



Strengthening Neighbourhood Substitutability

Martin Cooper

► To cite this version:

Martin Cooper. Strengthening Neighbourhood Substitutability. 13èmes Journées d'Intelligence Artificielle Fondamentale (JIAF 2019), Jul 2019, Toulouse, France. hal-02302979

HAL Id: hal-02302979

<https://hal.science/hal-02302979>

Submitted on 1 Oct 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Strengthening Neighbourhood Substitutability

Martin C. Cooper

IRIT, Université de Toulouse III, France
cooper@irit.fr

Résumé

La réduction de domaines est un outil essentiel dans la résolution de problèmes de satisfaction de contraintes (CSP). Pour les CSP binaires, la substitution de voisinage consiste à éliminer une valeur s'il existe une autre valeur qui peut la remplacer dans chaque contrainte. Nous démontrons qu'il est possible de rendre plus forte la notion de substitution de voisinage de deux façons distinctes sans augmenter la complexité temporelle. Nous démontrons que contrairement à ce qui se passe pour la substitution de voisinage, trouver la séquence optimale de ces nouvelles opérations est NP-difficile.

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 [28] 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 [25, 3]. 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 [17]. 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, 29].

In parallel, other research has explored domain-reduction 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. One family of satisfiability-preserving domain-reduction operations is value merging. For example, two values can be merged if the so-called broken triangle (BT) pattern does not occur on these two values [10]. 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 [11, 26]. Another family of satisfiability-preserving domain-reduction operations are based on the elimination of values that are not essential to obtain a solution [14]. 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) [13]. In binary CSPs, neighbourhood substitution can be applied until convergence in $O(ed^3)$ time [6]. 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

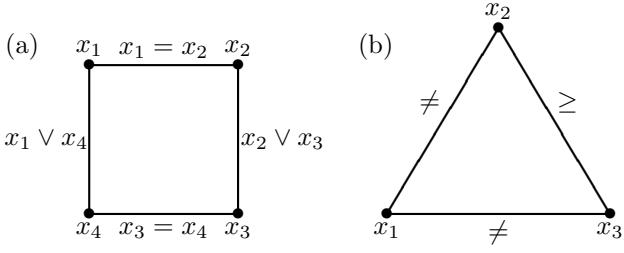


Figure 1: (a) A 4-variable CSP instance over boolean domains; (b) a 3-variable CSP instance over domains $\{0, 1, 2\}$ with constraints $x_1 \neq x_2$, $x_1 \neq x_3$ and $x_2 \geq x_3$.

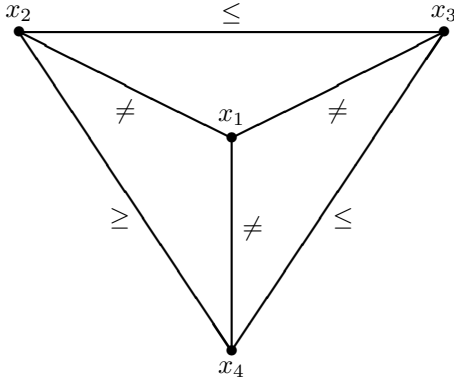


Figure 2: A 4-variable CSP instance over domain $\{0, 1, 2, 3\}$ with constraints $x_1 \neq x_2$, $x_1 \neq x_3$, $x_1 \neq x_4$, $x_2 \leq x_3$, $x_2 \geq x_4$ and $x_4 \leq x_3$.

complexity.

To illustrate the strength of the new notions of substitutability that we introduce in this paper, consider the instances shown in Figure 1 and Figure 2. 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 Section 2: SS, CNS and SCSS. We will show that snake substitution (SS) allows us to reduce all domains to singletons in the instance in Figure 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 Figure 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 Figure 2 (as well as both instances in Figure 1).

In Section 2 we define the substitution operations SS, CNS and SCSS. In Section 3 we prove the validity of these three substitution operations, in the sense that they define satisfiability-preserving value-elimination rules. In Section 4 we explain in detail the examples in Figures 1 and 2 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. Unlike NS, finding an optimal sequence of value eliminations by SS or CNS is NP-hard: this is shown in Section 6.

2 Definitions

We study binary constraint satisfaction problems.

