Constraints, 8, 79–97, 2003
© 2003 Kluwer Academic Publishers. Manufactured in The Netherlands.

General Properties and Termination Conditions for Soft Constraint Propagation

S. BISTARELLI bista@di.unipi.it
CNR Pisa, Istituto per le Applicazioni Telematiche, Area della Ricerca di Pisa, Via G. Moruzzi 1, 56124 Pisa, Italy

R. GENNARI rosella.gennari@cwi.nl
Applied and Computational Logic Group, CWI & ILLC, Universiteit van Amsterdam, *Kruislaan 413, 1098 SJ, Amsterdam, The Netherlands*

F. ROSSI frossi@math.unipd.it
Università di Padova, Dipartimento di Matematica Pura ed Applicata, Via G. B. Belzoni 7, 35131 Padova, Italy

Abstract. Soft constraints based on semirings are a generalization of classical constraints, where tuples of variables' values in each soft constraint are associated to elements from an algebraic structure called semiring. This framework is able to express, for example, fuzzy, classical, weighted, valued and over-constrained constraint problems.

Classical constraint propagation has been extended and adapted to soft constraints by defining a schema for soft constraint propagation [8]. On the other hand, in [1–3] it has been proven that most of the well known constraint propagation algorithms for classical constraints can be cast within a single schema.

In this paper we combine these two schemas and we provide a more general framework where the schema of [3] can be used for soft constraints. In doing so, we generalize the concept of soft constraint propagation, and we provide new sufficient and independent conditions for its termination.

Keywords: soft constraints, soft constraint propagation, termination, semiring, generic algorithm schema

1. Introduction

Soft constraints allow to model faithfully many real-life problems, especially those which possess features like preferences, uncertainties, costs, levels of importance or absence of solutions. In fact, a soft constraint problem (SCSP) is just like a classical constraint problem (CSP), except that each instantiation for the variables of a soft constraint is associated with an element taken from a set (usually ordered). These elements directly represent the desired features, since they can be interpreted as levels of preference, costs, levels of uncertainty, etc.

There are many formalizations of soft constraint problems [8–11, 14]. In this paper we consider the one based on semirings [4, 8], where the semiring specifies the set of elements to be used to represent the desired features, the order over such a set, and the operation for combining constraints. This formalism has many interesting instances, like classical, fuzzy [9], weighted, valued [14], probabilistic [10] and partial [11] constraints.

The constraint propagation techniques usually used for classical CSPs have been extended and adapted to deal with soft constraints, provided that certain conditions are

met. This leads to an algorithm schema for soft constraint propagation, in which, at each step, a small subproblem is solved, as in classical constraint propagation [8]. By studying the properties of this schema, it turns out that such steps can be seen as applications of functions which are monotonic, inflationary and idempotent with respect to a certain partial ordering.

On an orthogonal line of research, the concept of constraint propagation over classical constraints has been studied in depth in [1–3] and a general algorithmic schema (called GI) for achieving constraint propagation has been devised. The GI algorithm can be instantiated to most of the existing constraint propagation algorithms, given a partial ordering with bottom and a finite set of functions [3].

By studying and comparing the semiring-based framework for soft constraints and the GI schema, one can notice that indeed the latter can be instantiated to most of the constraint propagation algorithms for soft constraints as well: we just need to define an appropriate partial order between soft constraint satisfaction problems and to find suitable functions.

Moreover, by analyzing the properties of the GI algorithm, one can see that soft constraint propagation can be enforced by means of functions which are not necessarily idempotent (as instead required in [8]). This is a double generalization: we do not need, like in [8], that each constraint propagation function solves a subproblem, nor that it is idempotent. This allows us to model several forms of "approximate" constraint propagation, which were instead not captured by the schema in [8]. Example are bounds-consistency for classical constraints [13] and partial soft arc-consistency for soft constraints [5].

Therefore, one can use the GI algorithm schema to provide a generalized form of soft constraint propagation. Our main quest, at this point, is to understand under what conditions the resulting schema terminates. In fact, if we work with classical constraints over finite domains, the GI algorithm always terminates if it is given monotonic and inflationary functions. However, when moving to soft constraints over a semiring, even if the variable domain is finite, the GI algorithm is not guaranteed to terminate, because the semiring universe can be infinite. For example, fuzzy constraints have a semiring universe that contains all real numbers between 0 and 1, and the semiring of weighted constraints contains either all the reals or all the natural numbers.

With that in mind, however, we can identify some new interesting sufficient conditions for the termination of the GI algorithm over soft constraints. The first, predictable, condition concerns the partial order over soft constraint satisfaction problems: if this partial order is well-founded (that is, it forbids ascending chains of infinite length), then the whole algorithm terminates if it uses monotonic and inflationary functions.

The second condition, instead, depends on the two semiring operations: when the constraint propagation functions are defined via these two operations, then we can just consider the subset of the semiring universe that is obtained by taking the elements determined by the given problem and combining them under the two semiring operations. If the restriction of the semiring order to such a set has all descending chains (that is, all sequences of elements such that each element is smaller than the previous one) with finite length, then the GI algorithm terminates.

Both these conditions are sufficient for termination. However, they could be difficult to check, unless the partial orders have a well-known structure. Nevertheless, in a special case we can formally prove that there always exists a well-founded set; thus we can automatically ensure termination. This special case is related to the idempotence of the semiring multiplicative operation, the one that is used to combine constraints: if this operation is idempotent, then GI terminates. For example, in classical constraints the multiplicative operation is Boolean conjunction and in fuzzy constraints it is the minimum operation, thus we can formally prove that the GI algorithm over *any* classical or fuzzy constraint satisfaction problem *always* terminates, provided that the functions are defined via the two semiring operations.

We believe that the generalizations and termination conditions that we have developed will make soft constraints more widely applicable, as well as soft constraint propagation more practically usable.

The paper is organized as follows. First, Section 2 introduces the semiring-based formalisms for soft constraints and soft constraint propagation via rules, as in [8]. Then, Section 3 defines some orders among semirings, constraints, and problems, to be used later for generalizing soft constraint propagation. Section 4 summarizes the main features of the GI algorithmic schema for classical constraint propagation. Then, Section 5 shows how to apply the GI algorithm to soft constraint problems, thereby generalizing soft constraint propagation via rules and obtaining a new algorithm schema for soft constraint propagation. Finally, in Section 6, we prove three conditions for the termination of such schema, and Section 7 concludes the paper.

