Interchangeability in Soft CSPs

Stefano Bistarelli[1,2], Boi Faltings[3], and Nicoleta Neagu[3]

[1] Istituto di Informatica e Telematica, CNR, Pisa, Italy
Stefano.Bistarelli@iit.cnr.it,
[2] Dipartimento di Scienze, Universitá "G. D'annunzio", Pescara, Italy
bista@sci.unich.it,
[3] Artificial Intelligence Laboratory (LIA), EPFL, Ecublens, Switzerland
boi.faltings|nicoleta.neagu@epfl.ch

Abstract. Substitutability and interchangeability in constraint satisfaction problems (CSPs) have been used as a basis for search heuristics, solution adaptation and abstraction techniques. In this paper, we consider how the same concepts can be extended to *soft* constraint satisfaction problems (SCSPs).
We introduce two notions: *threshold* α and *degradation* δ for substitutability and interchangeability, ($_\alpha$substitutability/interchangeability and $^\delta$substitutability/interchangeability respectively). We show that they satisfy analogous theorems to the ones already known for hard constraints. In $_\alpha$interchangeability, values are interchangeable in any solution that is better than a threshold α, thus allowing to disregard differences among solutions that are not sufficiently good anyway. In $^\delta$interchangeability, values are interchangeable if their exchange could not degrade the solution by more than a factor of δ.
We give efficient algorithms to compute ($^\delta/_\alpha$)interchangeable sets of values for a large class of SCSPs.

1 Introduction

Substitutability and interchangeability in CSPs have been introduced by Freuder ([12]) in 1991 with the intention of improving search efficiency for solving CSP. Interchangeability has since found other applications in abstraction frameworks ([14, 20, 12, 8]) and solution adaptation ([19, 15]). One of the difficulties with interchangeability has been that it does not occur very frequently.

In many practical applications, constraints can be violated at a cost, and solving a CSP thus means finding a value assignment of minimum cost. Various frameworks for solving such soft constraints have been proposed [13, 10, 16, 11, 18, 5, 6, 2]. The soft constraints framework of c-semirings [5, 2] has been shown to express most of the known variants through different instantiations of its operators, and this is the framework we are considering in this paper.

The most straightforward generalization of interchangeability to soft CSP would require that exchanging one value for another does not change the quality of the solution at all. This generalization is likely to suffer from the same weaknesses as interchangeability in hard CSP, namely that it is very rare.

Fortunately, soft constraints also allow weaker forms of interchangeability where exchanging values may result in a degradation of solution quality by some measure δ. By allowing more degradation, it is possible to increase the amount of interchangeability in a problem to the desired level. We define ${}^{\delta}$substitutability/interchangeability as a concept which ensures this quality. This is particularly useful when interchangeability is used for solution adaptation.

Another use of interchangeability is to reduce search complexity by grouping together values that would never give a sufficiently good solution. In ${}_{\alpha}$substitutability/interchangeability, we consider values interchangeable if they give equal solution quality in all solutions better than α, but possibly different quality for solutions whose quality is $\leq \alpha$.

Just like for hard constraints, full interchangeability is hard to compute, but can be approximated by neighbourhood interchangeability which can be computed efficiently and implies full interchangeability. We define the same concepts for soft constraints, and prove that neighborhood implies full $({}^{\delta}/_{\alpha})$substitutability/interchangeability. We give algorithms for neighborhood $({}^{\delta}/_{\alpha})$substitutability/interchangeability, and we prove several interesting and useful properties of the concepts. Finally, we give two examples where $({}^{\delta}/_{\alpha})$interchangeability is applied to solution adaptation in configuration problems with two different soft constraint frameworks: delay and cost constraints, and show its usefulness in these practical contexts.

2 Background

2.1 Soft CSPs

Several formalization of the concept of *soft constraints* are currently available. In the following, we refer to the one based on c-semirings [2, 4, 5, 7], which can be shown to generalize and express many of the others [3]. A soft constraint may be seen as a constraint where each instantiations of its variables has an associated value from a partially ordered set which can be interpreted as a set of preference values. Combining constraints will then have to take into account such additional values, and thus the formalism has also to provide suitable operations for combination ($\times$) and comparison ($+$) of tuples of values and constraints. This is why this formalization is based on the concept of c-semiring, which is just a set plus two operations.

Semirings. A semiring is a tuple $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ such that: 1. A is a set and $\mathbf{0}, \mathbf{1} \in A$; 2. $+$ is commutative, associative and $\mathbf{0}$ is its unit element; 3. $\times$ is associative, distributes over $+$, $\mathbf{1}$ is its unit element and $\mathbf{0}$ is its absorbing element. A c-semiring is a semiring $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ such that: $+$ is idempotent, $\mathbf{1}$ is its absorbing element and $\times$ is commutative. Let us consider the relation $\leq_S$ over A such that $a \leq_S b$ iff $a + b = b$. Then it is possible to prove that (see [5]): 1. $\leq_S$ is a partial order; 2. $+$ and $\times$ are monotone on $\leq_S$; 3. $\mathbf{0}$ is its minimum and $\mathbf{1}$ its maximum; 4. $\langle A, \leq_S\rangle$ is a complete lattice and, for all $a, b \in A$, $a + b = lub(a, b)$ (where *lub* is the *least upper bound*). Moreover, if $\times$ is idempotent, then: $+$ distributes over $\times$; $\langle A, \leq_S\rangle$ is a complete distributive lattice and $\times$ its *glb* (*greatest lower bound*). Informally, the relation $\leq_S$ gives us a way to compare semiring values

and constraints. In fact, when we have $a \leq_S b$, we will say that b *is better than* a. In the following, when the semiring will be clear from the context, $a \leq_S b$ will be often indicated by $a \leq b$.