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

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

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 $s = \langle s_1, \dots, s_n \rangle$ such that $\forall i \in \{1, \dots, n\}$, $s_i \in \mathcal{D}(x_i)$ and for each distinct $i, j \in \{1, \dots, n\}$, $(s_i, s_j) \in R_{ij}$.

We say that $v_i \in \mathcal{D}(x_i)$ has a *support* at variable x_j if $\exists v_j \in \mathcal{D}(x_j)$ such that $(v_i, v_j) \in R_{ij}$. A binary CSP instance I is *arc consistent* if for all pairs of distinct variables x_i, x_j , each $v_i \in \mathcal{D}(x_i)$ has a support at x_j [20].

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 $i \in \{1, \dots, n\} \setminus \{i\}$. 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 [13] *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

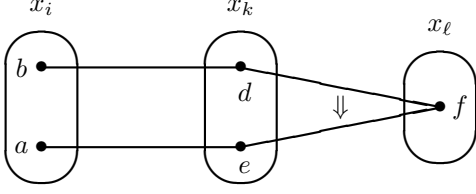


Figure 3: An illustration of the definition of $b \overset{ik}{\rightsquigarrow} a$.

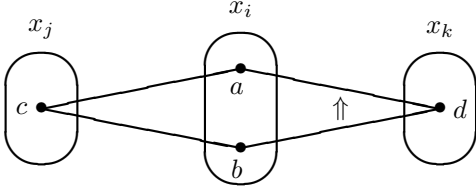


Figure 4: An illustration of the definition of conditioned neighbourhood-substitutability of b by a (conditioned by x_j).

are delayed until Section 3. We use the notation $b \overset{ik}{\rightsquigarrow} a$ for

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

This is illustrated in Figure 3, 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 \uparrow means that $(e, f) \in R_{k\ell} \Rightarrow (d, f) \in R_{k\ell}$. 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 $\ell \notin \{i, k\}$.

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

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 Figure 4.

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

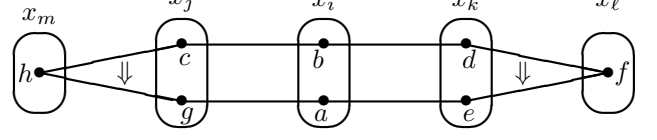


Figure 5: An illustration of the definition of snake-conditioned snake-substitutability of b by a .

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 . 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 Figure 4) 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 Figure 4) assigned to the conditioning variable x_j may also be replaced by a value g . This is illustrated in Figure 5.

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

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 [13]. 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 $(b, c) \in R_{ij}$, $\exists a \in \mathcal{D}(x_i) \setminus \{b\}$ such that

$$\forall k \notin \{i, j\}, b \overset{ik}{\rightsquigarrow} a \quad \wedge \quad (1)$$

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

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 $s = \langle s_1, \dots, s_n \rangle$ be a solution to I with $s_i = b$. It suffices to show that there is another solution $t = \langle t_1, \dots, t_n \rangle$ 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.

$$\forall d \in \mathcal{D}(x_k), (b, d) \in R_{ik} \Rightarrow$$

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

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 dependent 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 $\forall k \neq i, j$,

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

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 $t_i \in \mathcal{D}(x_r)$ for all $r \in \{1, \dots, n\}$. 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 $k, r \in \{1, \dots, n\}$. 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 equation 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 equation 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 equation 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 equation 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 equation 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 equation 3, we have $s_k \overset{kr}{\rightarrow} e(k)$ 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 equation 3 (which is possible since they are distinct and both different to i and j), we also have that $s_r \overset{rk}{\rightarrow} e(r)$. 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 illustrate the power of SS, CNS and SCSS using the examples given in Figure 1 and Figure 2. In Figure 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 Figure 3), since $(1, 1) \in R_{12}$ and $0 \overset{23}{\rightarrow} 1$; and $0 \overset{14}{\rightsquigarrow} 1$ since $0 \overset{14}{\rightarrow} 1$. Indeed, by a similar argument, the value 0 is snake substitutable by 1 in each domain. In Figure 1(b),

