Constraint Solving over Semirings

Stefano Bistarelli **Ugo Montanari** **Francesca Rossi**

University of Pisa, Dipartimento di Informatica
Corso Italia 40, 56125 Pisa, Italy
E-mail: {bista,ugo,rossi}@di.unipi.it

Abstract

We introduce a general framework for constraint solving where classical CSPs, fuzzy CSPs, weighted CSPs, partial constraint satisfaction, and others can be easily cast. The framework is based on a semiring structure, where the set of the semiring specifies the values to be associated to each tuple of values of the variable domain, and the two semiring operations ($+$ and $\times$) model constraint projection and combination respectively. Local consistency algorithms, as usually used for classical CSPs, can be exploited in this general framework as well, provided that some conditions on the semiring operations are satisfied. We then show how this framework can be used to model both old and new constraint solving schemes, thus allowing one both to formally justify many informally taken choices in existing schemes, and to prove that the local consistency techniques can be used also in newly defined schemes.

1 Introduction

Classical constraint satisfaction problems (CSPs) [Montanari, 1974; Mackworth, 1988] are a very expressive and natural formalism to specify many kinds of real-life problems. In fact, problems ranging from map coloring, vision, robotics, job-shop scheduling, VLSI design, etc., can easily be cast as CSPs and solved using one of the many techniques that have been developed for such problems or subclasses of them [Freuder, 1978; 1988; Mackworth and Freuder, 1985; Mackworth, 1977; Montanari, 1974].

However, they also have evident limitations, mainly due to the fact that they are not very flexible when trying to represent real-life scenarios where the knowledge is not completely available nor crisp. In fact, in such situations, the ability of stating whether an instantiation of values to variables is allowed or not is not enough or sometimes not even possible. For these reasons, it is natural to try to extend the CSP formalism in this direction.

For example, in [Rosenfeld *et al.*, 1976; Dubois *et al.*, 1993; Ruttkay, 1994] CSPs have been extended with the ability to associate to each tuple, or to each constraint, a level of preference, and with the possibility of combining constraints using min-max operations. This extended formalism has been called Fuzzy CSPs (FCSPs). Other extensions concern the ability to model incomplete knowledge of the real problem [Fargier and Lang, 1993], to solve overconstrained problems [Freuder and Wallace, 1992], and to represent cost optimization problems.

In this paper we define a constraint solving framework where all such extensions, as well as classical CSPs, can be cast. However, we do not relax the assumption of a finite domain for the variables of the constraint problems. The main idea is based on the observation that a semiring (that is, a domain plus two operations satisfying certain properties) is all what is needed to describe many constraint satisfaction schemes. In fact, the domain of the semiring provides the levels of consistency (which can be interpreted as cost, or degrees of preference, or probabilities, or others), and the two operations define a way to combine constraints together. Specific choices of the semiring will then give rise to different instances of our framework.

In classical CSPs, the so-called local consistency techniques [Freuder, 1978; 1988; Mackworth, 1988; 1977; Montanari, 1974; Montanari and Rossi, 1991] have been proved to be very effective when approximating the solution of a problem. In this paper we study how to generalize this notion to our framework, and we provide some sufficient conditions over the semiring operations which guarantee that such algorithms can be fruitfully applied also to our scheme. Here for being "fruitfully applicable" we mean that 1) the algorithm terminates and 2) the resulting problem is equivalent to the given one and it does not depend on the nondeterministic choices made during the algorithm.

The advantage of our framework, that we call SCSP (for Semiring-based CSP), is that one can just see his own constraint solving framework as an instance of SCSP over a certain semiring, and can immediately inherit the results obtained for the general framework. In particular, in this case, one can immediately see whether a local consistency technique can be applied or not. This allows one also to justify many informally taken choices in existing constraint solving schemes. In this paper we study several known and new constraint solving frameworks, casting them as instances of SCSP, and for some of them we prove the possibility of applying k-consistency algorithms.

Due to space limitations, some of the formal definitions, theorems, and proofs have been omitted or just informally sketched.

2 C-semirings and their properties

We associate a semiring to the standard definition of constraint problem, so that different choices of the semiring represent different concrete constraint satisfaction schemes. Such semiring will give us both the domain for the non-crisp statements and also the allowed operations on them. More precisely, in the following we will consider *c-semirings*, that is, semirings with additional properties of the two operations.