Constraint Problems. Given a semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ and an ordered set of variables V over a finite domain D, a *constraint* is a function which, given an assignment $\eta : V \to D$ of the variables, returns a value of the semiring. By using this notation we define $\mathcal{C} = \eta \to A$ as the set of all possible constraints that can be built starting from S, D and V.

Note that in this *functional* formulation, each constraint is a function (as defined in [7]) and not a pair (as defined in [4, 5]). Such a function involves all the variables in V, but it depends on the assignment of only a finite subset of them. So, for instance, a binary constraint $c_{x,y}$ over variables x and y, is a function $c_{x,y} : V \to D \to A$, but it depends only on the assignment of variables $\{x, y\} \subseteq V$. We call this subset the *support* of the constraint. More formally, consider a constraint $c \in \mathcal{C}$. We define its support as $supp(c) = \{v \in V \mid \exists \eta, d_1, d_2 . c\eta[v := d_1] \neq c\eta[v := d_2]\}$, where

$$\eta[v := d]v' = \begin{cases} d & \text{if } v = v', \\ \eta v' & \text{otherwise.} \end{cases}$$

Note that $c\eta[v := d_1]$ means $c\eta'$ where η' is η modified with the assignment $v := d_1$ (that is the operator [ ] has precedence over application). Note also that $c\eta$ is the application of a constraint function $c : V \to D \to A$ to a function $\eta : D \to A$; what we obtain, is a semiring value $c\eta = a$.

A *soft constraint satisfaction problem* is a pair $\langle C, con \rangle$ where $con \subseteq V$ and C is a set of constraints: *con* is the set of variables of interest for the constraint set C, which however may concern also variables not in *con*. Note that a classical CSP is a SCSP where the chosen c-semiring is: $S_{CSP} = \langle \{false, true\}, \vee, \wedge, false, true \rangle$. Fuzzy CSPs [17] can instead be modeled in the SCSP framework by choosing the c-semiring $S_{FCSP} = \langle [0, 1], max, min, 0, 1 \rangle$. Many other "soft" CSPs (Probabilistic, weighted, ...) can be modeled by using a suitable semiring structure ($S_{prob} = \langle [0, 1], max, \times, 0, 1 \rangle$, $S_{weight} = \langle \mathcal{R}, min, +, +\infty, 0 \rangle$, ...).

Fig. 1 shows the graph representation of a fuzzy CSP. Variables and constraints are represented respectively by nodes and by undirected (unary for c_1 and c_3 and binary for c_2) arcs, and semiring values are written to the right of the corresponding tuples. The variables of interest (that is the set *con*) are represented with a double circle. Here we assume that the domain D of the variables contains only elements a and b and c.

Combining and projecting soft constraints. Given the set $\mathcal{C}$, the combination function $\otimes : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ is defined as $(c_1 \otimes c_2)\eta = c_1\eta \times_S c_2\eta$. In words, combining two constraints means building a new constraint whose support involves all the variables of the original ones, and which associates with each tuple of domain values for such variables a semiring element which is obtained by multiplying

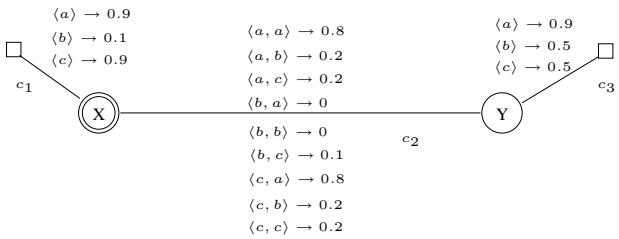

Fig. 1: A fuzzy CSP.

the elements associated by the original constraints to the appropriate subtuples. It is easy to verify that $supp(c_1 \otimes c_2) \subseteq supp(c_1) \cup supp(c_2)$.

Given a constraint $c \in \mathcal{C}$ and a variable $v \in V$, the *projection* of c over $V - \{v\}$, written $c \Downarrow_{(V-\{v\})}$ is the constraint c' s.t. $c'\eta = \sum_{d \in D} c\eta[v := d]$. Informally, projecting means eliminating some variables from the support. This is done by associating with each tuple over the remaining variables a semiring element which is the sum of the elements associated by the original constraint to all the extensions of this tuple over the eliminated variables. In short, combination is performed via the multiplicative operation of the semiring, and projection via the additive one.

Solutions. A *solution* of an SCSP $P = \langle C, con \rangle$ is the constraint $Sol(P) = (\bigotimes C) \Downarrow_{con}$. That is, we combine all constraints, and then project over the variables in *con*. In this way we get the constraint with support (not greater than) *con* which is "induced" by the entire SCSP. Note that when all the variables are of interest we do not need to perform any projection.