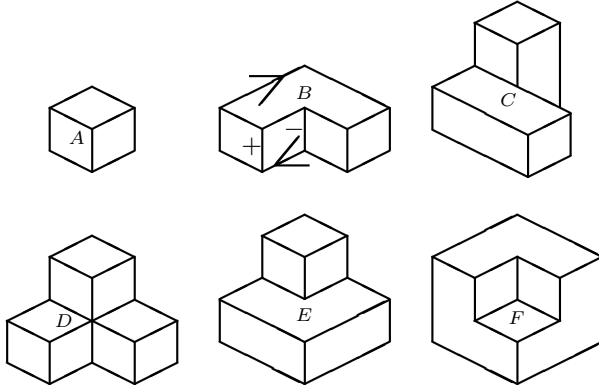


Figure 6: The six different types of trihedral vertices: A , B , C , D , E , F .

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.

In Figure 2, 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 $3 \xrightarrow{13} 1$ (taking $e(1, 3, 1, d) = 3$ for $d = 0, 1, 2$) and $3 \xrightarrow{14} 1$ (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 $3 \xrightarrow{13} 2$ (again taking $e(1, 3, 2, d) = 3$ for $d = 0, 1, 2$) and $3 \xrightarrow{14} 2$ (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 $\mathcal{D}(x_1)$.

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 [4, 18, 28]. There are six types of trihedral vertices: A , B , C , D , E and F , shown in Figure 6. 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 Figure 6 have been labelled to illustrate the meaning of these labels. This problem can be expressed as

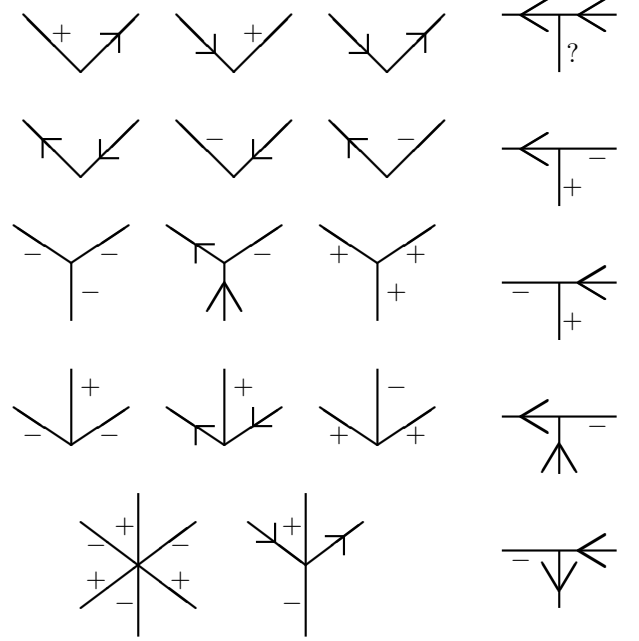


Figure 7: The catalogue of labelled junctions that are projections of trihedral vertices.

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 [4, 18]. 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]. The complete catalogue of labelled junctions is shown in Figure 7, where a question mark represents any of the four labels and rotationally symmetric labellings are omitted. 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 Figure 8, 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 Figure 8 is ambiguous. For example, any of lines AB , BC or CD could be projections of concave edges (meaning that the two blocks on the left side 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 Figure 8 is an exam-

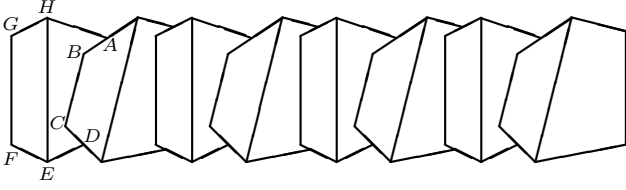


Figure 8: An example from a family of line drawings whose exponential number of labellings is reduced to one by snake substitution.

ple 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 this 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 for this family of line drawings.

Of course, there are line drawings where snake substitution is much less effective than in Figure 8. Nevertheless, in the six drawings in Figure 6, 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 denote the number of pairs $\{i, j\}$ such that x_i, x_j constrain each other. We use d to denote the maximum size of the domains $\mathcal{D}(x_i)$.

Using standard data structures and techniques inspired by efficient algorithms for arc consistency and neighbourhood substitution, we obtain the complexities for CNS and SCSS, as given below. Proofs are omitted due to space limitations.