This paper is an extended and revised version of [7]. In this version, all the statements are formally proved, and the termination conditions are studied in greater details. Examples in the SCSP section are also added to help the reader understand better the two frameworks considered.

2. Soft Constraints

In the semiring-based formalism for soft constraints [8], a soft constraint is just a constraint where each instantiation of its variables comes equipped with an associated value. Combining constraints has to take into account such additional values; thus the formalism has also to provide suitable operations for combination ($\times$) and comparison ($+$) of tuples of values for constraint variables. Here the notion of c-semiring proved to be useful [8].

Semirings and SCSPs

A *semiring* is a tuple $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ such that: A is a set and $\mathbf{0}, \mathbf{1} \in A$; $+$ is commutative, associative and $\mathbf{0}$ is its unit element; $\times$ is associative, distributes over $+$, $\mathbf{1}$ is its unit element and $\mathbf{0}$ is its absorbing element. Furthermore, some additional properties are added, leading to the notion of c-semiring: a *c-semiring* is a semiring $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ such that $+$ is idempotent with $\mathbf{1}$ as its absorbing element and $\times$ is commutative.

The partial order $\leq_S$ over the set A, defined as $a \leq_S b$ iff $a + b = b$, is used to compare elements in the semiring: $a \leq_S b$ means that b is "better" than a.

A *constraint system* is a tuple $CS = \langle S, D, V \rangle$ where S is a c-semiring, D is a finite set (the domain of the variables) and V is a finite ordered set of variables.

Given a semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ and a constraint system $CS = \langle S, D, V \rangle$, a *constraint* is a pair $\langle def, con \rangle$ where $con \subseteq V$ and def: $D^{|con|} \to A$. Therefore a constraint specifies an ordered set of variables, *con*, and assigns a semiring element to each instantiation for those variables. Intuitively, the assigned value to the instantiation specifies how much it is "preferred".

A *Soft Constraint Satisfaction Problem* (SCSP) on a constraint system CS is a pair $P = \langle C, con \rangle$, where $con \subseteq V$ and C is a set of constraints: *con* represents the set of variables of interest for the constraint set C, which however may contain constraints defined on variables not in *con*.

Combining and Projecting Soft Constraints

Given two constraints, $c_1 = \langle def_1, con_1 \rangle$ and $c_2 = \langle def_2, con_2 \rangle$, their *combination* $c_1 \otimes c_2$ is the constraint $\langle def, con \rangle$, defined by $con = con_1 \cup con_2$ and $def(t) = def_1(t \downarrow_{con_1}^{con}) \times def(t \downarrow_{con_2}^{con})$, where $t \downarrow_Y^X$ denotes the projection of tuple t from X onto Y. In other words, combining two constraints means building a new constraint involving all the variables of the original one; the new constraint associates to each tuple of domain values a semiring element that is obtained by multiplying the elements associated by the original constraints to the appropriate subtuples.

Given a constraint $c = \langle def, con \rangle$ and a subset I of V, the *projection* of c onto I, written as $c \Downarrow_I$, is the constraint $\langle def', con' \rangle$ where $con' = con \cap I$ and $def'(t') = \sum_{t | t \downarrow_{I \cap con}^{con} = t'} def(t)$. Thus, projecting means eliminating some variables. This is done by associating a semiring element with each tuple t over the remaining variables; that element is the sum of the elements associated by the original constraint with all the extensions of the tuple t over the eliminated variables.

In brief: combination is performed via the multiplicative operation of the semiring, whilst projection is achieved via the additive one.

Examples

Classical CSPs are SCSPs for which the chosen c-semiring is

$$Bool = \langle \{false, true\}, \vee, \wedge, false, true \rangle.$$

By means of *Bool* we can associate a Boolean value, either *false* or *true*, with each tuple of elements in D, then project and combine constraints via the Boolean connectives.

Fuzzy CSPs [9] can instead be modeled by choosing the c-semiring

$$Fuzzy = \langle [0, 1], max, min, 0, 1 \rangle.$$

In fact, each tuple has a value between 0 and 1; constraints are combined via the *min* operation and compared via the *max* operation. Figure 1 shows a fuzzy CSP. Variables

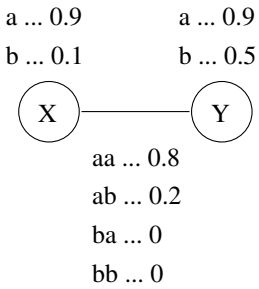

Figure 1. A fuzzy CSP.

are inside circles, constraints are represented by undirected arcs, domains are represented as unary constraints (with no arc), and semiring values are written to the right of the corresponding tuples. Here we assume that the domain of the variables contains only a and b, and that *con* contains all the variables.

Solutions

The *solution* of a SCSP $P = \langle C, con \rangle$ is the constraint $Sol(P) = (\bigotimes C) \Downarrow_{con}$: it is obtained by combining all constraints of P and then projecting over the variables in *con*. In this way, we get the constraint over *con* that is "induced" by the entire problem P.

For example, each solution of the fuzzy CSP of Figure 1 consists of a pair of domain values (that is, a domain value for each of the two variables) and an associated semiring element. Such an element is obtained by looking at the smallest value for all the subtuples (as many as the constraints) forming the pair. For example, for tuple $\langle a, a \rangle$ (that is, $x = y = a$), we have to compute the minimum between 0.9 (which is the value for $x = a$), 0.8 (which is the value for $\langle x = a, y = a \rangle$) and 0.9 (which is the value for $y = a$). Hence, the resulting value for this tuple is 0.8.

Two problems $P_1 = \langle C_1, con \rangle$ and $P_2 = \langle C_2, con \rangle$ (notice that they have the same *con*) are defined *equivalent*, written as $P_1 \equiv_P P_2$, if $Sol(P_1) = Sol(P_2)$.

Soft Constraint Propagation

Most of the traditional constraint propagation (also called *local consistency*) algorithms can be extended to SCSPs [8]. For this purpose, the notion of *constraint propagation rule* was introduced in [8]. The application of one of such rules consists of solving a subproblem of the given problem.

