ELSEVIER

Artificial Intelligence 90 (1997) 1–24

Artificial Intelligence

Fundamental properties of neighbourhood substitution in constraint satisfaction problems

Martin C. Cooper*

IRIT, Université Paul Sabatier, 31062 Toulouse, France

Received December 1994; revised September 1995

Abstract

In combinatorial problems it is often worthwhile simplifying the problem, using operations such as consistency, before embarking on an exhaustive search for solutions. Neighbourhood substitution is such a simplification operation. Whenever a value x for a variable is such that it can be replaced in all constraints by another value y, then x is eliminated.

This paper shows that neighbourhood substitutions are important whether the aim is to find one or all solutions. It is proved that the result of a convergent sequence of neighbourhood substitutions is invariant modulo isomorphism. An efficient algorithm is given to find such a sequence. It is also shown that to combine consistency (of any order) and neighbourhood substitution, we only need to establish consistency once.

1. Neighbourhood substitution

In a CSP (constraint satisfaction problem) on n variables, constraints are given in the form of relations $C(P_1), \ldots, C(P_r)$ on subsets $P_1, \ldots, P_r$ of the n variables. $C(P_j)$ is the set of legal labellings for the set of variables P_j. It may be given as an explicit list of labellings or in implicit closed form. The set of n-tuples satisfying all the constraints is simply the join of the constraints $C(P_1), \ldots, C(P_r)$. The domain of variable i is denoted by A_i. Many problems, such as school-timetabling, scheduling, line-drawing labelling, sketch-map interpretation and circuit design can be expressed in a natural way as CSPs. For example, in the graph colouring problem, in which the aim is to assign a colour to each node of a given graph G so that no two adjacent nodes are assigned the same colour, there is a binary constraint

* E-mail: cooper@irit.fr.

PII S0004-3702(96)00018-5

$$C(\{u, v\}) = \{(x, y): x, y \text{ different colours}\},$$

for each edge (u, v) in G. In the case of binary constraints, we use the shorthand C_{uv} for $C(\{u, v\})$.

Unfortunately, determining whether a constraint satisfaction problem has at least one solution is NP-complete [8]. Consistency is a well-known operation on CSPs which renders the information in the constraints more explicit: tuples are eliminated from constraints when it is discovered that they cannot be part of any globally consistent labelling. In general, the more explicit the constraints are, the less search is required to find one or all solutions. Arc consistency is very effective in certain problems, such as line-drawing labelling [12] and scheduling. However, in other problems, one example being the 8-queens problem, arc consistency produces no eliminations and so does not reduce the combinatorial explosion.

Freuder [5] defined a new reduction operation for binary CSPs: label $a \in A_i$ is neighbourhood substitutable for label $b \in A_i$ at variable i if for all constraints C_{ij}, and for all values $x \in A_j$,

$$(b, x) \in C_{ij} \Rightarrow (a, x) \in C_{ij}.$$

A straightforward generalisation to higher-order constraints is given in the following section. If the label b is eliminated, then the reduced CSP will have a set of solutions which may be a proper subset of the set of solutions to the original CSP. However, an important property is preserved: the reduced CSP has a solution iff the original CSP has a solution. Thus substitution operations are clearly useful when searching for a single solution. We will show in Section 6 that they are, in fact, equally useful when searching for all solutions. Local substitution operations, such as neighbourhood substitution, propagate in the same way that local consistency operations propagate [7].

Substitution operations can be applied to certain CSPs in which consistency would be of no help. Tsang [11] points out that consistency is useful in highly constrained problems but much less so in loosely constrained problems. We would hope to be able to profitably apply substitution in both types of problems, but especially in problems with a sparse constraint graph.

As an example, consider the graph colouring problem of Fig. 1(a). Here the constraint graph is a 5 by 5 grid. The list of possible colours for each node is given in the form of the initial letters of the colours. Thus, for example, the set of possible labels for the top right-hand node is {*red, green, blue*}. The problem is to colour each node, so that adjacent nodes are assigned different colours. This CSP is loosely constrained and arc consistency produces no eliminations of labels. Neighbourhood substitution, on the other hand, produces many eliminations. Consider the top right-hand node. The label *green* is neighbourhood substitutable for both *red* and *blue*, since *green* is consistent with all labels at the two adjacent nodes. Thus both *red* and *blue* can be eliminated at this node by neighbourhood substitution.

Unlike consistency operations, applying substitution operations until convergence (i.e. until no more eliminations are possible) does not always produce the

(a)

gy	rgb	yr	by	rgb
gb	ry	rb	gb	ry
rb	by	gy	byr	gb
byg	rb	gy	rb	gr
gy	gb	by	gr	by

(b)

y	g	y	b	g
g	y	r	g	y
r	b	gy	byr	gb
b	r	gy	rb	gr
y	g	b	g	b

(c)

g	b	y	b	r
b	y	b	g	y
r	b	gy	byr	gb
b	r	gy	rb	gr
y	g	b	g	y

Fig. 1. The results of applying two different convergent sequences of neighbourhood substitutions to the graph colouring problem in (a) are shown in (b) and (c).

same result. Two distinct convergent sequences of neighbourhood substitution operations applied to the problem of Fig. 1(a) produce the two distinct graph colouring problems shown in Figs. 1(b) and 1(c). In both cases, however, the reduced CSP is clearly easier to solve than the original CSP, since most variables are left with only one possible label.

It should be mentioned that what makes neighbourhood substitution so effective in this particular pseudo-random graph colouring problem is the fact that the domains A_i for each variable i are distinct. In a colouring problem on a connected graph, with identical domains A_i for all i, no neighbourhood substitutions are possible.

It is clear that the two reduced problems in Figs. 1(b) and 1(c) are very similar. Let CSP(b) and CSP(c) denote these two problems, and let A_i(b) and A_i(c) represent the domains in CSP(b) and CSP(c). The constraints C_{ij} are identical in the two problems. CSP(b) and CSP(c) are isomorphic, in the sense that, for each node i, there is a bijection

$$f_i: A_i(\mathrm{b}) \rightarrow A_i(\mathrm{c})$$

satisfying

$$(x, y) \in C_{ij} \Leftrightarrow (f_i(x), f_i(y)) \in C_{ij}.$$

We will show in the following section that two convergent sequences of neighbourhood substitutions applied to two copies of the same CSP always produce isomorphic CSPs. The definition of an isomorphism between arbitrary CSPs is given below (Definition 2.4).

2. Invariance of the result of neighbourhood substitutions

To simplify the notation in the following proofs, we assume that labels at different nodes are distinct. When we say that x is a label, we mean implicitly that x is a (label, node) pair (a_x, i_x) such that $a_x \in A_{i_x}$. This should not lead to any confusion, since the substitution of x by y is only possible if $i_x = i_y$. It simply avoids having to specify the node for each label.

Definition 2.1. Given two labels $x, y \in A_i$ for variable i, the label y is *neighbourhood substitutable* for x if for all constraints $C(P)$ such that $i \in P = \{i, i_1, \ldots, i_{k-1}\}$

$$\{(z_1, \ldots, z_{k-1}) \in A_{i_1} \times \cdots \times A_{i_{k-1}}: (x, z_1, \ldots, z_{k-1}) \in C(P)\},$$

$$\subseteq \{(z_1, \ldots, z_{k-1}) \in A_{i_1} \times \cdots \times A_{i_{k-1}}: (y, z_1, \ldots, z_{k-1}) \in C(P)\}.$$

We use the notation $x \rightarrow y$ to represent the corresponding neighbourhood substitution operation, which can be read as "x is eliminated because it can be substituted by y".

An algorithm which eliminates substitutable labels may also update constraints $C(P)$ by eliminating all labellings $(x, z_1, \ldots, z_{k-1})$ when the label x is eliminated. However, such eliminations are superfluous, in the sense that they cannot create new neighbourhood substitutions. This is because, in Definition 2.1, we never consider tuples $(x, z_1, \ldots, z_{k-1})$ in which some z_i or x has been eliminated from the corresponding domain. Furthermore, in many constraint satisfaction problems, constraints are given implicitly in the form of equations or inequalities, such as $v_i > v_j + v_k$. Such constraints do not easily lend themselves to updating. For these reasons we assume that neighbourhood substitution operations update only the domains A_i and not the constraints $C(P)$.