Definition 1 *A* **semiring** *is a tuple* $(A, +, \times, \mathbf{0}, \mathbf{1})$ *such that*

- *A is a set and $\mathbf{0}, \mathbf{1} \in A$;*
- *$+$, called the additive operation, is a closed (i.e., $a, b \in A$ implies $a + b \in A$), commutative (i.e., $a + b = b + a$) and associative (i.e., $a + (b + c) = (a + b) + c$) operation such that $a + \mathbf{0} = a = \mathbf{0} + a$ (i.e., $\mathbf{0}$ is its unit element);*
- *$\times$, called the multiplicative operation, is a closed and associative operation such that $\mathbf{1}$ is its unit element and $a \times \mathbf{0} = \mathbf{0} = \mathbf{0} \times a$ (i.e., $\mathbf{0}$ is its absorbing element);*
- *$\times$ distributes over $+$ (i.e., $a \times (b + c) = (a \times b) + (a \times c)$).*

A **c-semiring** *is a semiring such that $+$ is idempotent, $\times$ is commutative, and $\mathbf{1}$ is the absorbing element of $+$.* □

The idempotence of the $+$ operation is needed in order to define a partial ordering $\leq_S$ over the set A, which will enable us to compare different elements of the semiring. Such partial order is defined as follows: $a \leq_S b$ iff $a + b = a$. Intuitively, $a \leq_S b$ means that a is "better" than b. This will be used later to choose the "best" solution in our constraint problems. It is important to notice that both $+$ and $\times$ are monotone on such ordering.

The commutativity of the $\times$ operation is desirable when such operation is used to combine several constraints. In fact, were it not commutative, it would mean that different orders of the constraints give different results.

If $\mathbf{1}$ is also the absorbing element of the additive operation, then we have that $\mathbf{1} \leq_S a$ for all a. Thus $\mathbf{1}$ is the minimum (i.e., the best) element of the partial ordering. This implies that the $\times$ operation is *extensive*, that is, that $a \leq a \times b$. This is important since it means that combining more constraints leads to a worse (w.r.t. the $\leq_S$ ordering) result. The fact that $\mathbf{0}$ is the unit element of the additive operation implies that $\mathbf{0}$ is the maximum element of the ordering. Thus, for any $a \in A$, we have $\mathbf{1} \leq_S a \leq_S \mathbf{0}$.

In the following we will sometimes need the $\times$ operation to be closed on a certain finite subset of the c-semiring. More precisely, given any c-semiring $S = (A, +, \times, \mathbf{0}, \mathbf{1})$, consider a finite set $I \subseteq A$. Then, $\times$ is *I-closed* if, for any $a, b \in I$, $(a \times b) \in I$.

3 Constraint systems and problems

A constraint system provides the c-semiring to be used, the set of all variables, and their domain D. Then, a constraint over a given constraint system specifies the involved variables and the "allowed" values for them. More precisely, for each tuple of values of D for the involved variables, a corresponding element of the semiring is given. This element can be interpreted as the tuple weight, or cost, or level of confidence, or else. Finally, a constraint problem is then just a set of constraints over a given constraint system, plus a selected set of variables. These are the variables of interest in the problem, i.e., the variables of which we want to know the possible assignments compatibly with all the constraints.

Definition 2 *A* **constraint system** *is a tuple $CS = \langle S, D, V\rangle$, where S is a c-semiring, D is a finite set, and V is an ordered set of variables. Given a constraint system $CS = \langle S, D, V\rangle$, where $S = (A, +, \times, \mathbf{0}, \mathbf{1})$, a* **constraint** *over CS is a pair $\langle def, con\rangle$, where $con \subseteq V$ and it is called the* type *of the constraint, and $def : D^k \rightarrow A$ (where k is the size of con, that is, the number of variables in it), and it is called the* value *of the constraint. Moreover, a* **constraint problem** *P over CS is a pair $P = \langle C, con\rangle$, where C is a set*[1] *of constraints over CS and $con \subseteq V$.* □

In the following we will consider a fixed constraint system $CS = \langle S, D, V\rangle$, where $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$. Note that when all variables are of interest, like in many approaches to classical CSP, con contains all the variables involved in any of the constraints of the problem. This set will be denoted by $V(P)$ and can be recovered by looking at the variables involved in each constraint: $V(P) = \bigcup_{\langle def, con'\rangle \in C} con'$.