For example, the solution of the fuzzy CSP of Fig. 1 associates a semiring element to every domain value of variable x. Such an element is obtained by first combining all the constraints together. For instance, for the tuple $\langle a, a \rangle$ (that is, $x = y = a$), we have to compute the minimum between 0.9 (which is the value assigned to $x = a$ in constraint c_1), 0.8 (which is the value assigned to $\langle x = a, y = a \rangle$ in c_2) and 0.9 (which is the value for $y = a$ in c_3). Hence, the resulting value for this tuple is 0.8. We can do the same work for tuple $\langle a, b \rangle \rightarrow 0.2$, $\langle a, c \rangle \rightarrow 0.2$, $\langle b, a \rangle \rightarrow 0$, $\langle b, b \rangle \rightarrow 0$, $\langle b, c \rangle \rightarrow 0.1$, $\langle c, a \rangle \rightarrow 0.8$, $\langle c, b \rangle \rightarrow 0.2$ and $\langle c, c \rangle \rightarrow 0.2$. The obtained tuples are then projected over variable x, obtaining the solution $\langle a \rangle \rightarrow 0.8$, $\langle b \rangle \rightarrow 0.1$ and $\langle c \rangle \rightarrow 0.8$.

2.2 Interchangeability

Interchangeability in constraint networks was first proposed by Freuder [12] to capture equivalence among values of a variable in a discrete constraint satisfaction problem. Value $v = a$ is *substitutable* for $v = b$ if for any solution where $v = a$, there is an identical solution except that $v = b$. Values $v = a$ and $v = b$ are *interchangeable* if they are substitutable both ways.

Interchangeability offers three important ways for practical applications:

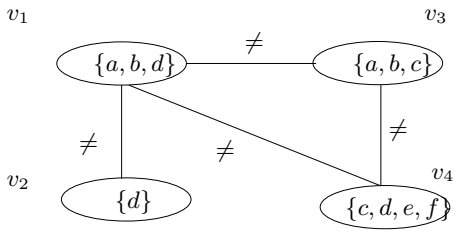

Fig. 2: An example of CSP with interchangeable values.

- by pruning the interchangeable values, which are redundant in a sense, the problem space can be simplified.
- interchangeability can be used as a solution updating tool; this can by used for user-interaction, can help users in taking decisions by offering alternatives, planning, scheduling ...
- can structure and classify the solution space.

Full Interchangeability considers all constraints in the problem and checks if a value a and b for a certain variable v can be interchanged without affecting the global solution. In the CSP in Fig. 2 (taken from [9]), d, e and f are fully interchangeable for v_4. This is because we inevitably have $v_2 = d$, which implies that v_1 cannot be assigned d in any consistent global solution. Consequently, the values d, e and f can be freely permuted for v_4 in any global solution.

There is no efficient algorithm for computing full Interchangeability, as it may require computing all solutions. The localized notion of *Neighbourhood Interchangeability* considers only the constraints involving a certain variable v. In this notion, a and b are *neighbourhood interchangeable* if for every constraint involving v, for every tuple that admits $v = a$ there is an otherwise identical tuple that admits $v = b$, and vice-versa. In Fig. 2, e and f are neighbourhood interchangeable for v_4.

Freuder showed that neighbourhood interchangeability always implies full interchangeability and can therefore be used as an approximation. He also provided an efficient algorithm (Algorithm 1) for computing neighborhood interchangeability [12], and investigated its use for preprocessing CSP before searching for solutions [1]. Every node in the discrimination tree (Fig. 1) corresponds to a set of assignments to variables in the neighbourhood of v that are compatible with some value of v itself. Interchangeable values are found by the fact that they follows the same path and fall into the same ending node. Fig. 3 shows an example of execution of Algorithm 1 for variable v_4. Domain values e and f are shown to be interchangeable.

3 Interchangeability in Soft CSPs

In soft CSPs, there is not any crisp notion of consistency. In fact, each tuple is a possible solution, but with different level of preference. Therefore, in this framework, the notion of interchangeability becomes finer: to say that values a and b are interchangeable we have also to consider the assigned semiring level.

Create the root of the discrimination tree for variable v_i;
Let $D_{v_i} = \{$the set of domain values d_{v_i} for variable $v_i\}$;
Let $Neigh(\{v_i\}) = \{$all neighborhood variables v_j of variable $v_i\}$;
for all $d_{v_i} \in D_{v_i}$ **do**
  for all $v_j \in Neigh(\{v_i\})$ **do**
    for all $d_{v_j} \in D_{v_j}$ s.t. d_{v_j} is consistent with d_{v_i} for v_i **do**
      if there exists a child node corresponding to $v_j = d_{v_j}$ **then**
        move to it,
      else
        construct such a node and move to it;
  Add $v_i, \{d_{v_i}\}$ to annotation of the node;
  Go back to the root of the discrimination tree.

Algorithm 1: Discrimination Tree for variable v_i.

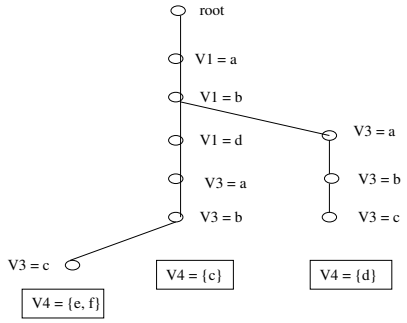

Fig. 3: An example of CSP with computation of neighborhood interchangeable values.

More precisely, if a domain element a assigned to variable v can be substituted in each tuple solution with a domain element b without obtaining a worse semiring level we say that b is full substitutable for a.

Definition 1 (Full Substitutability (FS)). *Consider two domain values b and a for a variable v, and the set of constraints C; we say that b is Full Substitutable for a on v ($b \in FS_v(a)$) if and only if*

$$\bigotimes C\eta[v := a] \leq_S \bigotimes C\eta[v := b]$$

When we restrict this notion only to the set of constraints C_v that involves variable v we obtain a local version of substitutability.