Let E be a sequence $(e_1, e_2, \ldots, e_r)$ of neighbourhood substitutions applied to a CSP. Each neighbourhood substitution e_k has the form $x \rightarrow y$, for some labels x and y. We say that E converges if no more neighbourhood substitutions are possible in the CSP resulting from the application of the sequence of substitutions E. As an example, consider the CSP shown in Fig. 2(a), consisting of three variables and two constraints C_{12} and C_{23}. A line joins label $x \in A_i$ and $y \in A_j$ if $(x, y) \in C_{ij}$. There are three convergent sequences of neighbourhood substitutions:

(1) $(e \rightarrow d, b \rightarrow a)$,
(2) $(b \rightarrow a, e \rightarrow d)$,
(3) $(b \rightarrow a, d \rightarrow e)$.

The resulting CSPs are shown in Fig. 2(b) for sequences (1) and (2), and in Fig. 2(c) for sequence (3). Although the CSPs are different, they are clearly isomorphic.

If there exist labels $a_1, a_2, \ldots, a_{k-1}$ such that E contains as a subsequence

$$(e_{i_1}, e_{i_2}, \ldots, e_{i_k}) = (x \rightarrow a_1, a_1 \rightarrow a_2, \ldots, a_{k-2} \rightarrow a_{k-1}, a_{k-1} \rightarrow y),$$

then we will write as a shorthand "$x \rightarrow \cdots y$ in E", or more explicitly

$$x \rightarrow a_1 \rightarrow a_2 \rightarrow \cdots \rightarrow a_{k-1} \rightarrow y.$$

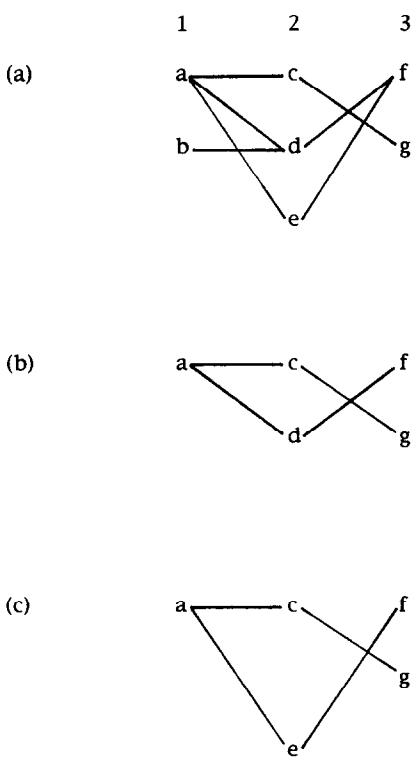

Fig. 2. (a) The consistency graph of a 3-variable CSP; (b), (c) two isomorphic CSPs which result from applying different convergent sequences of neighbourhood substitution operations to this CSP.

Note that E will often contain substitutions at different nodes, but that x, $a_1, \ldots, a_{k-1}$, y must all be labels for the same node. For example, if the CSP contains the single constraint shown in Fig. 3(a), then $E=(f \to d, c \to b, b \to a, d \to e)$ is a convergent sequence of neighbourhood substitutions which results in the CSP shown in Fig. 3(b). In E, $f \to \cdots e$ and $c \to \cdots a$ (subsequences of length 2), as well as $f \to \cdots d$, $c \to \cdots b$, $b \to \cdots a$ and $d \to \cdots e$ (subsequences of length 1). We consider $x \to \cdots x$ to be true for all x; this corresponds to a sequence of length 0.

We write $x \mapsto \cdots y$ in E if $x \to \cdots y$ in E and y is not eliminated in E.

Lemma 2.2. *Let $C(P)$ be a constraint in a CSP and E a sequence of neighbourhood substitutions that can be applied to the CSP. If $(u_1, \ldots, u_k) \in C(P)$ and $u_i \mapsto \cdots x_i$ in E, for $i=1, \ldots, k$, then $(x_1, \ldots, x_k) \in C(P)$.*

Proof. Let E_j denote the prefix $(e_1, \ldots, e_j)$ of $E=(e_1, \ldots, e_m)$, with E_0 defined, by default, as the empty sequence. Consider as inductive hypothesis H_j:

if $(u_1, \ldots, u_k) \in C(P)$ and $u_i \mapsto \cdots x_i$ in E_j, for $i = 1, \ldots, k$,

then $(x_1, \ldots, x_k) \in C(P)$.

H_0 is clearly true, since $u_i \mapsto \cdots u_i$ in E_0, the empty sequence. We will prove, by contradiction, that $H_j \Rightarrow H_{j+1}$. The only way that H_j could be true and H_{j+1} false is if, for some $(u_1, \ldots, u_k) \in C(P)$ such that $u_i \mapsto \cdots x_i$ in E_j, for $i=1, \ldots, k$,

$$(x_1, \ldots, x_{h-1}, x_h, x_{h+1}, \ldots, x_k) \in C(P)$$

and

$$(x_1, \ldots, x_{h-1}, x'_h, x_{h+1}, \ldots, x_k) \notin C(P),$$

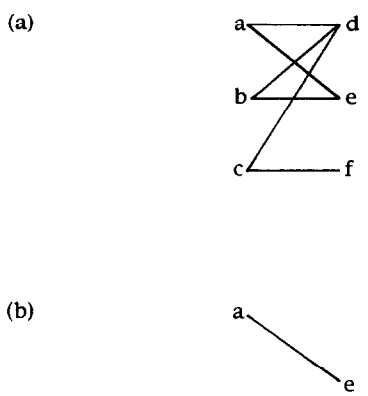

Fig. 3. (a) The consistency graph of a single-constraint CSP; (b) the result of the sequence of neighbourhood substitutions $(f \to d, c \to b, b \to a, d \to e)$.

where e_{j+1} is the substitution $x_h \mapsto \cdots x'_h$. This contradicts Definition 2.1 of neighbourhood substitution, given our assumption that none of the labels x_i, $(i=1,\ldots,k)$ have been eliminated from their domains in E_j. Therefore, $H_j \Rightarrow H_{j+1}$, and by induction we can deduce H_m, which is exactly the result we set out to prove, since $E_m = E$. □

The lemma would become false if $u_i \mapsto \cdots x_i$ were to be replaced by $u_i \rightarrow \cdots x_i$. For example, in the sequence of substitutions

$$E = (f \rightarrow d, c \rightarrow b, b \rightarrow a, d \rightarrow e)$$

that can be applied to the CSP of Fig. 3(a), we have $c \rightarrow \cdots b$ and $f \rightarrow \cdots f$. However, denoting the single constraint of this CSP by C_{12}, $(c, f) \in C_{12}$ does not imply $(b, f) \in C_{12}$. This is due to the fact, that when b is substitutable for c, f has already been eliminated from the domain A_2.

Let $x \rightarrow y$ be a substitution which can be applied to a CSP. Consider a convergent sequence F of neighbourhood substitutions applied to the same CSP. Since extra eliminations of labels cannot prevent neighbourhood substitutions, we can deduce that if x is not eliminated in F, then y must be. The following lemma extends this result by considering a subsequence $x \rightarrow \cdots y$ of eliminations within a sequence E, instead of a single substitution $x \rightarrow y$, and shows not only that y is eliminated but also that $y \mapsto \cdots x$ in F.

Lemma 2.3. *Let E and F be two sequences of neighbourhood substitutions for the same CSP, and suppose that F is convergent. If $x \rightarrow \cdots y$ in E and x is not eliminated in F, then $y \mapsto \cdots x$ in F.*

Proof. If $x \rightarrow \cdots \rightarrow z \rightarrow y$ in E where $z \rightarrow y$ is the pth substitution in the sequence E, then we say that the subsequence $x \rightarrow \cdots y$ ends at position p. We prove the lemma by induction on p. This gives the following inductive hypothesis.

H_p: Let E and F be two sequences of neighbourhood substitutions for the same CSP, where F is convergent. If $x \rightarrow \cdots y$ in E_p, the prefix of E ending at position p, and x is not eliminated in F, then $y \mapsto \cdots x$ in F.