In the $SCSP$ framework, the values specified for the tuples of each constraint are used to compute corresponding values for the tuples of values of the variables in con, according to the semiring operations: the multiplicative operation is used to combine the values of the tuples of each constraint to get the value of a tuple for all the variables, and the additive operation is used to obtain the value of the tuples of the variables of interest. More precisely, we can define the operations of *combination* ($\otimes$) and *projection* ($\Downarrow$) over constraints. Analogous operations have been originally defined for fuzzy relations in [Zadeh, 1975], and have then been used for fuzzy CSPs in [Dubois *et al.*, 1993]. Our definition is however more general since we do not consider a specific c-semiring but a general one.

Definition 3 *Given two constraints $c_1 = \langle def_1, con_1\rangle$ and $c_2 = \langle def_2, con_2\rangle$ over CS, their* **combination**, *$c_1 \otimes c_2$, is the constraint $c = \langle def, con\rangle$ with $con = con_1 \cup con_2$ and $def(t) = def_1(t \downarrow^{con}_{con_1}) \times def_2(t \downarrow^{con}_{con_2})$, where, for any tuple of values t for the variables in a set I, $t \downarrow^{I}_{I'}$ denotes the projection of t over the variables in the set I'. Moreover, given a constraint $c = \langle def, con\rangle$ over CS, and a subset w of con, its* **projection** *over w, written*

[1] Note that, if $\times$ is not idempotent, it is necessary to see C as a multiset, and not as a set.

$c \Downarrow_w$, *is the constraint* $\langle def', con' \rangle$ *over* CS *with* $con' = w$ *and* $def'(t') = \Sigma_{\{t \mid t\downarrow^{con}_{w}=t'\}} def(t)$. □

Using such operations, we can now define the notion of solution of a SCSP.

Definition 4 *Given a constraint problem* $P = \langle C, con \rangle$ *over a constraint system* CS, *the* **solution** *of* P *is a constraint defined as* $Sol(P) = (\bigotimes C) \Downarrow_{con}$, *where* $\bigotimes C$ *is the obvious extension of the combination operation to a set of constraints* C. □

In words, the solution of a SCSP is the constraint induced on the variables in *con* by the whole problem. Such constraint provides, for each tuple of values of D for the variables in *con*, an associated value of A. Sometimes, it is enough to know just the best value associated to such tuples. In our framework, this is still a constraint (over an empty set of variables), and will be called the best level of consistency of the whole problem, where the meaning of "best" depends on the ordering $\leq_S$ defined by the additive operation.

Definition 5 *Given a SCSP problem* $P = \langle C, con \rangle$, *we define the* **best level of consistency** *of* P *as* $blevel(P) = (\bigotimes C) \Downarrow_{\emptyset}$. *If* $blevel(P) = \langle blev, \emptyset \rangle$, *then we say that* P *is consistent if* $blev <_S \mathbf{0}$. *Instead, we say that* P *is* α*-consistent if* $blev = \alpha$. □

Informally, the best level of consistency gives us an idea of how much we can satisfy the constraints of the given problem. Note that $blevel(P)$ does not depend on the choice of the distinguished variables, due to the associative property of the additive operation. Thus, since a constraint problem is just a set of constraints plus a set of distinguished variables, we can also apply function *blevel* to a set of constraints only. Also, since the type of constraint $blevel(P)$ is always an empty set of variables, in the following we will just write the value of *blevel*.

Another interesting notion of solution, more abstract than the one defined above, but sufficient for many purposes, is the one that provides only the tuples that have an associated value which coincides with (the def of) $blevel(P)$. However, this notion makes sense only when $\leq_S$ is a total order. In fact, were it not so, we could have an incomparable set of tuples, whose sum (via $+$) does not coincide with any of the summed tuples. Thus it could be that none of the tuples has an associated value equal to $blevel(P)$.

By using the ordering $\leq_S$ over the semiring, we can also define a corresponding partial ordering on constraints with the same type, as well as a preorder and a notion of equivalence on problems.