Definition 2 (Neighborhood Substitutability (NS)). *Consider two domain values b and a for a variable v, and the set of constraints C_v involving v; we say that b is neighborhood substitutable for a on v ($b \in NS_v(a)$) if and only if*

$$\bigotimes C_v\eta[v := a] \leq_S \bigotimes C_v\eta[v := b]$$

When the relations hold in both directions, we have the notion of Full/Neighborhood interchangeability of b with a.

Definition 3 (Full and Neighborhood Interchangeability (*FI* and *NI*)). *Consider two domain values b and a, for a variable v, the set of all constraints C and the set of constraints C_v involving v. We say that b is fully interchangeable with a on v ($FI_v(a/b)$) if and only if $b \in FS_v(a)$ and $a \in FS_v(b)$, that is*

$$\bigotimes C\eta[v := a] = \bigotimes C\eta[v := b].$$

We say that b is Neighborhood interchangeable with a on v ($NI_v(a/b)$) if and only if $b \in NS_v(a)$ and $a \in NS_v(b)$, that is

$$\bigotimes C_v\eta[v := a] = \bigotimes C_v\eta[v := b].$$

This means that when a and b are interchangeable for variable v they can be exchanged without affecting the level of any solution.

Two important results that hold in the crisp case can be proven to be satisfied also with soft CSPs: transitivity and extensivity of interchangeability/substituability.

Theorem 1 (Extensivity: $NS \implies FS$ and $NI \implies FI$). *Consider two domain values b and a for a variable v, the set of constraints C and the set of constraints C_v involving v. Then, neighborhood (substituability) interchangeability implies full (substituability) interchangeability.*

Theorem 2 (Transitivity: $b \in NS_v(a), a \in NS_v(c) \implies b \in NS_v(c)$). *Consider three domain values a, b and c, for a variable v. Then,*

$$b \in NS_v(a), a \in NS_v(c) \implies b \in NS_v(c).$$

Similar results hold for FS, NI and FI.

As an example of interchangeability and substitutability consider the fuzzy CSP represented in Fig. 1. The domain value c is neighborhood interchangeable with a on x ($NI_x(a/c)$); in fact, $c_1 \otimes c_2\eta[x := a] = c_1 \otimes c_2\eta[x := c]$ for all η. The domain values c and a are also neighborhood substitutable for b on x ($\{a, c\} \in NS_v(b)$). In fact, for any η we have $c_1 \otimes c_2\eta[x := b] \leq c_1 \otimes c_2\eta[x := c]$ and $c_1 \otimes c_2\eta[x := b] \leq c_1 \otimes c_2\eta[x := a]$.

3.1 Degradations and Thresholds

In soft CSPs, any value assignment is a solution, but may have a very bad preference value. This allows broadening the original interchangeability concept to one that also allows degrading the solution quality when values are exchanged. We call this ${}^{\delta}$interchangeability, where δ is the *degradation* factor.

When searching for solutions to soft CSP, it is possible to gain efficiency by not distinguishing values that could in any case not be part of a solution of sufficient quality. In ${}_{\alpha}$interchangeability, two values are interchangeable if they do not affect the quality of any solution with quality better than α. We call α the *threshold* factor.

Both concepts can be combined, i.e. we can allow both degradation and limit search to solutions better than a certain threshold (${}^{\delta}_{\alpha}$interchangeability). By extending the previous definitions we can define thresholds and degradation version of full/neighbourhood substitutability/interchangeability.

Definition 4 (${}^{\delta}$Full Substitutability (${}^{\delta}FS$)). *Consider two domain values b and a for a variable v, the set of constraints C and a semiring level δ; we say that b is* ${}^{\delta}$full Substitutable *for a on v ($b \in {}^{\delta}FS_v(a)$) if and only if for all assignments η,*

$$\bigotimes C\eta[v := a] \times_S \delta \leq_S \bigotimes C\eta[v := b]$$

Definition 5 (${}_{\alpha}$Full Substitutability (${}_{\alpha}FS$)). *Consider two domain values b and a, for a variable v, the set of constraints C and a semiring level α; we say that b is* ${}_{\alpha}$full substitutable *for a on v ($b \in {}_{\alpha}FS_v(a)$) if and only if for all assignments η,*

$$\bigotimes C\eta[v := a] \geq \alpha \implies \bigotimes C\eta[v := a] \leq_S \bigotimes C\eta[v := b]$$

Similarly all the notion of ${}^{\delta}/_{\alpha}$Neighborhood Substitutability (${}^{\delta}/_{\alpha}NS$) and of ${}^{\delta}/_{\alpha}$Full/Neighborhood Interchangeability (${}^{\delta}/_{\alpha}FI/NI$) can be defined (just considering the relation in both directions and changing C with C_v).