H_0 is trivially true, since $x \rightarrow \cdots x$ in E_0 and F for all x.

Suppose that H_{p-1} is true, where $p \geq 1$. Consider a subsequence of E, $x \rightarrow \cdots \rightarrow z \rightarrow y$, which ends at position p, where x is not eliminated in the convergent sequence F. To demonstrate the truth of H_p, we must show that $y \mapsto \cdots x$ in F. Suppose that $y \mapsto \cdots v$ in F. Note that we do not discount the possibility that $v = y$.

Consider $(x, z_1, \ldots, z_{k-1}) \in C(P)$. We will show that, if none of $z_1, \ldots, z_{k-1}$ are eliminated in F, then $(v, z_1, \ldots, z_{k-1}) \in C(P)$.

Suppose that, for each $t=1,\ldots,k-1$, $z_t \mapsto \cdots w_t$ in E_p, the prefix of E ending at position p (the position at which $z \rightarrow y$). We do not exclude the possibility that

$w_t = z_t$. Since $(x, z_1, \ldots, z_{k-1}) \in C(P)$, $x \mapsto \cdots y$ in E_p and $z_t \mapsto \cdots w_t$ in E_p, we can deduce by Lemma 2.2 that

$$(y, w_1, \ldots, w_{k-1}) \in C(P). \tag{1}$$

This is illustrated in the left-hand side of Fig. 4 in the case of binary constraints. Vertical arrows represent subsequences of neighbourhood substitutions: $x \mapsto \cdots y$ and $z_1 \mapsto \cdots w_1$ in E_p; horizontal lines represent constraint membership:

$$(x, z_1), (y, w_1) \in C(P).$$

Each $z_t \to \cdots w_t$ is a subsequence of E which ends *before* position p. This is because $z \to y$ is the substitution at position p, and w_t cannot be equal to y since they are labels for different nodes. By our inductive hypothesis, applied to each of $z_t \to \cdots w_t$ in E_{p-1}, we deduce that *either*

(a) at least one of $z_1, \ldots, z_{k-1}$ is eliminated in F, *or*

(b) none of $z_1, \ldots, z_{k-1}$ is eliminated in F and, for each $t = 1, \ldots, k-1$, $w_t \mapsto \cdots z_t$ in F.

Consider case (b). $(y, w_1, \ldots, w_{k-1}) \in C(P)$ by (1), $y \mapsto \cdots v$ in F by definition of v, and $w_t \mapsto \cdots z_t$ in F for each $t = 1, \ldots, k-1$. Therefore, by Lemma 2.2,

$$(v, z_1, \ldots, z_{k-1}) \in C(P).$$

This is illustrated in the right-hand side of Fig. 4. Thus we have shown that if $(x, z_1, \ldots, z_{k-1}) \in C(P)$ and none of $z_1, \ldots, z_{k-1}$ is eliminated in F, then $(v, z_1, \ldots, z_{k-1}) \in C(P)$.

Hence, in the CSP which results from the sequence of substitutions F,

$$(\text{none of } z_1, \ldots, z_{k-1} \text{ eliminated}) \wedge (x, z_1, \ldots, z_{k-1}) \in C(P) \Rightarrow$$
$$(v, z_1, \ldots, z_{k-1}) \in C(P).$$

This implies, from Definition 2.1 of neighbourhood substitution, that x could be substituted by v, which contradicts the fact that F is convergent, *unless* $v = x$.

Since $y \mapsto \cdots v$, in F by definition of v, we have thus shown exactly what was required, namely

$$y \mapsto \cdots x \text{ in } F,$$

which completes our proof by induction. □

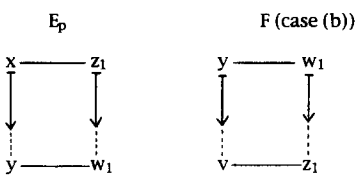

Fig. 4. In the case of binary constraints, every z_1 such that $(x, z_1) \in C(P)$, satisfies $(v, z_1) \in C(P)$ provided z_1 is not eliminated in F.

Definition 2.4. Consider two CSPs with domains A_i and A'_i $(i=1,\ldots,n)$ and constraints $C(P)$ and $C'(P)$ $(P\subseteq N=\{1,\ldots,n\})$. An *isomorphism* between the two CSPs is a bijection

$$f: \bigcup_{i=1,\ldots,n} A_i \rightarrow \bigcup_{i=1,\ldots,n} A'_i,$$

such that

(1) $\forall i\in\{1,\ldots,n\}\ \forall x\in A_i\ (f(x)\in A'_i)$,

(2) $\forall P\subseteq N\ ((u_1,\ldots,u_k)\in C(P) \Leftrightarrow (f(u_1),\ldots,f(u_k))\in C'(P))$.

If no constraint exists on P in a CSP, then $C(P)$ is simply the complete constraint (the cartesian product of the domains of the variables in P).

In the following theorem the two CSPs have identical constraints but different domains.

Theorem 2.5. *If E and F are two convergent sequences of neighbourhood substitutions for the same CSP, then the CSPs which result from the application of E and F are isomorphic.*

Proof. We use the notation $A_i(E)$ to represent the set of labels for node i not eliminated by E, and $U(E)$ to represent $\bigcup_{i=1,\ldots,n} A_i(E)$.

We define the function f: $U(E)\rightarrow U(F)$ as follows:

$$f(x)=y \quad \text{where } x \mapsto \cdots y \text{ in } F.$$

Let $x\in U(E)$. Since $x\rightarrow\cdots f(x)$ in F and x is not eliminated in E, Lemma 2.3 tells us that $f(x)\mapsto\cdots x$ in E. f is thus injective, since $f(x_1)=f(x_2)=y$ implies that $y\mapsto\cdots x_1$ and $y\mapsto\cdots x_2$ in E, which is clearly only possible if $x_1=x_2$.

Consider $y\in U(F)$, and let $x\in U(E)$ be such that $y\mapsto\cdots x$ in E. Then, by Lemma 2.3, $x\mapsto\cdots y$ in F and hence $f(x)=y$. f is therefore surjective.

Let $u_1,\ldots,u_k\in U(E)$. By definition of f, $u_i\mapsto\cdots f(u_i)$ in F, for each $i=1,\ldots,k$. Therefore, by Lemma 2.2,

$$(u_1,\ldots,u_k)\in C(P)\Rightarrow(f(u_1),\ldots,f(u_k))\in C(P).$$

Now, $f(u_i)\mapsto\cdots u_i$ in E, for each $i=1,\ldots,k$, by Lemma 2.3. Thus, again by Lemma 2.2,

$$(f(u_1),\ldots,f(u_k))\in C(P) \Rightarrow (u_1,\ldots,u_k)\in C(P).$$

Thus f is an isomorphism from $U(E)$ to $U(F)$. In particular, $(x_1,\ldots,x_n)$ is a solution to the CSP resulting from the application of E iff $(f(x_1),\ldots,f(x_n))$ is a solution to the CSP resulting from the application of F. □

The major consequence of this result is that there is no point trying to choose the best convergent sequence of neighbourhood substitutions operations to apply to a CSP since the resulting reduced CSPs are all isomorphic.

3. k-consistency and neighbourhood substitution

We remind the reader of some definitions concerning consistency. For notational convenience, we consider the domains A_i and the order-1 constraints $C(\{i\})$ to be synonyms. A CSP can thus be represented by a set of constraints $\mathscr{C} = \{C(P): P \subseteq N\}$, without mention of the domains.

Given a labelling $X_Q = (x_{i_1}, \ldots, x_{i_q})$ for a set of variables $Q = \{i_1, \ldots, i_q\} \subseteq N$, and a subset $P = (j_1, \ldots, j_p) \subseteq Q$, we use the notation $\Pi_P X_Q$ to denote $X_P = (x_{j_1}, \ldots, x_{j_p})$. We say that X_Q is an extension of X_P.

Definition 3.1. A labelling X_Q, for a set of variables $Q \subseteq N$ of size k, *k-satisfies* the set of constraints $\mathscr{C} = \{C(P): P \subseteq N\}$ if, for each $P \subseteq Q$, $\Pi_P X_Q \in C(P)$.