To model that, we use the notion of *typed location*. Informally, a typed location is just a location l (as in ordinary store-based programming languages) which has a set of variables *con* as type, thus can only be assigned a constraint $c = \langle def, con \rangle$ with the same type. In the following we assume to have a location for every set of variables; hence

we identify a location with its type. Given an SCSP $P = \langle C, con\rangle$, the *value* $[l]_P$ *of the location* l in P is defined as the constraint $\langle def, l\rangle \in C$ if it exists, as $\langle 1, l\rangle$ otherwise. Given n locations $l_1, \ldots, l_n$, the value $[\{l_1, \ldots, l_n\}]_P$ of this set of locations in P is defined as the set of constraints $\{[l_1]_P, \ldots, [l_n]_P\}$.

An *assignment* is a pair $l := c$ where $c = \langle def, l\rangle$. Given an SCSP $P = \langle C, con\rangle$, the result of the assignment $l := c$ is the problem $[l := c](P)$ defined as follows: $[l := c](P) := \langle\{\langle def', con'\rangle \in C \mid con' \neq l\} \cup c, con\rangle$.

Thus an assignment $l := c$ is seen as a function from SCSPs to SCSPs that modifies a given problem by changing just one constraint, namely the one with type l. The change consists in substituting such a constraint with c. If there is no constraint of type l, then the constraint c is added to the given problem. In other words, the assignment $l := c$ in P produces a new problem P' which is the same as P, except that it has an additional constraint c over the variables in l, and that the old constraints over l are removed. Note also that, by setting $|l| = 1$, we are able to modify domains, since a domain can be characterized as a unary constraint.

Consider a constraint system $CS = \langle S, D, V\rangle$, a location l and a set of locations L, where $l \in L$; a *constraint propagation rule* r_l^L is a function r_l^L which, taken any problem P over CS, returns $r_l^L(P) = [l := Sol(\langle [L]_P, l\rangle)](P)$.

Intuitively, the application of r_l^L to P adds the constraint $Sol(\langle [L]_P, l\rangle)$ over the variables in l to P. That constraint, by definition of Sol, is obtained by combining all the constraints of P that are identified by L and then projecting the resulting constraint over l.

Since a constraint propagation rule is a function from problems to problems, the application of a sequence S of rules to a problem is easily provided by function composition: we write $[r; S](P) = [S]([r_1](P))$ and mean that the problem $[r; S](P)$ is obtained applying first the rule r and then the rules of the sequence S, in the specified order.

An infinite sequence T of rules of a set R is called a *strategy*. A strategy is *fair* if each rule of R occurs in it infinitely often.

We are now ready to define constraint propagation algorithms by means of rules, as in [8]. Given a problem P, a set of rules R and a fair strategy T for R, a *constraint propagation algorithm via rules* applies to P the rules in R in the order prescribed by T. The algorithm terminates when the current problem is a fixed point of all the functions from R. In that case, we write $cp(P, R, T)$ to denote the resulting problem.

3. Some Useful Orders for Semiring-Based Constraints

We now introduce some orders: among semiring elements, as in [8], and new ones among constraints, constraint sets and problems.

In general, consider any partial ordering $\langle D, \sqsubseteq\rangle$ and the component-wise ordering $\langle D^n, \sqsubseteq_n\rangle$, with $n \geq 1$, where $\langle d_1, \ldots, d_n\rangle \sqsubseteq_n \langle d'_1, \ldots, d'_n\rangle$ iff $d_i \sqsubseteq d'_i$ for each $i = 1, \ldots, n$. Let f be a function from D^n to D. Then:

— f is *monotonic* iff $\langle d_1, \ldots, d_n\rangle \sqsubseteq_n \langle d'_1, \ldots, d'_n\rangle$ implies $f(\langle d_1, \ldots, d_n\rangle) \sqsubseteq f(\langle d'_1, \ldots, d'_n\rangle)$;

— f is *inflationary*, w.r.t. $\sqsubseteq$, iff $d_i \sqsubseteq f(\langle d_1, \ldots, d_n\rangle)$ for every $i = 1, \ldots, n$.

3.1. Semiring Order

All the orderings that we consider in this section are derived from the partial order $\leq_S$ between semiring elements, which, we recall, is defined as follows: $a \leq_S b$ iff $a + b = b$. The latter condition intuitively means that b is "preferred" to a.

It is easy to see that the following results hold, see also [8]:

— $\leq_S$ is a partial order;

— $\mathbf{0}$ is the bottom and $\mathbf{1}$ is the top of the order;

— if $\times$ is idempotent, then $\langle A, \leq_S \rangle$ is a distributive lattice where $+$ is the lub and $\times$ is the glb;

— $+$ and $\times$ are monotonic with respect to $\leq_S$;

— $+$ is inflationary with respect to $\leq_S$;

— $\times$ is inflationary with respect to $\geq_S$.

3.2. Constraint Order

Given the ordering $\leq_S$ over A, we can define a new order between constraints, as follows.

Definition 1. Consider $c_1 :=< def_1, con >$ and $c_2 :=< def_2, con >$, two constraints over the same constraint system CS. We write $c_1 \sqsubset_S c_2$ iff the following conditions are both satisfied:

1. for all tuples $s \in D^{|con|}$, $def_2(s) \leq_S def_1(s)$;
2. there exists (at least) a tuple $t \in D^{|con|}$ for which $def_2(t) <_S def_1(t)$.

We write $c_1 \sqsubseteq_S c_2$ iff only the first condition is requested.

Therefore, the constraint c_1 is smaller than or equal to c_2 in the order $\sqsubseteq_S$ iff the former assigns, to each variable instantiation, a greater value w.r.t. $\leq_S$ than c_2, or equal to it. Loosely speaking, c_2 is preferred to c_1 iff the former constraint is possibly "more restrictive" than the latter.

Theorem 1 ($\sqsubseteq_S$ is a po). *Given a semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ and a constraint system $CS = \langle S, D, V \rangle$, the relation $\sqsubseteq_S$ is a partial order between constraints over CS.*

Proof: We need to demonstrate that $\sqsubseteq_S$ is a reflexive, antisymmetric and transitive relation. Reflexivity holds trivially. To prove antisymmetry, suppose that $c_1 \sqsubseteq_S c_2$ and $c_2 \sqsubseteq_S c_1$; this yields that $con_1 = con_2$. Now, for all $t \in D^{|con_1|}$, we have both $def_1(t) \leq_S def_2(t)$ and $def_2(t) \leq_S def_1(t)$; hence $def_1(t) = def_2(t)$ and so $c_1 = c_2$. The transitivity of $\sqsubseteq_S$ follows from the transitivity of $\leq_S$. ■

3.3. Constraint Set Order

We can now easily extend the order $\sqsubseteq_S$ over constraints to a new order over *constraint sets* as follows; this is a typical construction in set theory and the resulting relation is called *Schmyd order*.[1]

Definition 2. Consider two sets of constraints, C_1 and C_2, over a constraint system CS. Suppose, furthermore, that $C_1 = \{c_i^1 : i \in I\}$, $C_2 = \{c_j^2 : j \in J\}$, $I \subseteq J$ and that, for each $i \in I$, the relation $c_i^1 \sqsubseteq_S c_i^2$ holds. Then we write $C_1 \sqsubseteq_C C_2$.

