedings/05/Papers/0495.pdf
3. Bessière, C., Régin, J., Yap, R.H.C., Zhang, Y.: An optimal coarse-grained arc consistency algorithm. Artif. Intell. **165**(2), 165–185 (2005). https://doi.org/10.1016/j.artint.2005.02.004
4. Chu, G., Stuckey, P.J.: Dominance breaking constraints. Constraints Int. J. **20**(2), 155–182 (2014). https://doi.org/10.1007/s10601-014-9173-7
5. Clowes, M.B.: On seeing things. Artif. Intell. **2**(1), 79–116 (1971). https://doi.org/10.1016/0004-3702(71)90005-1
6. Cohen, D.A., Cooper, M.C., Escamocher, G., Zivny, S.: Variable and value elimination in binary constraint satisfaction via forbidden patterns. J. Comput. Syst. Sci. **81**(7), 1127–1143 (2015). https://doi.org/10.1016/j.jcss.2015.02.001
7. Cooper, M.C.: Fundamental properties of neighbourhood substitution in constraint satisfaction problems. Artif. Intell. **90**(1–2), 1–24 (1997). https://doi.org/10.1016/S0004-3702(96)00018-5
8. Cooper, M.C.: Constraints between distant lines in the labelling of line drawings of polyhedral scenes. Int. J. Comput. Vis. **73**(2), 195–212 (2007). https://doi.org/10.1007/s11263-006-9783-7
9. Cooper, M.C.: Line Drawing Interpretation. Springer, Heidelberg (2008). https://doi.org/10.1007/978-1-84800-229-6
10. Cooper, M.C.: Strengthening neighbourhood substitution. CoRR abs/2007.06282 (2020). https://arxiv.org/abs/2007.06282
11. Cooper, M.C., Duchein, A., Mouelhi, A.E., Escamocher, G., Terrioux, C., Zanuttini, B.: Broken triangles: from value merging to a tractable class of general-arity constraint satisfaction problems. Artif. Intell. **234**, 196–218 (2016). https://doi.org/10.1016/j.artint.2016.02.001
12. Cooper, M.C., El Mouelhi, A., Terrioux, C.: Extending broken triangles and enhanced value-merging. In: Rueher, M. (ed.) CP 2016. LNCS, vol. 9892, pp. 173–188. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-44953-1_12
13. Cooper, M.C., Jguirim, W., Cohen, D.A.: Domain reduction for valued constraints by generalising methods from CSP. In: Hooker, J. (ed.) CP 2018. LNCS, vol. 11008, pp. 64–80. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98334-9_5
14. Freuder, E.C.: Eliminating interchangeable values in constraint satisfaction problems. In: Dean, T.L., McKeown, K.R. (eds.) Proceedings of the 9th National Conference on Artificial Intelligence, vol. 1, pp. 227–233. AAAI Press/The MIT Press (1991). http://www.aaai.org/Library/AAAI/1991/aaai91-036.php
15. Freuder, E.C., Wallace, R.J.: Replaceability and the substitutability hierarchy for constraint satisfaction problems. In: Benzmüller, C., Lisetti, C.L., Theobald, M. (eds.) GCAI 2017, 3rd Global Conference on Artificial Intelligence. EPiC Series in Computing, vol. 50, pp. 51–63. EasyChair (2017). http://www.easychair.org/publications/paper/mKkF

16. Garey, M.R., Johnson, D.S., Stockmeyer, L.J.: Some simplified NP-complete graph problems. Theor. Comput. Sci. **1**(3), 237–267 (1976). https://doi.org/10.1016/0304-3975(76)90059-1
17. Gent, I.P., Petrie, K.E., Puget, J.: Symmetry in constraint programming. In: Rossi et al. [26], pp. 329–376. https://doi.org/10.1016/S1574-6526(06)80014-3
18. van Hoeve, W., Katriel, I.: Global constraints. In: Rossi et al. [26], pp. 169–208. https://doi.org/10.1016/S1574-6526(06)80010-6
19. Huffman, D.A.: Impossible objects as nonsense sentences. In: Meltzer, B., Michie, D. (eds.) Machine Intelligence, vol. 6, pp. 295–323. Edinburgh University Press (1971)
20. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) Proceedings of Symposium on the Complexity of Computer Computations. The IBM Research Symposia Series, pp. 85–103. Plenum Press, New York (1972). http://www.cs.berkeley.edu/%7Eluca/cs172/karp.pdf
21. Lecoutre, C.: Constraint Networks Techniques and Algorithms. ISTE/Wiley, Hoboken (2009)
22. Likitvivatanavong, C., Yap, R.H.C.: Many-to-many interchangeable sets of values in CSPs. In: Shin, S.Y., Maldonado, J.C. (eds.) Proceedings of the 28th Annual ACM Symposium on Applied Computing, SAC 2013, pp. 86–91. ACM (2013). https://doi.org/10.1145/2480362.2480382
23. Mackworth, A.K.: Consistency in networks of relations. Artif. Intell. **8**(1), 99–118 (1977). https://doi.org/10.1016/0004-3702(77)90007-8
24. Mohr, R., Henderson, T.C.: Arc and path consistency revisited. Artif. Intell. **28**(2), 225–233 (1986). https://doi.org/10.1016/0004-3702(86)90083-4
25. Naanaa, W.: New schemes for simplifying binary constraint satisfaction problems. Discrete Math. Theor. Comput. Sci. (2019)
26. Rossi, F., van Beek, P., Walsh, T. (eds.): Handbook of Constraint Programming, Foundations of Artificial Intelligence, vol. 2. Elsevier, Amsterdam (2006). http://www.sciencedirect.com/science/bookseries/15746526/2
27. Waltz, D.: Understanding line drawings of scenes with shadows. In: Winston, P.H. (ed.) The Psychology of Computer Vision. Computer Science Series, pp. 19–91. McGraw-Hill, New York (1975)
28. Woodward, R.J., Karakashian, S., Choueiry, B.Y., Bessiere, C.: Revisiting neighborhood inverse consistency on binary CSPs. In: Milano, M. (ed.) CP 2012. LNCS, pp. 688–703. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33558-7_50