Definition 6 *Consider two constraints* c_1, c_2 *over* CS, *and assume that* $con_1 = con_2$. *Then we define the* **constraint ordering** $\sqsubseteq_S$ *as the following partial ordering:* $c_1 \sqsubseteq_S c_2$ *if and only if, for all tuples* t *of values from* D, $def_1(t) \leq_S def_2(t)$. *Notice that, if* $c_1 \sqsubseteq_S c_2$ *and* $c_2 \sqsubseteq_S c_1$, *then* $c_1 = c_2$. *Consider now two SCSP problems* P_1 *and* P_2 *such that* $P_1 = \langle C_1, con \rangle$ *and* $P_2 = \langle C_2, con \rangle$. *Then we define the* **problem preorder** $\sqsubseteq_P$ *as:* $P_1 \sqsubseteq_P P_2$ *if* $Sol(P_1) \sqsubseteq_S Sol(P_2)$. *If* $P_1 \sqsubseteq_P P_2$ *and* $P_2 \sqsubseteq_P P_1$, *then they have the same solution. Thus we say that* P_1 *and* P_2 *are* **equivalent** *and we write* $P_1 \equiv P_2$. □

The notion of problem preorder can also be useful to show that, as in the classical CSP case, also the SCSP framework is monotone: $(C, con) \sqsubseteq_P (C \cup C', con)$. That is, if some constraints are added, the solution (as well as the *blevel*) of the new problem is worse or equal than that of the old one.

4 Local Consistency

Computing any one of the previously defined notions (like the best level of consistency and the solution) is an NP-hard problem. Thus it can be convenient in many cases to approximate such notions. In classical CSP, this is done using the so-called local consistency techniques. Such techniques can be extended also to constraint solving over any semiring, provided that some properties are satisfied. Here we define what k-consistency [Freuder, 1978; 1988; Kumar, 1992] means for SCSP problems. Informally, an SCSP problem is k-consistent when, taken any set W of k-1 variables and any k-th variable, the constraint obtained by combining all constraints among the k variables and projecting it onto W is better or equal (in the ordering $\sqsubseteq_S$) than that obtained by combining the constraints among the variables in W only.

Definition 7 *Given a SCSP problem* $P = \langle C, con \rangle$ *we say that* P *is* **k-consistent** *if, for all* $W \subseteq V(P)$ *such that size(W)=* $k-1$, *and for all* $x \in (V(P) - W)$, $((\otimes\{c_i \mid c_i \in C \wedge con_i \subseteq (W \cup \{x\})\}) \Downarrow_W) \sqsubseteq_S (\otimes\{c_i \mid c_i \in C \wedge con_i \subseteq W)\}$, *where* $c_i = \langle def_i, con_i \rangle$ *for all* $c_i \in C$. □

Note that, since $\times$ is extensive, in the above formula for k-consistency we could also replace $\sqsubseteq_S$ by $\equiv_S$. In fact, the extensivity of $\times$ assures that the formula always holds when $\sqsupseteq_S$ is used instead of $\subseteq_S$.

Making a problem k-consistent means explicitating some implicit constraints, thus possibly discovering inconsistency at a local level. In classical CSP, this is crucial, since local inconsistency implies global inconsistency. This is true also in SCSPs. But here we can be even more precise, and relate the best level of consistency of the whole problem to that of its subproblems.

Theorem 8 *Consider a set of constraints* C *over* CS, *and any subset* C' *of* C. *If* C' *is* α*-consistent, then* C *is* β*-consistent, with* $\alpha \leq_S \beta$.

Proof: If C' is α-consistent, it means that $\otimes C' \Downarrow_{\emptyset} = \langle \alpha, \emptyset \rangle$. Now, C can be seen as $C' \otimes C''$ for some C''. By extensivity of $\times$, and the monotonicity of $+$, we have that $\beta = \otimes(C' \otimes C'') \Downarrow \emptyset \sqsupseteq_S \otimes(C) \Downarrow \emptyset = \alpha$. □

If a subset of constraints of P is inconsistent (that is, its blevel is 0), then the above theorem implies that the whole problem is inconsistent as well.

We now define a generic k-consistency algorithm, by extending the usual one for classical CSPs [Freuder, 1978; Kumar, 1992]. We assume to start from a *SCSP* problem where all constraints of arity k-1 are present. If some are not present, we just add them with a non-restricting definition. That is, for any added constraint

$c = \langle def, con\rangle$, we set $def(t) = \mathbf{1}$ for all con-tuples t. This does not change the solution of the problem, since $\mathbf{1}$ is the unit element for the $\times$ operation.

The idea of the (naive) algorithm is to combine any constraint c of arity k-1 with the projection over such k-1 variables of the combination of all the constraints connecting the same k-1 variables plus another one, and to repeat such operation until no more changes can be made to any (k-1)-arity constraint.