As an example consider Fig. 1. The domain values c and b for variable y are ${}_{0.2}$Neighborhood Interchangeable. In fact, the tuple involving c and b only differ for the tuple $\langle b, c\rangle$ that has value 0.1 and for the tuple $\langle b, b\rangle$ that has value 0. Since we are interested only to solutions greater than 0.2, these tuples are excluded from the match. The meaning of degradation assume different meanings when instantiated to different semirings:

1. fuzzy CSP: $b \in {}^{\delta}FS_v(a)$ gets instantiated to:

   $$min(min_{c\in C}(c\eta[v := a]), \delta) \leq min_{c\in C}(c\eta[v := b])$$

   which means that changing $v := b$ to $v := a$ does not make the solution worse than before or worse than δ. In the practical case where we want to only consider solutions with a quality better than δ, this means that substitution will never put a solution out of this class.
2. weighted CSP: $b \in {}^{\delta}FS_v(a)$ gets instantiated to:

   $$\sum_{c\in C} c\eta[v := a] + \delta \geq \sum_{c\in C} c\eta[v := b]$$

   which means that the penalty for the solution does not increase by more than a factor of δ. This allows for example to express that we would not want to tolerate more than δ in extra cost. Note, by the way, that $\leq_S$ translates to $\geq$ in this version of the soft CSP.
3. probabilistic CSP: $b \in {}^{\delta}FS_v(a)$ gets instantiated to:

   $$(\prod_{c\in C} c\eta[v := a]) \cdot \delta \leq \prod_{c\in C} c\eta[v := b]$$

which means that the solution with $v = b$ is not degraded by more than a factor of δ from the one with $v = a$.

4. crisp CSP: $b \in {}^{\delta}FS_v(a)$ gets instantiated to:

$$(\bigwedge_{c \in C} c\eta[v := a]) \wedge \delta \Rightarrow (\bigwedge_{c \in C} c\eta[v := b])$$

which means that when $\delta = true$, whenever a solution with $v = a$ satisfies all constraints, so does the same solution with $v = b$. When $\delta = false$, it is trivially satisfied (i.e. δ is too loose a bound to be meaningful).

3.2 Properties of Degradations and Thresholds

As it is very complex to determine full interchangeability/substitutability, we start by showing the fundamental theorem that allows us to approximate ${}^{\delta}/_{\alpha}FS/FI$ by ${}^{\delta}/_{\alpha}NS/NI$:

Theorem 3 (Extensivity). *${}^{\delta}$neighbourhood substitutability implies ${}^{\delta}$full substitutability and ${}_{\alpha}$neighbourhood substitutability implies ${}_{\alpha}$full substitutability.*

This theorem is of fundamental importance since it gives us a way to approximate full interchangeability by neighborhood interchangeability which is much less expensive to compute.

Theorem 4 (Transitivity using thresholds and degradations). *Consider three domain values a, b and c, for a variable v. Then,*

$$b \in {}^{\delta_1}NS_v(a), a \in {}^{\delta_2}NS_v(c) \implies b \in {}^{\delta_1 \times \delta_2}NS_v(c) \text{ and}$$
$$b \in {}_{\alpha_1}NS_v(a), a \in {}_{\alpha_2}NS_v(c) \implies b \in {}_{\alpha_1+\alpha_2}NS_v(c)$$

Similar results holds for FS, NI, FI.

In particular when $\alpha_1 = \alpha_2 = \alpha$ and $\delta_1 = \delta_2 = \delta$ we have:

Corollary 1 (Transitivity and equivalence classes). *Consider three domain values a, b and c, for a variable v. Then,*

- *Threshold interchangeability is a transitive relation, and partitions the set of values for a variable into equivalence classes, that is*

$$b \in {}_{\alpha}NS_v(a), a \in {}_{\alpha}NS_v(c) \implies b \in {}_{\alpha}NS_v(c)$$
$${}_{\alpha}NI_v(b/a), {}_{\alpha}NI_v(a/c) \implies {}_{\alpha}NI_v(b/c)$$

- *If the $\times_S$-operator is idempotent, then degradation interchangeability is a transitive relation, and partitions the set of values for a variable into equivalence classes, that is*

$$b \in {}^{\delta}NS_v(a), a \in {}^{\delta}NS_v(c) \implies b \in {}^{\delta}NS_v(c)$$
$${}^{\delta}NI_v(b/a), {}^{\delta}NI_v(a/c) \implies {}^{\delta}NI_v(b/c)$$

By using degradations and thresholds we have a nice way to decide when two domain values for a variable can be substituable/interchangeable. In fact, by changing the α or δ parameter we can obtain different results.

In particular we can show that an extensivity results for the parameters hold. In fact, it is straightforward to notice that if two values are ${}^{\delta}_{\alpha}$substitutable, they have to be also ${}^{\delta'}_{\alpha'}$substitutable for any $\delta' \leq \delta$ and $\alpha' \geq \alpha$.

Theorem 5 (Extensivity for α and δ). *Consider two domain values a and b, for a variable v, two thresholds α and α' s.t. $\alpha \leq \alpha'$ and two degradations δ and δ' s.t. $\delta \geq \delta'$. Then,*

$$a \in {}^{\delta}NS_v(b) \implies a \in {}^{\delta'}NS_v(b) \text{ and } a \in {}_{\alpha}NS_v(b) \implies a \in {}_{\alpha'}NS_v(b)$$

Similar results holds for FS, NI, FI.

As a corollary when threshold and degradation are $\mathbf{0}$ or $\mathbf{1}$ we have some special results.

Corollary 2. *When $\alpha = \mathbf{0}$ and $\delta = \mathbf{1}$, we obtain the non approximated versions of NS. When $\alpha = \mathbf{1}$ and $\delta = \mathbf{0}$, all domain values are substitutable.*

$$\forall a, b,\ a \in {}_{\mathbf{0}}NS_v(b) \text{ and } a \in {}^{\mathbf{1}}NS_v(b) \iff a \in NS(b)$$
$$\forall a, b,\ a \in {}_{\mathbf{1}}NS_v(b) \text{ and } a \in {}^{\mathbf{0}}NS_v(b)$$

Similar results holds for FS, NI, FI.