The intuitive reading of $C_1 \sqsubseteq_C C_2$ is that C_2 is a problem generally "more constraining" than C_1 is, because C_2 has (possibly) a larger number of "more restrictive" constraints than C_1 has.

Theorem 2 ($\sqsubseteq_C$ is a partial order). *Given a semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$, and a constraint system $CS = \langle S, D, V\rangle$, we have the following results*:

- *the relation $\sqsubseteq_C$ is a partial order between constraint sets over CS*;
- *the bottom of the relation is the empty set itself.*

Proof: We only prove the first claim, the other one being straightforward. Reflexivity trivially holds. As far as antisymmetry is concerned, suppose that $C_1 = \{c_i^1\}_{i\in I}$, $C_2 = \{c_j^2\}_{j\in J}$ and both $C_1 \sqsubseteq_C C_2$ and $C_2 \sqsubseteq_C C_1$ hold; this means that $I = J$. Moreover, the following relations hold for every $i \in I$: $c_i^1 \sqsubseteq_S c_i^2$ and $c_i^2 \sqsubseteq_S c_i^1$. Hence $c_i^1 = c_i^2$ for every $i \in I$, because $\sqsubseteq_S$ is a partial order relation, see Theorem 1. Transitivity follows similarly, by exploiting the transitivity of $\sqsubseteq_S$. ■

3.4. Problem Order

So far, we have introduced two partial orders: one between constraints ($\sqsubseteq_S$) and another one between constraint sets ($\sqsubseteq_C$). However, constraint propagation algorithms take constraint problems as input; therefore, we need an order relation between problems if we want to enforce soft constraint propagation by means of the `GI` algorithm (see later in Section 5).

Definition 3. Given a constraint system CS, consider two problems over it, say $P_1 = \langle C_1, con\rangle$ and $P_2 = \langle C_2, con\rangle$. We write $P_1 \sqsubseteq_P P_2$ iff $C_1 \sqsubseteq_C C_2$.

Now, we are able to define a partially ordered structure that contains all of the SCSPs that could be "generated" by constraint propagation, starting from a given input problem.

Definition 4. Consider a constraint system CS and an SCSP P over it. The *upward closure of P*, briefly $P\uparrow$, is the class of all problems P' on CS such that $P \sqsubseteq_P P'$.

The role of the upward closure of an SCSP during constraint propagation will be more clear after formalizing constraint propagation in our setting; see Definition 5 later in this paper.

Theorem 3 ($\sqsubseteq_P$ is a po). *Given a constraint system $CS = \langle S, D, V\rangle$ and a problem P on it, we have*:

- *the relation $\sqsubseteq_P$ is a partial order between SCSPs on CS*;
- *in particular, $\langle P\uparrow, \sqsubseteq_P\rangle$ is a partial ordering*;
- *the bottom of $\langle P\uparrow, \sqsubseteq_P\rangle$ is P itself.*

Proof: We prove the first claim; the other ones following immediately from it and the definition of $P\uparrow$. As usual, we only prove that the relation is antisymmetric, because transitivity can be proved similarly and reflexivity trivially holds. Hence, suppose that both $P_1 \sqsubseteq_P P_2$ and $P_2 \sqsubseteq_P P_1$ hold. This means that we have the following relations: $con_1 = con_2$, $C_1 \sqsubseteq_C C_2$, $C_2 \sqsubseteq_C C_1$. From the two last relations and Theorem 2, it follows that $C_1 = C_2$; hence $P_1 = P_2$. ■

3.5. Order-Related Properties of Soft Constraint Propagation Rules

We recall, from Section 2, that two problems P_1 and P_2 on the same set of variables are equivalent if they have the same solution set; if that is the case, we write $P_1 \equiv_P P_2$.

Now we can list some useful properties of soft constraint propagation rules (see [8] for the details), which are related to equivalence and to our new problem order. Here we assume that a constraint system CS and a rule r on CS are given.

— (Equivalence) $P \equiv_P r(P)$ if $\times$ is idempotent.

— (Inflationarity) $P \sqsubseteq_P r(P)$. This means that the new semiring values computed by means of a rule are either smaller than or equal to the input ones with respect to $\leq_S$.

— (Monotonicity) Consider two SCSPs, $P_1 = \langle C_1, con\rangle$ and $P_2 = \langle C_2, con\rangle$, over CS. If $P_1 \sqsubseteq_P P_2$, then $r(P_1) \sqsubseteq_P r(P_2)$.

It is easy to prove that all the results about constraint propagation rules hold also for a whole constraint propagation algorithm, which applies several rules in a sequence. Moreover, the strategy (that is, the order in which such rules are applied), if fair, does not influence the result, see [8].

4. The Generic Iteration Algorithm

The Generic Iteration (GI) algorithm of [3] is applied to compute the least fixed point of finitely many functions that are defined on a partial ordering with bottom. The GI algorithm is used as a schema for classical constraint propagation: in fact, most of the existing constraint propagation algorithms are instances of that schema.[2]

Given a partial ordering with bottom, say $\langle D, \sqsubseteq, \bot\rangle$, consider now a set of functions $F := \{f_1, \ldots, f_k\}$ on D. The following algorithm can compute the least common fixed point of the functions in F.

GENERIC ITERATION ALGORITHM (GI)

$d := \bot$;
$G := F$;
while $G \neq \varnothing$ **do**
  choose $g \in G$;
  $G := G - \{g\}$;
  $G := G \cup update(G, g, d)$;
  $d := g(d)$
od

For all G, g, d, the set of functions $update(G, g, d)$ from F needs satisfy the following conditions:

A. $\{f \in F - G \mid f(d) = d \wedge f(g(d)) \neq g(d)\} \subseteq update(G, g, d)$;

B. $g(d) = d$ implies $update(G, g, d) = \varnothing$;

C. $g(g(d)) \neq g(d)$ implies $g \in update(G, g, d)$.

Assumption **A** states that $update(G, g, d)$ at least contains all the functions from F–G for which d is a fixed point but $g(d)$ is not. So, at each loop iteration, such functions are added to the set G. In turn, assumption **B** states that no functions are added to G in case the value of d did not change. Note that, even though after the assignment $G := G - \{g\}$ we have $g \in F - G$, still $g \notin \{f \in F - G \mid f(d) = d \wedge f(g(d)) \neq g(d)\}$ holds. So assumption **A** does not provide any information when g is to be added back to G. This information is provided in assumption **C**. On the whole, the idea is to keep in G at least all functions f for which the current value of d is not a fixed point.