Definition 3.2. A set of constraints $\mathscr{D} = \{D(P): P \subseteq N\}$ is a *k-solution* to a CSP $\mathscr{C} = \{C(P): P \subseteq N\}$ if

(1) $P \subseteq N$ such that $1 \leq |P| \leq k$, $D(P) \subseteq C(P)$.

(2) $(X_P \in D(P)) \wedge (P \subseteq Q \subseteq N) \wedge (|Q| = k) \Rightarrow$ there exists an extension X_Q of X_P such that X_Q k-satisfies $\mathscr{D}$.

(3) There does not exist a set of constraints $\mathscr{E} = \{E(P): P \subseteq N\}$ satisfying properties (1) and (2) and such that $D(P) \subset E(P)$ for some $P \subseteq N$.

A k-solution to a CSP is strong k-consistent and is unique [3].

The result which is proved in this section is that if we wish to apply both k-consistency and neighbourhood substitution operations to a CSP, then the best strategy is to

(A) establish strong k-consistency, by finding the k-solution to the original CSP, and then

(B) apply neighbourhood substitution operations until convergence.

This strategy will be shown to be optimal, in that any other sequence of k-consistency and neighbourhood substitution operations produces a CSP which is, at best, isomorphic to the result of executing (A) and then (B), above. At worst, the CSP produced is less tightly constrained.

This strategy is valid for all values of k, and, in particular for arc consistency (2-consistency) [2, 9] and path consistency (3-consistency) [6, 10].

The interaction between consistency and neighbourhood substitution includes certain subtleties as illustrated by the following examples. The following example shows that applying neighbourhood substitution operations until convergence and then establishing strong k-consistency, in other words inverting the order of (A) and (B) above, does not always produce as many eliminations.

Example 3.3. Consider the CSP in Fig. 5(a). No neighbourhood substitutions are possible in this CSP. The arc consistent version of this CSP is shown in Fig. 5(b). This is thus the result of executing operations (B) and then (A), with $k = 2$. However, by first establishing arc consistency, we can then apply the following

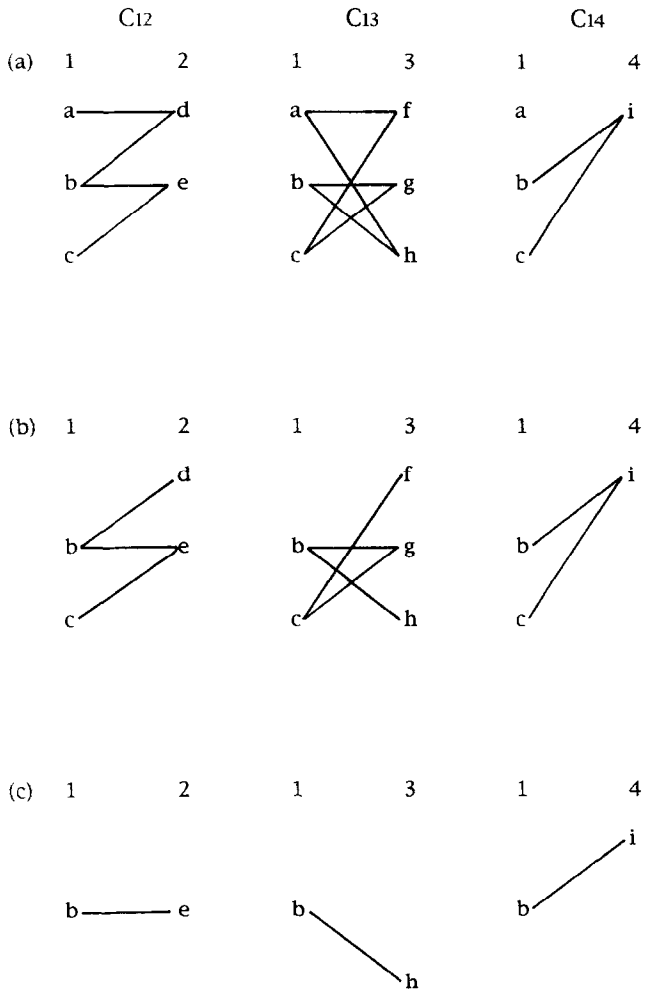

Fig. 5. (a) A 3-constraint CSP on 4 variables; (b) the result of applying arc consistency; (c) the result of then applying neighbourhood substitutions.

convergent sequence of neighbourhood substitutions: $d \rightarrow e$, $f \rightarrow g$, $c \rightarrow b$, $g \rightarrow h$. The resulting CSP is given in Fig. 5(c).

Establishing strong k-consistency is an operation which does not have a unique result. For example, if a labelling (a, b) is eliminated from $C(\{i, j\})$, then strong k-consistency does not impose the elimination of (a, b, x) from $C(\{i, j, k\})$ for all $x \in A_k$. These eliminations are optional. In a k-solution (for $k \geq 3$), these eliminations are mandatory. Such eliminations are redundant in terms of future deletions by consistency operations, but can be useful in allowing neighbourhood substitutions which would otherwise be blocked by the presence of (a, b, x) in $C(\{i, j, k\})$.

Example 3.4. Consider the strong 3-consistent CSP composed of the following constraints

C_{123}:

1	2	3
a	b	c
a	b	d
e	***b***	***d***
e	***g***	***c***
e	g	h

C_{12}:

1	2
a	b
e	g

C_{23}:

2	3
b	c
b	d
g	h

C_{13}:

1	3
a	c
a	d
e	h

$A_1 = \{a, e\}$, $A_2 = \{b, g\}$, $A_3 = \{c, d, h\}$.

The labellings (e, b, d) and (e, g, c) in C_{123} are shown in bold italics since they are superfluous; they are not present in the corresponding 3-solution. The presence of the labelling (e, g, c) in C_{123} blocks the neighbourhood substitution $c \rightarrow d$ at variable 3, and the presence of the labelling (e, b, d) blocks the neighbourhood substitution $d \rightarrow c$. In other words, the elimination of (e, b, d) and (e, g, c) from C_{123} allows us to eliminate either c or d from A_3 by neighbourhood substitution. The resulting reduced CSP has only two solutions compared to the three solutions of the original CSP, shown above.

Examples 3.3 and 3.4 show that new neighbourhood substitutions may be induced by establishing strong k-consistency and even more may be induced by finding a k-solution. The heart of the proof of the main result of this section consists in showing that finding a k-solution cannot invalidate eliminations by neighbourhood substitution and that a strong k-consistent CSP remains strong k-consistent after neighbourhood substitution eliminations.

In order to prove that finding a k-solution before applying neighbourhood substitution operations is the best strategy, we first need to define the basic consistency operations which are employed to find a k-solution. Only two consistency operations are required [3]:

upward-propagation:
if $(x_1, \ldots, x_r) \notin C(P)$ where $|P| < k$
then for all $i \notin P$, for all $x \in A_i$
  delete $(x_1, \ldots, x_r, x)$ from $C(P \cup \{i\})$

downward-propagation:
if, for some $i \in N$ and some $P \subseteq N$, such that $|P| < k$,
  there is no $x \in A_i$ such that $(x_1, \ldots, x_r, x) \in C(P \cup \{i\})$
then delete $(x_1, \ldots, x_r)$ from $C(P)$

As remarked above, A_i and $C(\{i\})$ are synonyms, which means that all deletions from $C(\{i\})$ automatically apply to A_i. Unlike neighbourhood substitution, the above consistency operations update constraints of order up to k, not just the domains.

It is common practice when establishing arc consistency [2, 9] not to store those binary constraints which were not present in the original CSP, since this saves considerable space and does not incur any loss of the information gained by the propagation of constraints, the constraints C_{ij} which are not stored being just $A_i \times A_j$. To simplify the presentation of our proofs we assume that all binary constraints are stored when finding a 2-solution, although this is clearly not necessary for constraints C_{ij} which are always equal to $A_i \times A_j$.