3.3 Computing ${}^{\delta}/_{\alpha}$-substitutability/interchangeability

The result of Theorem 1 is fundamental since it gives us a way to approximate full substituability/interchangeability by neighbourhood substituability/interchangeability which is much less costly to compute.

The most general algorithm for neighborhood substituability/interchangeability in the soft CSP framework is to check for each pair of values whether the condition given in the definition holds or not. This algorithm has a time complexity exponential in the size of the neighbourhood and quadratic in the size of the domain (which may not be a problem when neighbourhoods are small).

Better algorithms can be given when the times operator of the semiring is idempotent. In this case, instead of considering the combination of all the constraint C_v involving a certain variable v, we can check the property we need (NS/NI and their relaxed versions ${}^{\delta}_{\alpha}NS/NI$) on each constraint itself.

Theorem 6. *Consider two domain values b and a, for a variable v, and the set of constraints C_v involving v. Then we have:*

$$\forall c \in C_v . c\eta[v := a] \leq_S c\eta[v := b] \implies b \in NS_v(a)$$
$$(\forall c \in C_v . c\eta[v := a] \geq \alpha \implies c\eta[v := a] \leq_S c\eta[v := b]) \implies b \in {}_{\alpha}NS_v(a)$$

If the times operator of the semiring is idempotent we also have:

$$\forall c \in C_v . c\eta[v := a] \times_S \delta \leq_S c\eta[v := b] \implies b \in {}^{\delta}NS_v(a)$$

By using Theorem 6 (and Corollary 1 for ${}^{\delta}/_{\alpha}NS$) we can find substitutable/interchangeable domain values more efficiently. Algorithm 2 shows an algorithm that can be used to find domain values that are Neighborhood Interchangeable. It uses a data structure similar to the *discrimination trees*, first introduced by Freuder in [12] . Algorithm 2 can compute different versions of

1: Create the root of the discrimination tree for variable v_i
2: Let $C_{v_i} = \{c \in C \mid v_i \in supp(c)\}$
3: Let $D_{v_i} = \{$the set of domain values d_{v_i} for variable $v_i\}$
4: **for all** $d_{v_i} \in D_{v_i}$ **do**
5: **for all** $c \in C_v$ **do**
6: execute Algorithm NI-Nodes(c, v, d_{v_i}) to build the nodes associated with c
7: Add $v_i, \{d_{v_i}\}$ to annotation of the last build node,
8: Go back to the root of the discrimination tree.

Algorithm 2: Algorithm to compute neighbourhood interchangeable sets for variable v_i.

neighbourhood interchangeability depending on the algorithm $NI-nodes$ used. Algorithm 3 shows the simplest version without threshold or degradation. The

1: **for all** assignments η_c to variables in $supp(c)$ **do**
2: compute the semiring level $\beta = c\eta_c[v_i := d_{v_i}]$,
3: **if** there exists a child node corresponding to $\langle c = \eta_c, \beta \rangle$ **then**
4: move to it,
5: **else**
6: construct such a node and move to it.

Algorithm 3: NI-Nodes(c, v, d_{v_i}) for Soft-NI.

algorithm is very similar to that defined by Freuder in [12], and when we consider the semiring for classical CSPs $S_{CSP} = \langle \{false, true\}, \vee, \wedge, false, true \rangle$ and all constraints are binary, it computes the same result. Notice that for each node we add also an information representing the cost of the assignment η_c.

When all constraints are binary, considering all constraints involving variable v is the same as considering all variables connected to v by a constraint, and our algorithm performs steps as that given by Freuder.

We can determine the complexity of the algorithm by considering that the algorithm calls $NI - Nodes$ for each $k - ary$ constraint exactly once for each value of each the k variables; this can be bounded from above by $k * d$ with d the maximum domain size. Thus, given m constraints, we obtain a bound of

$$O(m * k * d * O(Algorithm NI-nodes)).$$

The complexity of $Algorithm NI-nodes$ strictly depends on the size of the domain d and from the number of variables k involved in each constraint and is given as

$$O(Algorithm NI-nodes) = d^{k-1}.$$

For complete constraint graphs of binary constraints ($k = 2$), we obtain the same complexity bound of $O(n^2d^2)$ as Freuder in [12].

1: **for all** assignments η_c to variables in $supp(c)$ s.t. $\beta = c\eta_c[v_i := d_{v_i}]$ and $\alpha \leq_S \beta$ **do**
2: **if** there exists a child node corresponding to $\langle c = \eta_c, \beta \rangle$ **then**
3: move to it,
4: **else**
5: construct such a node and move to it.

Algorithm 4: NI-Nodes(c, v, d_{v_i}) for Soft $_\alpha NI$.

1: **for all** assignments η_c to variables in $supp(c)$ **do**
2: compute the semiring level $\beta = c\eta_c[v_i := d_{v_i}]$,
3: **if** there exists a child node corresponding to $\langle c = \eta_c, \beta', \bar{\beta} \rangle$ with $(\bar{\beta} \leq \beta) \wedge (\beta \times \delta \leq \beta')$ **then**
4: move to it and change the label to $\langle c = \eta_c, glb(\beta', \beta), \bar{\beta} + (\beta \times \delta) \rangle$,
5: **else**
6: construct the node $\langle c = \eta_c, \beta, \beta \times \delta \rangle$ and move to it.

Algorithm 5: NI-Nodes(c, v, d_{v_i}) for Soft $^\delta NI$.