Hereafter, we recall parts of Theorem 2.4 in [3] that show the role of monotonicity and inflationarity:

1. Suppose that all functions in F are monotonic. Then every terminating execution of the GI algorithm computes in d the least common fixpoint of all the functions from F.
2. Suppose that all functions in F are inflationary and that D is finite. Then every execution of the GI algorithm terminates.

5. Soft Constraint Propagation via the GI Algorithm

Our goal in this section is to extend soft constraint propagation via rules (as defined in Section 2), by means of the GI schema. In this way, we aim at obtaining a new schema which is more general, in the sense that it can be applied both to classical and to soft constraint satisfaction problems, and it can perform constraint propagation steps which are more general than the rules of Section 2.

More precisely, first we combine the two formalisms described so far, namely the semiring framework for soft constraints and the GI algorithm schema for classical constraints. Then, we show that any constraint propagation algorithm via rules is an instance of GI algorithm, when applied to SCPSs.

5.1. GI *for Generalized Soft Constraint Propagation*

While constraint propagation rules are idempotent (since they solve a subproblem), the GI algorithm does not require this property of its functions. This means that we can generalize constraint propagation via rules for soft constraints, by dropping idempotence, as follows.

Definition 5. Consider a constraint system CS and the set $\mathrm{Pr}(CS)$ of all CS problems. A *constraint propagation function* is a function $f : \mathrm{Pr}(CS) \to \mathrm{Pr}(CS)$ that satisfies the following conditions:

1. if P is a problem over a variable set con, so is $f(P)$;
2. f is monotonic and inflationary over $\sqsubseteq_P$.

Observe that, if f is a constraint propagation function and P is a problem on CS, then $P\uparrow$ is *closed under constraint propagation functions*, i.e. $f : P\uparrow \to P\uparrow$. This is the content of the following proposition. Incidentally, its proof brings light on the role of inflationarity in "generating" problems that are always "higher" in the order than the input one P, w.r.t $\sqsubseteq_P$; hence it also provides motivations to our Definition 4 of upward closure of P.

Proposition 1. *Let $P := \langle C, con\rangle$ be a problem on a constraint system CS. If f is a constraint propagation function, then $f : P\uparrow \to P\uparrow$.*

Proof: We need to prove that, if $P' \in P\uparrow$, then $f(P')$ belongs to $P\uparrow$ too; namely that $P \sqsubseteq_P f(P')$. By the first item in Definition 5, we know that P' and $f(P')$ are over the same variable set. Furthermore, $P' \in P\uparrow$ means that $P \sqsubseteq_P P'$; this and f inflationary yield that $P \sqsubseteq_P f(P')$. ■

With this definition of constraint propagation function we relax two conditions of a constraint propagation rule (w.r.t. the definition of a rule in Section 2):

— that it yields the solution of a subproblem;

— that it is idempotent.

The second generalization was suggested by the results in [2] concerning the GI algorithm; in fact, as we state in the following for SCSPs, idempotence is not needed for GI to compute the least fixed point and hence to enforce constraint propagation, given in input constraint propagation functions. Moreover, many practical constraint propagation algorithms do not exactly solve subproblems, but generate an approximation of the solution. Consider, for example, the definition of *bounds consistency* in [13] or the notion of partial soft arc-consistency in [5]. Thus, the extension of GI to SCSPs allows to include as instances more practical algorithms. The following result is a trivial consequence of Theorem 2.4 in [3], extended to well-founded orderings, see also the first item of Section 4.

Proposition 2 (GI for SCSPs). *Given a constraint system CS and an SCSP problem P on it, let us apply the* GI *algorithm to the partial order $\langle P\uparrow, \sqsubseteq_P\rangle$, with a finite set R of*

constraint propagation functions. Then every terminating execution of the GI *algorithm computes in the output problem P' the least common fixed point of all the functions from R.*

5.2. GI *for Constraint Propagation via Rules Over Soft Constraints*

The functions that GI needs in input are defined on a partial ordering with bottom. In the case of constraint propagation rules for SCSPs, the partial ordering is $\langle P\uparrow, \sqsubseteq_P\rangle$, and the bottom is the problem P itself; see Theorem 3.

Constraint propagation rules are monotonic and inflationary functions. Thus the GI algorithm can be used to perform constraint propagation over soft constraint problems via rules. Even more, a soft constraint propagation algorithm via rules and the GI algorithm, with the same constraint propagation rules as input functions, compute the same result, namely the least common fixed point of the rules. Hence, the following result is a corollary of Theorem 2 and the properties of constraint propagation algorithms via rules quoted in Section 2.

Corollary 1 (GI with soft constraint propagation rules). *Given an SCSP P over a constraint system CS, consider a constraint propagation algorithm via rules in R to P according to a fair strategy T; call it AlgR. Consider also the partial order $\langle P\uparrow, \sqsubseteq_P\rangle$, and the set of functions R, and apply the* GI *algorithm to such input. Then* GI *terminates iff AlgR does; in such case, the output is the least fixed point of all the functions from R.*

The previous corollary states that the GI algorithm schema over SCSPs is "at least" as expressive as the constraint propagation algorithm schema via rules. In the previous section, we have shown that GI is actually more expressive (i.e., it can be instantiated to more constraint propagation algorithms over soft constraints) than the other algorithm schema.

What is now important to investigate is the issue of termination for GI with SCSPs and constraint propagation functions. This is not an easy task in case of SCSPs; in fact, even if the variable domain is finite, the semiring may be infinite, which is obviously a source of possible non-termination.

6. Termination of the GI Algorithm for SCSPs

As already noted, the presence of a possibly infinite semiring universe may lead to a constraint propagation algorithm which does not always terminate. In this section we focus on this problem and investigate under which conditions the GI schema terminates.