The neighbourhood substitution operation, corresponding to a substitution $a \rightarrow b$ at node i is given by Definition 2.1:

ns-elimination:
if for some $a,b \in A_i$
  for all constraints $C(P)$ such that $i \in P = \{i, i_1, \ldots, i_r\}$
    $\forall (x_1, \ldots, x_r) \in A_{i_1} \times \cdots \times A_{i_r}$
      $((a, x_1, \ldots, x_r) \in C(P) \Rightarrow (b, x_1, \ldots, x_r) \in C(P))$
then delete a from A_i

Neighbourhood substitution may destroy the property of being a k-solution, but, as the following lemma shows, the crucial property of strong k-consistency is preserved.

Lemma 3.5. *Let CSP_0 be a CSP in which the neighbourhood substitution $a \rightarrow b$ is valid at node i, let CSP_1 be the result of the elimination of label a from A_i, and CSP_2 the result of this elimination and the following updating of all constraints $C(P)$ on sets $P = \{i, i_1, \ldots, i_r\}$ containing i:*

$$C(P) := C(P) \cap (A_i \times A_{i_1} \times \cdots \times A_{i_r}).$$

(a) CSP_1 *and* CSP_2 *are isomorphic.*
(b) CSP_0 *is strong k-consistent* $\Rightarrow$ CSP_1 *is strong k-consistent.*
(c) CSP_0 *is a k-solution* $\Rightarrow$ CSP_2 *is a k-solution.*

Proof. (a) follows immediately from Definition 2.4 of an isomorphism. An isomorphism is a mapping between elements of domains A_j and hence is independent of elements $(x_1, \ldots, x_r)$ of constraints $C(\{i_1, \ldots, i_r\})$ for which some label x_j is not an element of the corresponding domain A_{i_j}.

(b) CSP_1 can fail to be strong k-consistent only if, for some labelling $(y_1, \ldots, y_q) \in C(Q)$, $(a, y_1, \ldots, y_q)$ is the only consistent extension of $(y_1, \ldots, y_q)$ to $\{i\} \cup Q$. However, by Definition 2.1 of neighbourhood substitution, we know that $(b, y_1, \ldots, y_q)$ is another consistent extension of $(y_1, \ldots, y_q)$ to $\{i\} \cup Q$. Therefore, CSP_1 remains strong k-consistent despite the elimination of a from A_i.

(c) CSP_2 is the result of applying all possible upward-propagation operations to CSP_1. It is therefore sufficient to prove that no downward-propagation operations are possible in CSP_2. The elimination of $(a, y_1, \ldots, y_q)$ from $C(\{i\} \cup Q)$ cannot induce the elimination of $(a, y_1, \ldots, y_{j-1}, y_{j+1}, \ldots, y_q)$ from the constraint $C(\{i\} \cup Q - \{j\})$ since this elimination has already been performed. Furthermore, the elimination of $(a, y_1, \ldots, y_q)$ from $C(\{i\} \cup Q)$ cannot induce the elimination of $(y_1, \ldots, y_q)$ from $C(Q)$, by the same argument as in the proof of (b), above. □

Lemma 3.6. *If CSP_A and CSP_B are isomorphic, via the isomorphism f, then*
(1) *CSP_A is strong k-consistent $\Leftrightarrow$ CSP_B is strong k-consistent;*
(2) *the neighbourhood substitution $a \rightarrow b$ is valid at variable i of CSP_A iff the neighbourhood substitution $f(a) \rightarrow f(b)$ is valid at variable i of CSP_B.*

Lemma 3.6 follows immediately from the definitions.

Lemma 3.5 tells us that neighbourhood substitutions cannot destroy strong k-consistency. The following theorem is a much stronger result. We say that a sequence of upward-propagation, downward-propagation and ns-elimination operations is convergent if no more such operations are possible in the resulting CSP. The resulting CSP is necessarily a k-solution.

Theorem 3.7. *Let S be a convergent sequence of k-consistency and neighbourhood substitution operations. The result of applying S is isomorphic to the CSP which results from first finding the k-solution and then applying any convergent sequence of neighbourhood substitutions.*

Proof. Let S' be a copy of the sequence S in which each neighbourhood substitution is preceded by the operation of finding a k-solution. Finding a k-solution is also performed as the final operation in S'. The upward-propagation and downward-propagation operations in the original sequence S can be discarded since they are clearly swallowed up by the new operations of finding a k-solution.

We will now show that all labels eliminated in S are also eliminated in S'. We will later show that all but the first determination of a k-solution in S' are redundant.

S' may contain substitutions $x \rightarrow y$ such that x has already been eliminated in S' by consistency operations. Such substitutions are retained in S', but are considered to have no effect.

We can number the substitutions in S (starting from 1). By CSP_p we refer to the state of the constraints just before the application of the pth substitution in S. By CSP'_p we refer to the state of the constraints just before applying the corresponding substitution in S'. Note that, in both cases, we do not number the consistency operations but only the substitution operations.

Suppose that the pth substitution in S is $a \rightarrow b$ at node i. This means, by Definition 2.1, that in CSP_p

$$\begin{aligned}&\text{for all constraints } C(P) \text{ such that } i \in P = \{i, i_1, \ldots, i_r\}\\ &\quad \forall (x_1, \ldots, x_r) \in A_{i_1} \times \cdots \times A_{i_r}\\ &\qquad (a \in A_i \wedge (a, x_1, \ldots, x_r) \in C(P) \Rightarrow b \in A_i \wedge (b, x_1, \ldots, x_r) \in C(P)).\end{aligned} \tag{2}$$

We show by contradiction that applying consistency operations earlier in S' than in S cannot invalidate neighbourhood substitutions. Let p be the first position at which (2) holds in CSP_p but not in CSP'_p. By this choice of p, all labellings eliminated by neighbourhood substitution in CSP_p are also eliminated in CSP'_p. By the definition of a k-solution, all labellings eliminated by upward-propagation and downward-propagation operations in CSP_p are also eliminated by the same operations in CSP'_p. Indeed, the premises of these propagation rules cannot be invalidated by any *extra* eliminations in CSP'_p, and these rules are applied as soon as possible in S'. We can deduce that there is a constraint $C(P)$ and values $x_1, \ldots, x_r$ such that

$$\begin{aligned}&(a, x_1, \ldots, x_r) \in C(P) \cap (A_i \times A_{i_1} \times \cdots \times A_{i_r}) \quad \text{in } \mathrm{CSP}_p,\\ &(b, x_1, \ldots, x_r) \in C(P) \cap (A_i \times A_{i_1} \times \cdots \times A_{i_r}) \quad \text{in } \mathrm{CSP}_p,\\ &(a, x_1, \ldots, x_r) \in C(P) \cap (A_i \times A_{i_1} \times \cdots \times A_{i_r}) \quad \text{in } \mathrm{CSP}'_p,\\ &(b, x_1, \ldots, x_r) \notin C(P) \cap (A_i \times A_{i_1} \times \cdots \times A_{i_r}) \quad \text{in } \mathrm{CSP}'_p.\end{aligned} \tag{3}$$

This must be because $(b, x_1, \ldots, x_r)$ was eliminated by the extra consistency operations in CSP'_p. Among all such constraints $C(P)$ and values $x_1, \ldots, x_r$, for which (3) holds, let $(b, x_1, \ldots, x_r)$ be the first labelling to have been eliminated by the extra consistency operations in CSP'_p. The case $r=0$ corresponds to the elimination of b from A_i. Let $\mathrm{CSP}'_{\mathrm{ELIM}}$ be the state of the constraints just before the elimination of $(b, x_1, \ldots, x_r)$. There are three possible reasons for the elimination of $(b, x_1, \ldots, x_r)$ in $\mathrm{CSP}'_{\mathrm{ELIM}}$:

(a) $\exists j \in \{1, \ldots, r\}$ such that $(b, x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_r) \notin C(P - \{i_j\})$,
(b) $(x_1, \ldots, x_r) \notin C(P - \{i\})$,
(c) $\exists h \notin P$ such that $\forall x \in A_h((b, x_1, \ldots, x_r, x) \in C(P \cup \{h\}))$.

(a) and (b) correspond to upward-propagation and (c) corresponds to downward-propagation. We consider each case separately.

Case (a). Since $(b, x_1, \ldots, x_r)$ is the first labelling satisfying (3) and eliminated by S', there are three possibilities:

(i) $(b, x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_r) \notin C(P - \{i_j\})$ in CSP_p or
(ii) $(a, x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_r) \notin C(P - \{i_j\})$ in CSP_p or
(iii) $(a, x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_r) \notin C(P - \{i_j\})$ in CSP'_p.