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

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

Thus, the complexity of applying the value-elimination rules CNS and SS is comparable to the $O(ed^3)$ time complexity of applying neighbourhood substitution (NS) [6]. This is interesting because (in instances with more than one variable) CNS and SS both strictly subsume NS.

Theorem 4 *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 $O(ed^5)$ time and $O(ed^2)$ space.*

6 Optimal sequences of eliminations

It is known that applying different sequences of neighbourhood operations until convergence produces isomorphic instances [6]. 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 5 *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 [19], the well-known NP-complete problem which, given sets $S_1, \dots, S_m \subseteq U$ and an integer k , consists in determining whether there are k sets S_{i_1}, \dots, S_{i_k} which cover U (i.e. such that $S_{i_1} \cup \dots \cup S_{i_k} = U$). We can assume that $S_1 \cup \dots \cup S_m = U$ and $k < m$, otherwise the problem is trivially solvable. Given sets $S_1, \dots, S_m \subseteq U$, we create a 2-variable CSP instance with $\mathcal{D}(x_1) = \{1, \dots, m\}$, $\mathcal{D}(x_2) = U$ 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 $S_1, \dots, S_{i-1}, S_{i+1}, \dots, S_m$ 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 $\{1, \dots, m\}$, 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 6 *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 [15]. Consider an instance I_{2SAT} of MAX 2-SAT with variables X_1, \dots, 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, \dots, 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 Figure 9 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 Figure 10 simulates the relationship between a MAX 2-SAT variable X and its negation \bar{X} .
3. the construction in Figure 11 allows us to create multiple copies of a MAX 2-SAT literal X .
4. the construction in Figure 12 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.

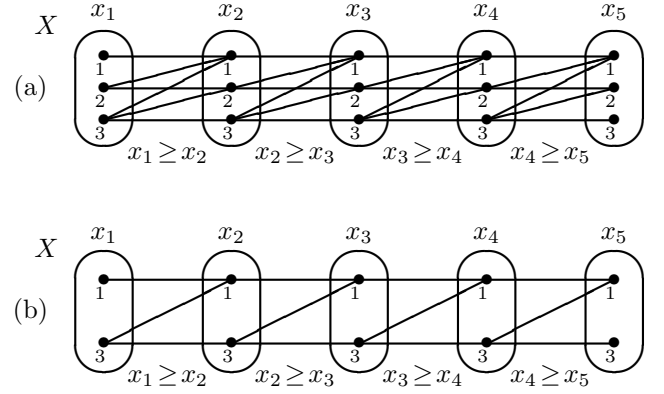


Figure 9: A construction to simulate a MAX 2-SAT variable X : (a) $X = 0$, (b) $X = 1$.

Figure 9(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 Figure 9(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 Figure 9(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 $X = 1$.

The construction in Figure 10 joins the two path-of-CSP-variables constructions corresponding to the literals X and \bar{X} . This construction ensures that exactly one of X and \bar{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 $2 \overset{\bar{x}_0}{\rightsquigarrow} 3$, but that after elimination of the value 2 from either of $\mathcal{D}(x_0)$ or $\mathcal{D}(\bar{x}_0)$, 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, \dots) corresponding to X , as shown in Figure 9(b). By a symmet-

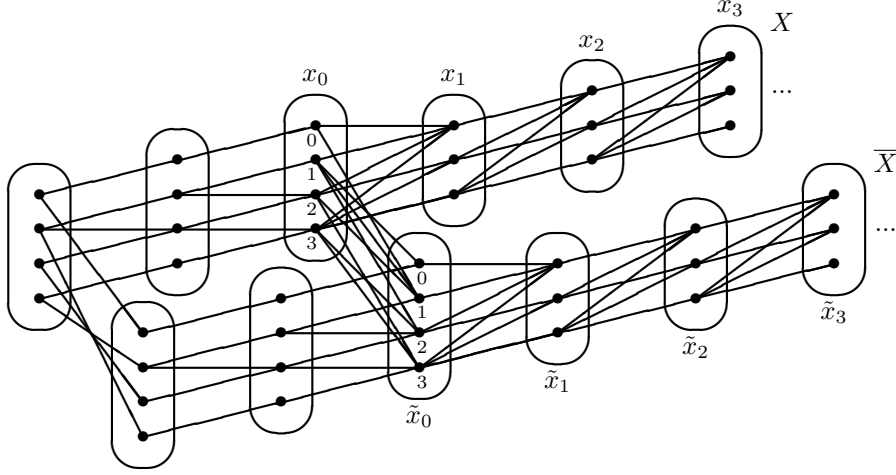


Figure 10: A construction to simulate a MAX 2-SAT variable X and its negation \bar{X} .