The first condition is a predictable extension of the one in the second item of Section 4: instead of requiring the finiteness of the domain of computation, we just require that its ascending chains have finite length. This guarantees the termination of the GI schema, because the application of each local consistency function to a problem P results in a problem that is greater than or equal to P; thus each computation follows an ascending chain, and therefore by assumption it cannot be infinite.

Theorem 4 (termination 1). *Given a constraint system CS and an SCSP problem P on it, let us instantiate the* GI *algorithm with* $\langle P\uparrow, \sqsubseteq_P\rangle$ *and a finite set R of constraint propagation functions. Suppose that the order* $\sqsubset_{CS}$*, restricted to* $P\uparrow$*, satisfies the Ascending Chain Condition* (*ACC*): *namely, each ascending chain of problems in* $P\uparrow$

$$P_0 \sqsubset_P \cdots \sqsubset_P P_n \sqsubset_P P_{n+1} \sqsubset_P \cdots$$

is finite. Then every execution of the GI *algorithm terminates.*

Proof: It follows from Theorem 2.4 in [3]. There, the author says to need a finite (and not only satisfying the ACC condition) order, but the proof does not actually use this restriction. ■

The previous theorem can be used to prove termination in many cases. For example:

— Classical constraints over finite domains always generate a finite problem order, which thus trivially satisfies the ACC property. Therefore the above theorem guarantees termination.

— Another example occurs when dealing with weighted soft constraints: here the semiring is $\langle N, min, +, 0, +\infty\rangle$; thus we have an infinite problem ordering which however satisfies the ACC property. Thus algorithm GI over such soft constraints always terminates.

However, there are also many interesting cases in which $\langle P\uparrow, \sqsubset_P\rangle$ does not necessarily satisfy the ACC property. For example, in the case of fuzzy CSPs, the semiring is $\langle[0, 1], max, min, 0, 1\rangle$. Consider the fuzzy constraint problem with variable domain $D = \{0, 1\}$, variables x and y, and which has just the trivial constraint $c = \langle 1, \{x, y\}\rangle$, where 1 is the function that assigns value 1 to each possible instantiation of x and y in D. Then $P\uparrow$ is the class of all problems on x and y, and it is easy to see that the problem order does not satisfy the ACC. In fact, it is enough to consider, for instance, the infinite chain of problems P_n on x, y for which each problem P_n has the two variables x and y and just one constraint, say c_n, which assigns to each possible instantiation of x and y the value $1/n$, with n a natural number greater than 0. Therefore, the above theorem cannot be applied to assure termination in this case. A similar argument applies to probabilistic CSPs, that is, SCSPs where each constraint has a certain probability to be in the real problem, which are modelled via the semiring $\langle[0, 1], max, \times, 0, 1\rangle$.

Thus Theorem 4 does not help us if we deal with either fuzzy or probabilistic CSPs. However, even in those cases where the order has chains of infinite length, we can observe the following: if we restrict our attention to constraint propagation functions defined via $+$ and $\times$, we can provide another condition on the input problem that guarantees the termination of the GI algorithm. This condition depends on the two semiring operations, $+$ and $\times$, and on their properties. To state such a condition, we first need to introduce the notion of semiring closure of a soft constraint problem.

Definition 6 (semiring closure). Consider a constraint system CS, an SCSP P over it and the set of semiring values that appear in P: namely,

$$Values(P) = \bigcup_{\langle def', con' \rangle \in C} \{def'(d) \mid d \in D^{|con'|}\}.$$

Then the *semiring closure* of P is the smallest set B that satisfies the following conditions:

1. $Values(P) \subseteq B \subseteq A$;
2. B is closed with respect to $+$ and $\times$.

Then we write $\overline{P}^S$ for the semiring closure of P.

Notice that the previous definition is meaningful, since there is always a set that satisfies the above conditions 1 and 2, namely A itself. In particular, the semiring closure of a problem P contains all those values of A which are in the problem P, and all their possible combinations via the two semiring operations.

What we need now is to single out those constraint propagation functions that, applied to a problem P, compute values that are in the semiring closure of P. Such functions, intuitively, are defined via the two semiring operations, $+$ and $\times$; in fact, given in input a SCSP P, such functions will return values that either are in $Values(P)$ or obtained from elements of $Values(P)$ via applications of $+$ or $\times$. Hence, the following definition captures the intuitive meaning of what we informally call *functions defined via the semiring operations*.

Definition 7 (closure-small functions). Consider a constraint system CS with semiring $S = \langle A, +, \times, 0, 1 \rangle$, an SCSP P on CS and a constraint propagation function $g : P \uparrow \rightarrow P \uparrow$. Function g is *closure-small* iff, for every $P' \in P \uparrow$, $\overline{g(P')}^S \subseteq \overline{P'}^S$.

We can now prove that, if constraint propagation functions are defined via $+$ and $\times$ (i.e., are closure-small), and if $<_S$ is "well behaved" when it is restricted to $\overline{P}^S$, then constraint propagation terminates. The result is more precisely stated in the Theorem 5. But first we need the following technical lemma.

Lemma 1. *Consider a constraint system CS with semiring $S = \langle A, +, \times, 0, 1 \rangle$, an SCSP P on CS and a finite set of constraint propagation functions R which are all closure-small. Assume also that the semiring order $<_S$ satisfies the Descending Chain Condition (DCC), when it gets restricted to $\overline{P}^S$: namely, there are no infinite descending $<_S$-chains of elements of $\overline{P}^S$. Let $P^S \uparrow$ be the smallest (w.r.t. set inclusion) of all subclasses $\mathcal{P}$ of $P \uparrow$ that satisfy the following conditions*:

1. $P \in \mathcal{P}$;
2. $P' \in \mathcal{P}$ *and* $g \in R$ *implies that* $g(P') \in \mathcal{P}$.

Then the restriction of $\sqsupset_P$ to $P^S \uparrow$ satisfies the DCC.