Since (2) holds in CSP_p, we can deduce that

$$\begin{aligned}&(a, x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_r) \in C(P - \{i_j\}) \Rightarrow\\ &(b, x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_r) \in C(P - \{i_j\}) \quad \text{in } \mathrm{CSP}_p.\end{aligned}$$

Therefore (i) implies (ii), which in turn implies (iii), since, as observed above, all labellings eliminated in CSP_p are also eliminated in CSP'_p. Now (iii) implies that $(a, x_1, \ldots, x_r) \notin C(P)$ in CSP'_p since CSP'_p is a k-solution. This is a contradiction of (3).

Case (b). $(x_1, \ldots, x_r) \notin C(P - \{i\})$ in CSP'_{ELIM} implies immediately that $(x_1, \ldots, x_r) \notin C(P - \{i\})$ in CSP'_p and hence that $(a, x_1, \ldots, x_r) \notin C(P)$ in CSP'_p since CSP'_p is a k-solution. This contradicts (3).

Case (c). Since $(b, x_1, \ldots, x_r)$ is the first labelling eliminated in S' which satisfies (3), for each $x \in A_h$ there are three possibilities:

(i) $(b, x_1, \ldots, x_r, x) \notin C(P \cup \{h\})$ in CSP_p or
(ii) $(a, x_1, \ldots, x_r, x) \notin C(P \cup \{h\})$ in CSP_p or
(iii) $(a, x_1, \ldots, x_r, x) \notin C(P \cup \{h\})$ in CSP'_p.

Since b is neighbourhood substitutable for a in CSP_p, we know that

$$(a, x_1, \ldots, x_r, x) \in C(P \cup \{h\}) \Rightarrow$$
$$(b, x_1, \ldots, x_r, x) \in C(P \cup \{h\}) \quad \text{in } CSP_p.$$

Therefore (i) implies (ii), which in turn implies (iii). Now the fact that (iii) is true for all $x \in A_h$ implies that $(a, x_1, \ldots, x_r) \notin C(P)$ in CSP'_p since CSP'_p is k-consistent. This again contradicts (3).

We have thus shown, by contradiction, that (2) holds in CSP'_p. Therefore, all labels a which are eliminated by neighbourhood substitutions in S are also eliminated in S' (either by neighbourhood substitution or by consistency operations).

Given this result, it is relatively easy to show that S and S' eliminate exactly the same set of labellings. We have just shown that all labels eliminated by neighbourhood substitution in S are also eliminated in S'. The fact that all labels eliminated by neighbourhood substitution in S' are also eliminated in S is a direct consequence of the definition of S'.

We now consider eliminations by consistency. Let $(x_1, \ldots, x_r)$ be the first labelling which is eliminated by strong consistency operations in one of S and S', but not the other. Whether $(x_1, \ldots, x_r)$ was eliminated by upward-propagation or downward-propagation, the premise of the same rule will become true at some point of the other sequence, by the above result that all labels eliminated by neighbourhood substitution in one sequence are also eliminated in the other. Since both sequences produce a k-solution, by their respective definitions, $(x_1, \ldots, x_r)$ will also be eliminated in the other sequence. This contradiction shows that S and S' eliminate exactly the same set of labellings.

Let S'' be a copy of S' in which we discard all but the first determination of a k-solution. In other words, S'' is simply the determination of a k-solution followed by the sequence of neighbourhood substitutions in S. We know from Lemma 3.5(c) that a neighbourhood substitution applied to a k-solution can only induce eliminations by upward-propagation. Lemma 3.5(a) and Lemma 3.6 tell us that these eliminations cannot in turn induce new eliminations by neighbourhood

substitution or downward-propagation. The CSPs resulting from the application of S' and S'' are thus identical except for updates to constraints of the form

$$C(P) := C(P) \cap (A_i \times A_{i_1} \times \cdots \times A_{i_r}),$$

and by Lemma 3.5(a), these two CSPs are isomorphic. Theorem 3.7 follows from Theorem 2.5 and the transitivity of isomorphism. □

4. Local substitution

Freuder [5] defined the general notion of substitutability as follows: given two possible labels a and b for a variable i, a is substitutable for b iff substituting the value of a for b at variable i in any solution yields another solution. Jeavons et al. [7] generalized substitutability to sets of labellings for sets of variables: given two sets of labellings A and B for the set of variables X, A is substitutable for B on the set of variables X if each solution whose projection on X is a labelling $b \in B$ can be converted into another solution by the replacement of b by some labelling $a \in A$. This is particularly interesting in the case that X is the scope of a constraint, $B = C(X)$ and $A = C(X) - \{c\}$ for some labelling c.

Unfortunately, in the worst case, testing for substitutability is as difficult as solving the CSP. In the same way that global consistency has local versions which can be applied in polynomial time, substitutability has tractable local versions, such as neighbourhood substitutability. A more powerful version of local substitutability was defined in [7]. Let $\mathrm{cl}(X)$ denote the closure of X, the union of the scopes Y of constraints $C(Y)$ such that $X \cap Y \neq \emptyset$.

Definition 4.1. The set of labellings A is said to be *locally substitutable* for B on the set of variables X if, for all $b \in B$, and for each labelling u which satisfies the constraints on $\mathrm{cl}(X)$ and such that $\Pi_X u = b$, there is another labelling v which satisfies the constraints on $\mathrm{cl}(X)$ and such that $\Pi_{\mathrm{cl}(X)-X} v = \Pi_{\mathrm{cl}(X)-X} u$ and $\Pi_X v \in A$.

The notation Π_X, borrowed from relational algebra, denotes the projection operation onto the set of variables X.

Local substitutability is more powerful than neighbourhood substitutability, but is potentially much more costly to apply since we need to exhaust over all consistent labellings of $\mathrm{cl}(X)$. In the worst case this is a combinatorial function of the size of $\mathrm{cl}(X)$. We would not envisage applying local substitutability to a CSP whose constraint graph was the complete graph, since $\mathrm{cl}(X)$ would be the set of all variables.

Since applying k-consistency, for $k > 2$, converts the constraint graph into the complete graph, we would not apply local substitution after k-consistency for $k > 2$. On the other hand, the concept of local substitutability subsumes arc consistency (2-consistency). For example, a label a for node i which cannot be extended to a consistent labelling of the edge (i, j) will be eliminated by local

substitutability since there is no labelling of $\mathrm{cl}(X)$ in which the node i is labelled by a. This shows that no equivalent of Theorem 3.7 exists for local substitution since we have no reason to combine it with consistency.

Theorem 2.5 tells us that two convergent sequences of neighbourhood substitutions produce isomorphic CSPs. A simple example will suffice to show that two convergent sequences of local substitution operations can produce very different CSPs. Consider the example of a graph colouring problem, consisting of a single constraint, illustrated in Fig. 6(a). Whether $X=\{1\}$ or $\{2\}$, $\mathrm{cl}(X)$ is $\{1, 2\}$. The set of all consistent labellings of $\mathrm{cl}(X)$ is:

$$\{(red, green), (red, blue), (green, blue), (blue, green)\}.$$

The set of labels $\{green, blue\}$ is substitutable for $\{red, green, blue\}$ at node 1, which leaves the CSP shown in Fig. 6(b). The set of consistent labellings is

$$\{(green, blue), (blue, green)\}.$$

Alternatively we could have chosen to replace the set of labels $\{red, green, blue\}$ at node 1 by $\{red\}$. In the resulting constraint $\{(red, green), (red, blue)\}$, the set of labels $\{blue\}$ is substitutable for $\{green, blue\}$ at node 2, leaving the CSP shown in Fig. 6(c). The set of consistent labellings is

$$\{(red, blue)\}.$$

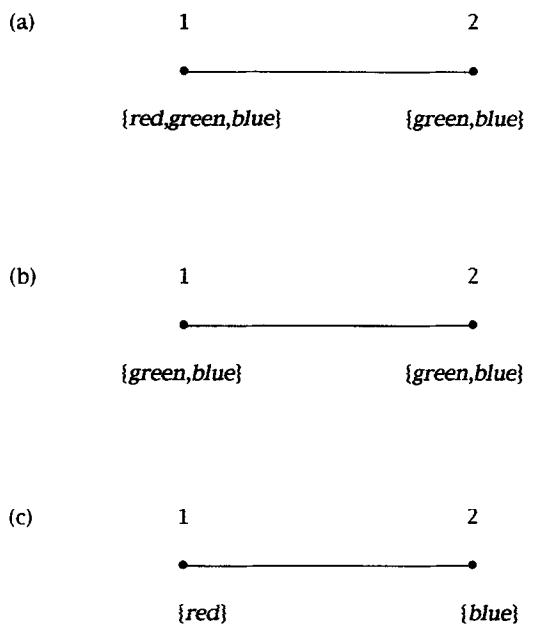

Fig. 6. (a) The constraint graph of a simple graph colouring problem; (b), (c) two non-isomorphic problems which result from the application of two different sequences of local substitution operations.