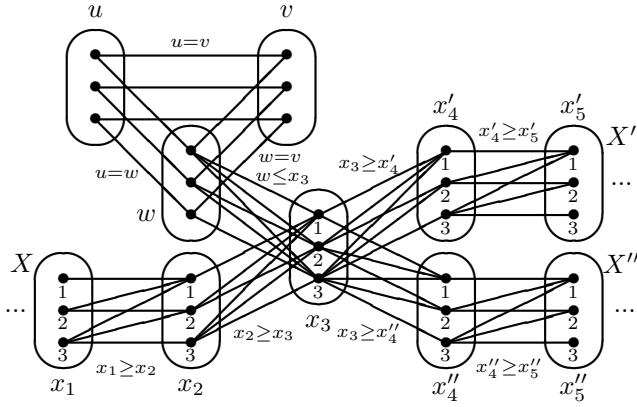


Figure 11: A construction to create two copies X' and X'' of the MAX 2-SAT variable X .

rical argument, if 2 is eliminated from $\mathcal{D}(\tilde{x}_0)$, then this elimination propagates along the path of CSP variables $(\tilde{x}_1, \tilde{x}_2, \tilde{x}_3, \dots)$ corresponding to \bar{X} . Thus, this construction simulates the assignment of a truth value to X and its complement to \bar{X} .

Since any literal of I_{2SAT} may occur in several clauses, we need to be able to make copies of any literal. Figure 11 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

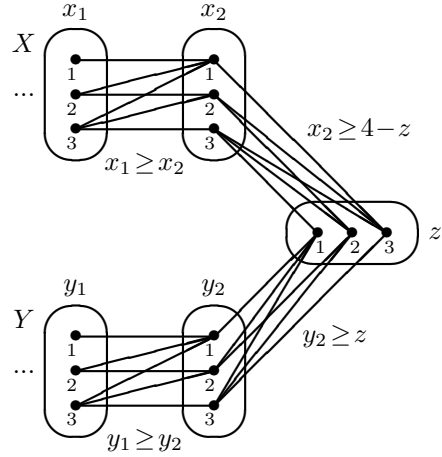


Figure 12: A construction to simulate a MAX 2-SAT clause $X \vee Y$.

Figure 9: 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 Figure 9. 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 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 Figure 12 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 Figure 9. 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 Figure 11) 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. ■

7 Discussion and Conclusion

We have given two different value-elimination rules, namely snake substitutability (SS) and conditioned neighbourhood substitutability (CNS), which 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 Figures 1 and 2 show that these three rules are strictly stronger than neighborhood substitution and that SS and CNS are incomparable. We found snake substitution to be particularly effective when applied to the problem of labelling line-drawings of polyhedral scenes.

Further research is required to investigate generalisations of SS, CNS or SCSS to non-binary or even global constraints. Another obvious avenue of research is the generalisation to valued CSPs (also known as cost-function networks). It is known that the generalisation of neighbourhood substitution to binary valued CSPs [16, 21] can be applied to convergence in $O(ed^4)$ time if the aggregation operator is strictly monotonic or idempotent [7]. 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 (which is a particular example of a strictly monotonic operator) [12]. However, further research is required to determine the complexity of applying this operation until convergence.

It is known that it is possible to efficiently find all (or a given number of) solutions to a CSP after applying neighbourhood substitution: given the set of all solutions to the reduced instance, it is possible to reconstruct $K \geq 1$ solutions to the original instance I (or to determine that I does not have K solutions) in $O(K(de + n^2))$ time [6]. This also holds for the case of conditioned neighbourhood substitution, since, as for neighbourhood substitution, 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 snake value-elimination rule (which is a weaker version of SS) [5], 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.

This work begs the interesting theoretical question as to the existence of reduction operations which strengthen other known reduction operations without increasing complexity.