Proof: Suppose that there is an infinite chain of $P^S \uparrow$ sets,

$$P_0 \sqsubset_P \cdots \sqsubset_P P_n \sqsubset_P P_{n+1} \sqsubset_P \cdots \tag{1}$$

where, for each $n \geq 1$, $P_{n+1} := g_n(P_n)$, for some $g_n \in R$. We prove that an infinite $\overline{P}^S$ descending $<_S$-chain can be extracted from (1), contradicting our assumption on $<_S$. In fact, the fact that $P_{n+1} = g_n(P_n)$ implies that, for each $n \in \omega$, there is a constraint $c_n \in P_n$, on some set of variable *con*, that is replaced by $c_{n+1} \in P_{n+1}$, such that $c_n \sqsubset_S c_{n+1}$. As the chain in (1) is infinite while P_0 has only finitely many constraints, there must exist a constraint $c \in P_0$ that is modified infinitely many times in the chain (1); hence, we can extract an infinite chain of constraints on *con* from (1):

$$c_0 \sqsubset_S \cdots \sqsubset_S c_m \sqsubset_S c_{m+1} \sqsubset_S \cdots \tag{2}$$

Now, $c_m \sqsubset_S c_{m+1}$ means that there exists a tuple $t \in D^{|con|}$ such that

$$\mathrm{def}_m(t) >_S \mathrm{def}_{m+1}(t); \tag{3}$$

while, for all other tuples s of $D^{|con|}$, we have, at least, that $def_m(s) \geq_S def_{m+1}(s)$. As the variable domain D is finite while the chain in equation (2) is infinite, there must be a tuple $t \in D^{|con|}$ for which the strict inequality in (3) holds for infinitely many $m \in \omega_0$. Therefore, we can extract an infinite chain of semiring elements from (2):

$$\mathrm{def}_0(t) >_S \cdots >_S \mathrm{def}_k(t) >_S \mathrm{def}_{k+1}(t) >_S \cdots \tag{4}$$

All semiring elements that occur in (4) belong to $\overline{P}^S$, because all the functions of R are closure-small. Therefore, the restriction of the order $<_S$ to the set $\overline{P}^S$ does not satisfy the DCC, which is absurd because of our hypothesis. ■

Theorem 5 (termination 2). *Consider a constraint system CS with semiring* $S = \langle A, +, \times, 0, 1 \rangle$, *an SCSP P on CS and a finite set of constraint propagation functions R defined via* $+$ *and* $\times$ (*i.e., closure-small*). *Assume, also, that the semiring order* $<_S$ *satisfies the Descending Chain Condition* (*DCC*) *when restricted to* $\overline{P}^S$: *namely, there are no infinite descending* $<_S$*-chains of* $\overline{P}^S$ *elements. Then every execution of the* GI *algorithm terminates.*

Proof: Let us consider the Cartesian product $P^S \uparrow \times \mathbf{N}$, where $P^S \uparrow$ is as defined in Lemma 1, and the lexicographical order $<_{lex}$ on it that is defined as follows:

$$(D', n') <_{lex} (D, n) \quad \text{iff} \quad D' \sqsupset_P D \quad \text{or} \quad D = D' \quad \text{and} \quad n' <_\omega n,$$

where $<_\omega$ is the natural order. Consider a **while**-loop of the GI algorithm: let g be the chosen function and G_i the set of functions under consideration, before removing g, and G_o the set of functions at the end of the **while**-loop; furthermore, let P_i be the input problem and P_o the problem $g(P_i)$. At the end of the **while**-loop, two cases are possible: either P_i is not a fixed point of g and, hence, $P_o \sqsupset_P P_i$ by definition of local consistency

functions (that are inflationary w.r.t. $\sqsubseteq_P$); or P_i is a fixed point of g and so $P_o = P_i$ and $G_o = G_i - \{g\}$. In both cases, we have that

$$(P_o, |G_o|) <_{lex} (P_i, |G_i|).$$

As both orders $<_\omega$ and $\sqsupset_P$ satisfy the DCC (see Lemma 1), so does the lexicographical order $<_{lex}$; hence the GI algorithm terminates. ■

If we have a fuzzy constraint problem, then $\overline{P}^S$ is precisely the set of all semiring values appearing in the initial problem; i.e., it coincides with the set *Values*(P). In fact, the latter is closed with respect to *min* and *max*, which are the two semiring operations in the case of fuzzy CSPs. Moreover, the semiring order restricted to *Values*(P) satisfies the DCC, since it is finite. Thus Theorem 5 can be used to prove that constraint propagation (where each step uses only the two semiring operations) over fuzzy CSPs always terminates.

Another example is constraint optimization over the reals: if the initial problem only contains natural numbers, then $\overline{P}^S$ is a set of natural numbers whose order satisfies the DCC and which is closed w.r.t. $+$ (*min*) and $\times$ (*sum*). Thus again Theorem 5 can be used to prove that constraint propagation over such problems terminates.[3]

Notice that the previous theorem is similar to the one in [8] that concerns termination; however, in [8], the set $\overline{P}^S$ is forced to be finite in order to guarantee the termination of a local consistency algorithm (see Theorem 4.14 of [8]); a hypothesis that is much stronger than ours in Theorem 5.

It is now interesting to investigate the relationship between the hypotheses of Theorem 4 and 5, which, as we will now show, are independent.

Let us consider the following two cases:

1. (Bounds consistency): as shown in [2], the GI schema can be instantiated to the algorithm for enforcing bounds consistency on classical CSPs. In order to do so, it is necessary to adopt functions that exploit the variable domain structure; in fact, the semiring operations alone are not sufficient to define the needed constraint propagation functions. Hence Theorem 5 cannot be applied in this case; however, Theorem 4 can be applied and it is sufficient to guarantee termination.
2. (Fuzzy CSPs): as already shown in the example above, there are problem orderings which do not satisfy the ACC in case of fuzzy CSPs; hence, Theorem 4 is not always applicable in that framework. However, as observed above, Theorem 5 can be applied to prove termination.

Yet, even Theorem 5 may be difficult to apply. In fact, it might not always be trivial to compute the semiring closure of a given SCSP P; furthermore, we should also check that the restriction of the semiring order to the closure satisfies the DCC. There is however a special case in which we do not have to search for such a set. This fortunate case occurs when the multiplicative operation of the semiring is idempotent. In fact, in that case, we can easily construct the semiring closure of any given problem and that closure turns out to be always finite (hence satisfying the DCC). Indeed, this is an highly desirable property, because it provides an easy way to guarantee the termination of the GI algorithm schema.