Algorithms for the relaxed versions of NI are obtained by substituting different versions of Algorithm 3. For $_\alpha NI$, the algorithm needs to only consider tuples whose semiring value is greater than α, as shown in Algorithm 4. For $^\delta NI$, the algorithm needs to only consider tuples that can cause a degradation by more than δ, as shown in Algorithm 5. The idea here is to save in each node the information needed to check at each step $^\delta NS$ in both directions. In a semiring with total order, the information represent the "interval of degradation". As both algorithms consider the same assignments as Algorithm 3, their complexity remains unchanged at $O(d^{k-1})$.

4 An Example

Fig. 4 shows the graph representation of a CSP which might represent a car configuration problem. A product catalog might represent the available choices through a soft CSP. With different choices of semiring, the CSP of Fig. 4 can represent different problem formulations; a possible example follows:

Example 1 An optimization criterion might be the time it takes to build the car. Delay is determined by the time it takes to obtain the components and to

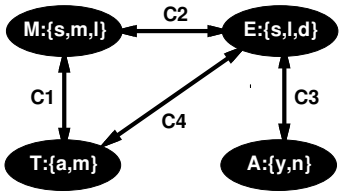

Fig. 4: Example of a CSP modeling car configuration. It has 4 variables: M = model, T = transmission, A = Air Conditioning, E = Engine.

reserve the resources for the assembly process. For the delivery time of the car, only the longest delay would matter. This could be modelled by the semiring $< \Re^+, min, max, +\infty, 0 >$[1], with the binary constraints:

$$C_1 = \begin{array}{cc|ccc} & & \multicolumn{3}{c}{M} \\ & & s & m & l \\ \hline T & a & \infty & 3 & 4 \\ & m & 2 & 4 & \infty \end{array} \quad C_2 = \begin{array}{cc|ccc} & & \multicolumn{3}{c}{M} \\ & & s & m & l \\ \hline & s & 2 & 3 & \infty \\ E & l & 30 & 3 & 3 \\ & d & 2 & 3 & \infty \end{array} \quad C_3 = \begin{array}{cc|ccc} & & \multicolumn{3}{c}{E} \\ & & s & l & d \\ \hline A & y & 5 & 4 & 7 \\ & n & 0 & 30 & 0 \end{array} \quad C_4 = \begin{array}{cc|ccc} & & \multicolumn{3}{c}{E} \\ & & s & l & d \\ \hline T & a & \infty & 3 & \infty \\ & m & 4 & 10 & 3 \end{array}$$

and unary constraints C_M, C_E, C_T and C_A that model the time to obtain the components:

$$C_M = \begin{array}{c|ccc} s & m & l \\ \hline 2 & 3 & 3 \end{array} \quad C_E = \begin{array}{c|ccc} s & l & d \\ \hline 3 & 2 & 3 \end{array} \quad C_T = \begin{array}{c|cc} a & m \\ \hline 1 & 2 \end{array} \quad C_A = \begin{array}{c|cc} y & n \\ \hline 3 & 0 \end{array}$$

△

Let us now consider the variable E of Example 1 and compute ${}^{\delta}/_{\alpha}NS/NI$ between its values by using Definition 4 and Definition 5. In Fig. 5 directed arcs are added when the source can be ${}^{\delta}/_{\alpha}$substituted to the destination node. It is easy to see how the occurrences of ${}^{\delta}/_{\alpha}NS$ change, depending on δ and α degrees.

We can notice that when δ takes value 0 (the **1** of the optimization semiring), small degradation is allowed in the CSP tuples when the values are substituted; thus only value s can be substituted for value d. As δ increases in value (or decreases from the semiring point of view) higher degradation of the solutions is allowed and thus the number of substitutabilities increase with it.

In the second part of Fig. 5 we can see that for $\alpha = 0$ all the values are interchangeable (in fact, since there are no solutions better than $\alpha = 0$, by definition all the elements are ${}_{\alpha}$interchangeable).

For a certain threshold ($\alpha = 4$) values s and d are ${}_{\alpha}$interchangeable and value l can substitute values s and d. Moreover, when α is greater than 5 we only have that s can substitute d.

We will show now how to compute interchangeabilities by using the Discrimination Tree algorithm. In Fig. 6 the Discrimination Tree is described for variable

[1] This semiring and the fuzzy one are similar, but the first uses an opposite order. Let us call this semiring *opposite-fuzzy*.

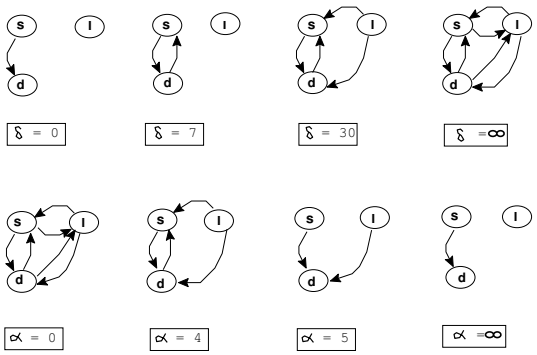

Fig. 5: Example of how δ-substitutability and α-substitutability varies in the opposite-fuzzy CSP over the values of variable E.

M when $\alpha = 2$ and $\alpha = 3$. We can see that values m and l for variable M are $_2$interchangeable whilst there are no interchangeabilities for $\alpha = 3$.