References

- [1] Bessiere, C.: Constraint propagation. In: Rossi et al. [27], pp. 29–83
- [2] Bessière, C., Debruyne, R.: Optimal and sub-optimal singleton arc consistency algorithms. In: Kaelbling, L.P., Saffioti, A. (eds.) IJCAI-05. pp. 54–59. Professional Book Center (2005)
- [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)
- [4] Clowes, M.B.: On seeing things. *Artif. Intell.* **2**(1), 79–116 (1971)
- [5] 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)
- [6] Cooper, M.C.: Fundamental properties of neighbourhood substitution in constraint satisfaction problems. *Artif. Intell.* **90**(1-2), 1–24 (1997)

- [7] Cooper, M.C.: Reduction operations in fuzzy or valued constraint satisfaction. *Fuzzy Sets and Systems* **134**(3), 311–342 (2003)
- [8] Cooper, M.C.: Constraints between distant lines in the labelling of line drawings of polyhedral scenes. *International Journal of Computer Vision* **73**(2), 195–212 (2007)
- [9] Cooper, M.C.: *Line Drawing Interpretation*. Springer (2008)
- [10] Cooper, M.C., Duchain, 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)
- [11] Cooper, M.C., El Mouelhi, A., Terrioux, C.: Extending Broken Triangles and Enhanced Value-Merging. In: Rueher, M. (ed.) *CP 2016. Lecture Notes in Computer Science*, vol. 9892, pp. 173–188. Springer (2016)
- [12] Cooper, M.C., Jguirim, W., Cohen, D.A.: Domain reduction for valued constraints by generalising methods from CSP. In: Hooker, J.N. (ed.) *CP 2018. Lecture Notes in Computer Science*, vol. 11008, pp. 64–80. Springer (2018)
- [13] Freuder, E.C.: Eliminating interchangeable values in constraint satisfaction problems. In: Dean, T.L., McKeown, K.R. (eds.) *Proc. 9th National Conference on Artificial Intelligence*, Vol. 1. pp. 227–233. AAAI Press / The MIT Press (1991)
- [14] 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. EPIc Series in Computing*, vol. 50, pp. 51–63. EasyChair (2017)
- [15] Garey, M.R., Johnson, D.S., Stockmeyer, L.J.: Some simplified np-complete graph problems. *Theor. Comput. Sci.* **1**(3), 237–267 (1976)
- [16] Goldstein, R.F.: Efficient rotamer elimination applied to protein side-chains and related spin glasses. *Biophysical Journal* **66**(5), 1335–1340 (1994)
- [17] van Hoeve, W., Katriel, I.: Global constraints. In: Rossi et al. [27], pp. 169–208
- [18] 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)
- [19] Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) *Proceedings symposium on the Complexity of Computer Computations*. pp. 85–103. Plenum Press, New York (1972)
- [20] Lecoutre, C.: *Constraint Networks Techniques and Algorithms*. ISTE/Wiley (2009)
- [21] Lecoutre, C., Roussel, O., Dehani, D.E.: WCSP integration of soft neighborhood substitutability. In: Milano [24], pp. 406–421
- [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 '13*. pp. 86–91. ACM (2013)
- [23] Mackworth, A.K.: Consistency in networks of relations. *Artif. Intell.* **8**(1), 99–118 (1977)
- [24] Milano, M. (ed.): *CP 2012, Lecture Notes in Computer Science*, vol. 7514. Springer (2012)
- [25] Mohr, R., Henderson, T.C.: Arc and path consistency revisited. *Artif. Intell.* **28**(2), 225–233 (1986)
- [26] Naanaa, W.: New schemes for simplifying binary constraint satisfaction problems. <https://hal.archives-ouvertes.fr/hal-01731250> (2018)
- [27] Rossi, F., van Beek, P., Walsh, T. (eds.): *Handbook of Constraint Programming, Foundations of Artificial Intelligence*, vol. 2. Elsevier (2006)
- [28] Waltz, D.: Understanding line drawings of scenes with shadows. In: Winston, P.H. (ed.) *The Psychology of Computer Vision*, pp. 19–91. Computer Science Series, McGraw-Hill (1975)
- [29] Woodward, R.J., Karakashian, S., Choueiry, B.Y., Bessiere, C.: Revisiting neighborhood inverse consistency on binary CSPs. In: Milano [24], pp. 688–703