The two CSPs in Figs. 6(b) and 6(c) are clearly not isomorphic. Indeed, they have a different number of solutions. This implies that heuristics may exist which indicate which elements of constraints to eliminate by local substitutions in order to minimize the size of the resulting CSP. Theorem 2.5 showed that any such heuristic would have no effect in the case of neighbourhood substitutions.

5. Neighbourhood substitution algorithm

One way of finding a convergent sequence of neighbourhood substitutions is to repeatedly apply the ns-elimination rule (see Section 3):

```
NS-1
  repeat
    for i:=1 to n do
      for each a∈A_i do
        for each b∈A_i such that b≠a do
          if for all constraints C(P) such that i∈P={i, i_1, ..., i_r}
            for all (x_1, ..., x_r)∈A_{i_1}×···×A_{i_r}
              (a, x_1, ..., x_r)∈C(P) ⇒ (b, x_1, ..., x_r)∈C(P)
          then
            Record (a→b, i) in sequence of substitutions E;
            Delete a from A_i;
          end_if
  until there are no deletions in an iteration
```

To avoid both $a \rightarrow b$ and $b \rightarrow a$ being accepted as neighbourhood substitutions, in the case that a and b are interchangeable [5], it is essential that the test $b \in A_i$ be actually performed at each iteration.

Let k be an upper bound on the order of the constraints in the CSP, and let c be the number of constraints. We assume that k is a constant. To calculate the worst-case time complexity of NS-1, it is sufficient to count the maximum number of constraint accesses. For each of the iterations of the repeat–until loop, the number of constraint accesses is $O(a^2ca^{k-1})$. The number of iterations of the repeat–until loop is bounded above by the maximum number of label deletions, which is $O(an)$. Therefore the worst-case time complexity of NS-1 is

$$O(a^{k+2}nc).$$

We can make a significant improvement on this worst-case time complexity using ideas from Bessière's arc consistency algorithm AC-6 [2]. We first need the following definition.

Definition 5.1. For $a,b \in A_i$, (x, P) is a *block* for $a \rightarrow b$ at variable i if $i \in P = \{i, i_1, \ldots, i_r\}$, $x = (x_1, \ldots, x_r) \in A_{i_1} \times \cdots \times A_{i_r}$, $(a, x) \in C(P)$ and $(b, x) \notin C(P)$.

Note that x is a tuple $x=(x_1, \ldots, x_r)$, where $r+1$ is the degree of the constraint $C(P)$, and $r \geq 0$. The existence of (x, P) blocks the elimination of a by the neighbourhood substitution $a \rightarrow b$.

We assume that, for each variable i, the set of possible blocks (x, P) has been assigned an arbitrary total ordering, such as a lexicographical ordering. The algorithm NS-2 keeps track of a block for $a \rightarrow b$ at i, for each variable i and for each pair of distinct labels $a,b \in A_i$. When this block (x, P), where $x=(x_1, \ldots, x_r)$, is no longer valid following the elimination of some x_j $(1 \leq j \leq r)$ by neighbourhood substitution, NS-2 searches for the next block according to the total ordering. If no block remains then the neighbourhood substitution $a \rightarrow b$ is itself accepted.

When a neighbourhood substitution $a \rightarrow b$ is accepted it is added to *NS_List* to be processed later. Before it is actually processed, we must verify that neither a nor b have already been deleted from A_i.

Blocks is a set of ordered pairs of the form $((x, P), (a \rightarrow b, i))$ where (x, P) is a block for the neighbourhood substitution $a \rightarrow b$ at i. After initialization, we will find in *Blocks* exactly one block for each potential neighbourhood substitution $(a \rightarrow b, i)$ which has not already been added to *NS_List*.

```
NS-2
{Initialization}
  NS_List := ∅;
  Blocks := ∅;
  for i := 1 to n do
    for all pairs of labels a,b∈A_i such that b≠a do
    begin
      Look for first block (x, P) for (a → b, i);
      if (x, P) exists
      then insert ((x, P), (a → b, i)) in Blocks
      else add (a → b, i) to NS_List
    end;

{Propagation}
  while NS_List≠∅ do
    Select and delete an element (a → b, i) from NS_List;
    if a,b∈A_i
    then
      Record (a → b, i) in sequence of substitutions E;
      Delete a from A_i;
      {all blocks (y, Q) which used the value a for
        variable i are now invalid and must be replaced}
      for each ((y, Q), (c → d, j))∈Blocks such that
        i∈Q={j, i, i_1, ..., i_s} and y=(a, y_1, ..., y_s) do
          if c,d∈A_j
          then
            Delete ((y, Q), (c → d, j)) from Blocks;
```

```
                Look for next block (y', Q') for (c → d, i);
                if (y', Q') exists
                then insert ((y', Q'), (c → d, j)) in Blocks
                else add (c → d, j) to NS_List
            end_if
        end_if
    end_while
```

Blocks contains at most a^2n elements. In order to have direct access to each element $((x, P), (a \rightarrow b, i))$ of *Blocks* through each of the component labels (x_j, i_j) of the block (x, P), where $P=\{i, i_1, \ldots, i_r\}$ and $x=(x_1, \ldots, x_r)$, a suitable data structure is an array of lists, indexed by the component labels (x_j, i_j). Thus $((x, P), (a \rightarrow b, i))$ is, in fact, stored r times. We assume that r is a constant, since it is bounded above by $k-1$, where k is the maximum order of the constraints.

A block (x, P) for the neighbourhood substitution $a \rightarrow b$ must satisfy $(a, x) \in C(P)$ and $(b, x) \notin C(P)$. The search for a block for $a \rightarrow b$ will thus be most efficient when constraints are tight (there are few labellings $(a, x) \in C(P)$) or when constraints are loose (there are few labellings $(b, x) \notin C(P)$). Define m to be the maximum value of $\min\{|C(P)|, |C(P)'|\}$ over all constraints $C(P)$, where $C(P)'$ denotes the complement of $C(P)$.

In NS-2, the total time spent searching for blocks is $O(acm)$, assuming that in tight constraints we exhaust over all x such that $(a, x) \in C(P)$ and in loose constraints we exhaust over all x such that $(b, x) \notin C(P)$. The number of iterations of the while loop in NS-2 is bounded above by a^2n, the total number of possible neighbourhood substitutions. Making the very reasonable assumptions that $c \geq n$ and $m \geq a$, we can conclude that the worst-case time complexity of NS-2 is

$$O(acm).$$

In the worst case, this is

$$O(a^{k+1}c).$$

For binary constraints [1], this gives a complexity of $O(a^3c)$. The space complexity of both *Blocks* and *NS_List* is $O(a^2n)$, which is independent of the order of the constraints.