Theorem 6 (termination 3). *Consider a constraint system CS, an SCSP P on CS and a finite set of constraint propagation functions R defined via* $+$ *and* $\times$ (*i.e., closure-small*). *Assume, also, that* $\times$ *is idempotent. Then, every execution of the* GI *algorithm terminates.*

Proof: Consider the *finite* set $Values(P)$, which, we recall, is the set of all semiring elements appearing in P. Since a c-semiring with $\times$ idempotent is also a distributive lattice (cf. [8]), the set generated from a finite set like $Values(P)$ via $+$ and $\times$ is itself finite. In fact, due to the distributivity, each element of such a set can be denoted by a sum of products of subsets of $Values(P)$. Since $Values(P)$ is finite, also the number of its subsets is so. Thus Theorem 5 yields termination. ■

Consider again fuzzy constraint problems. Here $\times$ is *min*; thus it is idempotent. Therefore, by Theorem 6, GI always terminates when it is instantiated with fuzzy CSPs. This is an alternative and easier way than Theorem 5 to guarantee that soft constraint propagation for fuzzy CSPs always terminates. In fact, we do not have to find a semiring closure of the problem, but just check that the multiplicative operation is idempotent.

Given all the above theorems concerning termination, we can devise different strategies towards proving the termination of the GI algorithm, when it is given a soft constraint problem P and some constraint propagation functions.

In particular, the termination condition of the previous theorems can assure the efficient application of some specific local propagation schemas. Moreover, the possibility to check the termination of the schema by looking at the semiring or at the constraint system can lead to several scenarios:

- If the constraint propagation functions are defined via the two operations of semiring S, and the multiplicative operation of S is idempotent, then GI terminates (by Theorem 6).
- If, instead, $\times$ is not idempotent, but the constraint propagation functions are defined via the two semiring operations, we can check if the restriction of the semiring order to the semiring closure of P satisfies the DCC. In that case, GI terminates (by Theorem 5).
- If the restriction of the semiring order to the semiring closure of P does not satisfy the DCC or the constraint propagation functions are not defined via $+$ and $\times$, then we can try to prove that the problem order on $P\uparrow$ satisfies the ACC. In that case, GI terminates (by Theorem 4).

7. Conclusions and Future Work

The results of this paper show that it is possible to treat in a uniform way hard and soft constraint propagation. This was already shown in [8], but restricted to constraint propagation algorithms in which each step computes the solution of a subproblem. In this paper, instead, we have extended the notion of constraint propagation algorithms via

the GI schema of [3], so that also constraint propagation steps that approximate solutions (as is the case for bounds-consistency) can be modelled.

Moreover, we have also studied the conditions under which this more general notion of soft constraint propagation terminates. Some results were already present in [8] and [3], but here we have generalized and extended them, providing more general, even easily checkable, sufficient conditions.

In particular, we have defined and proved three sufficient conditions which guarantee termination, and which are based on different aspects of the constraint solving framework: the structure of the partial order of constraint problems, the structure of a sub-ordering of the semiring elements, and the properties of the semiring operations. Depending on what is easier to check, one may use one or another of these conditions to prove termination of the available constraint propagation framework.

We are confident that the results of this paper could be useful to extend many of the existent local propagation algorithms for crisp constraint problems to the soft constraint framework. By using the proposed schema and the results of this paper, we can both model them, and also in many cases prove their termination, thus making them practical to use. We therefore plan to consider many of such algorithms and study their generalization to soft constraints.

According to the results in [6], there is a strict relationship between constraint propagation and abstraction of soft constraints. In fact, it has been shown that performing soft constraint propagation can be seen as a special form of constraint abstraction, where one passes from one semiring to a simpler one. We therefore plan to investigate the use of soft constraint propagation algorithms, and of the results of this paper, as an aid in making soft constraint abstraction more practical.

Acknowledgments

The second author is indebted to K. R. Apt for his advice to investigate the topic of this paper, and to M. de Rijke for his suggestions. This research has been partially supported by Italian MURST project TOSCA.

Notes

1. Thanks to M. de Rijke for pointing it out to us.
2. Cf. [12] for a generalization of GI to a more expressive schema for classical constraint propagation algorithms like AC-4 and AC-5.
3. Notice that here the problem is that it could return a problem which is not equivalent, due to the non-idempotence of the multiplicative operation.

References

1. Apt, K. (1999a). The essence of constraint propagation. *Theoretical Computer Science*, 221(1–2): 179–210.
2. Apt, K. (1999b). The rough guide to constraint propagation (corrected version). In *Proc. of the 5th International Conference on Principles and Practice of Constraint Programming (CP'99)*, pp. 1–23.

3. Apt, K. (2000). The role of commutativity in constraint propagation algorithms. *ACM TOPLAS*, 22(6): 1002–1036.
4. Bistarelli, S. (2001). Soft constraint solving and programming: A general framework. Ph.D. thesis, Dipartimento di Informatica, Università di Pisa, Italy.
5. Bistarelli, S., Codognet, P., Georget, Y., & Rossi, F. (2000a). Labeling and partial local consistency for soft constraint programming. In *Proc. of the 2nd International Workshop on Practical Aspects of Declarative Languages (PADL'00)*.
6. Bistarelli, S., Codognet, P., & Rossi, F. (2000b). An abstraction framework for soft constraint and its relationship with constraint propagation. In *Proc. SARA2000 Symposium on Abstraction, Reformulation and Approximation*.
7. Bistarelli, S., Gennari, R., & Rossi, F. (2000c). Constraint propagation for soft constraint satisfaction problems: Generalization and termination conditions. In *Proc. Sixth Int. Conf. on Principles and Practice of Constraint Programming (CP2000)*.
8. Bistarelli, S., Montanari, U., & Rossi, F. (1997). Semiring-based constraint solving and optimization. *Journal of the ACM*, 44(2): 201–236.
9. Dubois, D., Fargier, H., & Prade, H. (1993). The calculus of fuzzy restrictions as a basis for flexible constraint satisfaction. In *Proc. IEEE International Conference on Fuzzy Systems*, pp. 1131–1136.
10. Fargier, H., & Lang, J. (1993). Uncertainty in constraint satisfaction problems: A probabilistic approach. In *Proc. European Conference on Symbolic and Qualitative Approaches to Reasoning and Uncertainty (ECSQARU)*, pp. 97–104.
11. Freuder, E., & Wallace, R. (1992). Partial constraint satisfaction. *AI Journal*, 58.
12. Gennari, R. (2000). Arc consistency algorithms via iterations of subsumed functions. In *Proc. of the 1st International Conference on Computational Logic (CL'00)*, pp. 358–372.
13. Marriott, K, & Stuckey, P. (1998). *Programming with Constraints*. MIT Press.
14. Schiex, T., Fargier, H., & Verfaille, G. (1995). Valued constraint satisfaction problems: Hard and easy problems. In *Proc. IJCAI95*. San Francisco, CA, pp. 631–637.