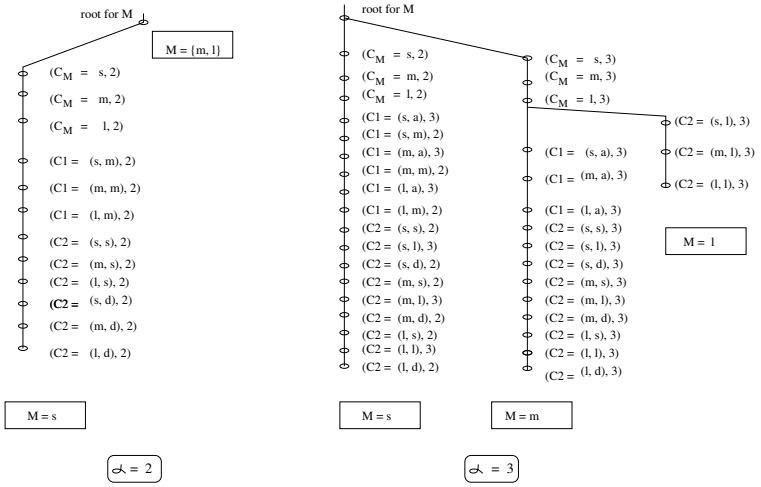

Fig. 6: Example of a search of α-interchangeability computing by the use of discrimination trees.

5 Conclusions

Interchangeability in CSPs has found many applications for problem abstraction and solution adaptation. In this paper, we have shown how the concept can be extended to soft CSPs in a way that maintains the attractive properties already known for hard constraints.

The two parameters α and δ allow us to express a wide range of practical situations. The threshold α is used to eliminate distinctions that would not interest us anyway, while the allowed degradation δ specifies how precisely we want to optimize our solution. We are now conducting a detailed investigation on how variation of these parameters affects interchangeability on random problems.

References

[1] Benson, B., Freuder, E.: Interchangeability preprocessing can improve forward checking search. In: Proc. of the 10th ECAI, Vienna, Austria. (1992) 28–30

[2] Bistarelli, S.: Soft Constraint Solving and programming: a general framework. PhD thesis, Dipartimento di Informatica, Università di Pisa, Italy (2001)

[3] Bistarelli, S., Fargier, H., Montanari, U., Rossi, F., Schiex, T., Verfaillie, G.: Semiring-based CSPs and Valued CSPs: Frameworks, properties, and comparison. CONSTRAINTS: An international journal. Kluwer **4** (1999)

[4] Bistarelli, S., Montanari, U., Rossi, F.: Constraint Solving over Semirings. In: Proc. IJCAI95, San Francisco, CA, USA, Morgan Kaufman (1995)

[5] Bistarelli, S., Montanari, U., Rossi, F.: Semiring-based Constraint Solving and Optimization. Journal of the ACM **44** (1997) 201–236

[6] Bistarelli, S., Montanari, U., Rossi, F.: Semiring-based Constraint Logic Programming: Syntax and Semantics. ACM Transactions on Programming Languages and System (TOPLAS) **23** (2001) 1–29

[7] Bistarelli, S., Montanari, U., Rossi, F.: Soft concurrent constraint programming. In: Proc. ESOP, 2002, Grenoble, France. LNCS, Springer-Verlag (2002)

[8] Choueiry, B.Y.: Abstraction Methods for Resource Allocation. PhD thesis, EPFL PhD Thesis no 1292 (1994)

[9] Choueiry, B.Y., Noubir, G.: On the computation of local interchangeability in discrete constraint satisfaction problems. In: Proc. of AAAI-98. (1998) 326–333

[10] Dubois, D., Fargier, H., Prade, H.: The calculus of fuzzy restrictions as a basis for flexible constraint satisfaction. In: Proc. IEEE International Conference on Fuzzy Systems, IEEE (1993) 1131–1136

[11] Fargier, H., Lang, J.: Uncertainty in constraint satisfaction problems: a probabilistic approach. In: Proc. European Conference on Symbolic and Qualitative Approaches to Reasoning and Uncertainty (ECSQARU). Volume 747 of LNCS., Springer-Verlag (1993) 97–104

[12] Freuder, E.C.: Eliminating interchangeable values in constraint satisfaction problems. In: Proc. of AAAI-91, Anaheim, CA (1991) 227–233

[13] Freuder, E., Wallace, R.: Partial constraint satisfaction. AI Journal **58** (1992)

[14] Haselbock, A.: Exploiting interchangeabilities in constraint satisfaction problems. In: Proc. of the 13th IJCAI. (1993) 282–287

[15] Neagu, N., Faltings, B.: Exploiting interchangeabilities for case adaptation. In: In Proc. of the 4th ICCBR01. (2001)

[16] Ruttkay, Z.: Fuzzy constraint satisfaction. In: Proc. 3rd IEEE International Conference on Fuzzy Systems. (1994) 1263–1268

[17] Schiex, T.: Possibilistic constraint satisfaction problems, or "how to handle soft constraints?". In: Proc. 8th Conf. of Uncertainty in AI. (1992) 269–275

[18] Schiex, T., Fargier, H., Verfaille, G.: Valued Constraint Satisfaction Problems: Hard and Easy Problems. In: Proc. IJCAI95, San Francisco, CA, USA, Morgan Kaufmann (1995) 631–637

[19] Weigel, R., Faltings, B.: Interchangeability for case adaptation in configuration problems. In: Proc. of the AAAI98 Spring Symposium on Multimodal Reasoning, Stanford, CA. (1998) TR SS-98-04.

[20] Weigel, R., Faltings, B.: Compiling constraint satisfaction problems. Artificial Intelligence **115** (1999) 257–289