6. A novel algorithm to find all consistent labellings

It is clear that neighbourhood substitutions are potentially useful when searching for a single solution. They also turn out to be useful when searching for all solutions. Let NS denote any algorithm to detect and eliminate neighbourhood substitutable labels until convergence. Let SOLVE denote any algorithm which returns the set of all solutions to the CSP passed as a parameter. Then the following algorithm finds all solutions to a CSP $\mathcal{P}$, taking advantage of any neighbourhood substitutions.

```
NEIGHBOURHOOD_SOLVE(𝒫):
begin
  NS(𝒫, 𝒫', E);  {𝒫' is the reduced version of the CSP 𝒫 after
                  neighbourhood substitution operations; E is
                  the convergent sequence of neighbourhood
                  substitutions x → y used to eliminate labels}
  Sol := SOLVE(𝒫');  {Sol is the set of solutions to the CSP 𝒫'}
  RECONSTRUCT(Sol, 𝒫, E);
end;  {Sol is now the set of solutions to the original CSP 𝒫}

RECONSTRUCT(Sol, 𝒫, E):
begin
  L := Sol;
  while L≠∅ do
    Select a solution (x_1, ..., x_n)∈L and delete it from L;
    for i:=1 to n do
      for each x_i' such that the neighbourhood
      substitution x_i' → x_i is in E do
        if x' = (x_1, ..., x_{i-1}, x_i', x_{i+1}...x_n) satisfies all
          constraints C(P) such that i∈P
        then
          if x'∉Sol
          then add x' to L and to Sol;
  end;
end;
```

6.1. *Proof that algorithm finds all consistent labellings*

Let $(y_1, \ldots, y_n)$ be a consistent labelling. Let E be the sequence of substitutions applied to $\mathscr{P}$ to produce $\mathscr{P}'$. For each node $i=1, \ldots, n$, there is a subsequence of substitutions at node i: $y_i \rightarrow \cdots x_i$ in E such that x_i is not eliminated in E.

Let E_{pref} be any prefix of E, and, for each $i=1, \ldots, n$, let z_i be the label such that $y_i \mapsto \cdots z_i$ in E_{pref}. By Lemma 2.2, the fact that $(y_1, \ldots, y_n)$ satisfies all the original constraints implies that $(z_1, \ldots, z_n)$ also satisfies all the original constraints. In particular, when $E_{\text{pref}}=E$, we can deduce that $(x_1, \ldots, x_n)$ is a solution to $\mathscr{P}'$ since it satisfies all constraints and none of the x_i have been eliminated. Hence the algorithm will find $(x_1, \ldots, x_n)$. By an easy induction, working back from $(x_1, \ldots, x_n)$ to $(y_1, \ldots, y_n)$, we can show that the algorithm finds all labellings $(z_1, \ldots, z_n)$ corresponding to prefixes E_{pref} of E. Hence the algorithm finds all consistent labellings $(y_1, \ldots, y_n)$.

6.2. *Complexity of* NEIGHBOURHOOD_SOLVE

The complexity of NS has already been discussed in detail in Section 5. The sequence E is stored as an array of lists E_x of all neighbourhood substitutions

$y \rightarrow x$ in E. The space required is O(an) since each label for each node can be eliminated at most once in E. The sets L and *Sol* can be stored in an a-way tree to ensure that the insertion, deletion and test of membership operations can all be achieved in O(n) time and O(Nan) space, where N is the total number of solutions to $\mathcal{P}$.

In this a-way tree, each node at level i corresponds to a prefix $x_1, \ldots, x_i$ of one of the N solutions. Direct access to each possible extension x_{i+1} of $x_1, \ldots, x_i$ to level $i+1$ requires an array of length a. Minor trade-offs between time and space are clearly possible according to the choice of data structure for L and *Sol*.

We assume direct access to the set of constraints involving node i, and that, for each constraint $C(P)$, the test $x' \in C(P)$ is O(1). To calculate the worst case time complexity of RECONSTRUCT(*Sol*, $\mathcal{P}$, E), it is sufficient to count the number of accesses to constraints and the number of membership tests of *Sol*. The number of constraint accesses is at most

$$N \sum_{i=1}^{n} a \cdot (\text{number of constraints involving } i) = \mathrm{O}(Nac),$$

where c is the number of constraints in $\mathcal{P}$. The factor a is very pessimistic; it is an upper bound on the number of labels x_i' such that $x_i' \rightarrow x_i$ is in E, for a given label x_i.

The number of membership tests of *Sol* is O(Nn) since each solution $(y_1, \ldots, y_n)$ is generated at most n times. This is because we can only generate $(y_1, \ldots, y_n)$ in RECONSTRUCT(*Sol*, $\mathcal{P}$, E) by replacing x_i by y_i, for some $i \in \{1, \ldots, n\}$, in a solution $(y_1, \ldots, y_{i-1}, x_i, y_{i+1}, \ldots, y_n)$ where x_i is the unique label such that $y_i \rightarrow x_i$ in E.

Since a constraint access is an O(1) operation and a membership test of *Sol* is an O(n) operation, the time complexity of RECONSTRUCT(*Sol*, $\mathcal{P}$, E) is

$$\mathrm{O}(N(ac + n^2)).$$

Its space complexity is

$$\mathrm{O}(Nan).$$

We note in passing that simply outputting the N solutions is an O(Nn) operation.

If N is large, we might prefer to stop after generating K solutions, for some fixed value K. In this case, the time and space complexities of RECONSTRUCT(*Sol*, $\mathcal{P}$, E) are O($K(ac+n^2)$) and O(Kan) respectively.

The efficiency of NEIGHBOURHOOD_SOLVE depends critically on the time required to find all solutions to the reduced problem $\mathcal{P}'$, which may greatly exceed the time complexity of RECONSTRUCT(*Sol*, $\mathcal{P}$, E). However, we have shown that neighbourhood substitution is useful not only when searching for a single solution, but also when searching for all solutions.

7. Conclusion

Applying a convergent sequence of neighbourhood substitutions is a reduction operation for constraints which has the following properties:

(1) The result is invariant, modulo isomorphism.
(2) It can be applied in $O(a^{k+1}c)$ time for order-k constraints, where a is the maximum number of labels in a domain, and c the number of constraints.
(3) Combining it with k-consistency only requires establishing k-consistency once.
(4) After solving the reduced CSP we can generate all N solutions to the original CSP in $O(N(ac+n^2))$ time, where n is the number of variables.

We can conclude that neighbourhood substitution has a sufficient number of interesting properties to make it useful in many constraint satisfaction problems.

Further theoretical research and experimental trials on specific problems are required to determine in which classes of CSPs neighbourhood substitutions are common and in which classes of CSPs applying neighbourhood substitutions during backtracking would be worthwhile.

Acknowledgements

The author would like to thank the anonymous referees whose comments contributed significantly to the presentation of this paper.

References

[1] A. Bellicha, C. Capelle, M. Habib, T. Kökény and M.C. Vilarem, CSP techniques using partial orders on domain values, in: *Proceedings ECAI'94 Workshop on Constraint Satisfaction Issues Raised by Practical Applications*, Amsterdam (1994).
[2] C. Bessière, Arc-consistency and arc-consistency again, *Artif. Intell.* **65** (1) (1994) 179–190.
[3] M.C. Cooper, An optimal k-consistency algorithm, *Artif. Intell.* **41** (1990) 89–95.
[4] E.C. Freuder, Synthesizing constraint expressions, *Commun. ACM* **21** (11) (1978) 958–966.
[5] E.C. Freuder, Eliminating interchangeable values in constraint satisfaction problems, in: *Proceedings AAAI-91*, Anaheim, CA (1991) 227–233.
[6] C. Han and C. Lee, Comments on Mohr and Henderson's path consistency algorithm, *Artif. Intell.* **36** (1988) 125–130.
[7] P.G. Jeavons, D.A. Cohen and M.C. Cooper, A substitution operation for constraints, in: *Proceedings PPCP94*, Lecture Notes in Computer Science **874** (Springer, Berlin, 1994) 1–9.
[8] A.K. Mackworth, Consistency in networks of relations, *Artif. Intell.* **8** (1977) 99–118.
[9] R. Mohr and T.C. Henderson, Arc and path consistency revisited, *Artif. Intell.* **28** (1986) 225–233.
[10] U. Montanari, Networks of constraints: fundamental properties and applications to picture processing, *Information Sci.* **7** (1974) 95–132.
[11] E. Tsang, *Foundations of Constraint Satisfaction* (Academic Press, London, 1993).
[12] D.L. Waltz, Understanding line drawings of scenes with shadows, in: P.H. Winston, ed., *The Psychology of Computer Vision* (McGraw-Hill, New York, 1975).