In doing that, we will use the additional notion of *typed locations*. Informally, a typed location is just a location (as in ordinary imperative programming) which can be assigned to a constraint of the same type. This is needed since the constraints defined in Definition 2 are just pairs $\langle def, con\rangle$, where def is a *fixed* function and thus not modifiable. In this way, we can also assign the value of a constraint to a typed location (only if the type of the location and that of the constraint coincide), and thus achieve the effect of modifying the value of a constraint.

Definition 9 *A* **typed location** *is an object $l : con$ whose type is con. The assigment operation $l := c$, where c is a constraint $\langle def, con\rangle$, has the meaning of associating, in the present store, the value def to l. Whenever a typed location appears in a formula, it will denote its value.* □

Definition 10 *Consider an SCSP problem $P = \langle C, con\rangle$ and take any subset $W \subseteq V(P)$ such that size(W) $= k - 1$ and any variable $x \in (V(P) - W)$. Let us now consider a typed location l_i for each constraint $c_i = \langle def_i, con_i\rangle \in C$ such that $l_i : con_i$. Then a* **k-consistency algorithm** *works as follows.*

1. *Inizialize all locations by performing $l_i := c_i$ for each $c_i \in C$.*
2. *Consider*
   - $l_j : W$,
   - $A(W) = \otimes\{l_i \mid con_i \subseteq (W \cup \{x\})\} \Downarrow W$, *and*
   - $B(W) = \otimes\{l_i \mid con_i \subseteq W\}$.

   Then, if $A(W) \not\sqsubseteq_S B(W)$, perform $l_j := l_j \otimes A(W)$.
3. *Repeat step 2 on all W and x until $A(W) \sqsubseteq B(W)$ for all W.*

Upon stability, assume that each typed location $l_i : con_i$ has $eval(l_i) = def'_i$. Then the result of the algorithm is a new SCSP problem $P' =$ k-cons(P) $= \langle C', con\rangle$ such that $C' = \bigcup_i \langle def'_i, con_i\rangle$. □

Assuming the termination of such algorithm (we will discuss such issue later), it is obvious to show that the problem obtained at the end is k-consistent. This is a very naive algorithm, whose efficiency can be improved easily by using the methods which have been adopted for classical k-consistency.

In classical CSP, any k-consistency algorithm enjoys some important properties. We now will study these same properties in our SCSP framework, and point out the corresponding properties of the semiring operations which are necessary for them to hold. The desired properties are as follows: that any k-consistency algorithm returns a problem which is equivalent to the given one; that it terminates in a finite number of steps; and that the order in which the (k-1)-arity subproblems are selected does not influence the resulting problem.

Theorem 11 *Consider a SCSP problem P and a SCSP problem $P' =$ k-cons(P). Then, $P \equiv P'$ (that is, P and P' are equivalent) if $\times$ is idempotent.*

Proof: Assume $P = \langle C, con\rangle$ and $P' = \langle C', con\rangle$. Now, C' is obtained by C by changing the definition of some of the constraints (via the typed location mechanism). For each of such constraints, the change consists of combining the old constraint with the combination of other constraints. Since the multiplicative operation is commutative and associative (and thus also $\otimes$), $\otimes C'$ can also be written as $(\otimes C) \otimes C''$, where $\otimes C'' \sqsubseteq_S \otimes C$. If $\times$ is idempotent, then $((\otimes C) \otimes C'') = (\otimes C)$. Thus $(\otimes C) = (\otimes C')$. Therefore $P \equiv P'$. □

Theorem 12 *Consider any SCSP problem P where $CS = \langle S, D, V\rangle$ and the set $AD = \bigcup_{\langle def, con\rangle \in C} R(def)$, where $R(def) = \{a \mid \exists t \text{ with } def(t) = a\}$. Then the application of the k-consistency algorithm to P terminates in a finite number of steps if AD is contained in a set I which is finite and such that $+$ and $\times$ are I-closed.*

Proof: Each step of the k-consistency algorithm may change the definition of one constraint by assigning a different value to some of its tuples. Such value is strictly worse (in terms of $\leq_S$) since $\times$ is extensive. Moreover, it can be a value which is not in AD but in $I - AD$. If the state of the computation consists of the definitions of all constraints, then at each step we get a strictly worse state (in terms of $\sqsubseteq_S$). The sequence of such computation states, until stability, has finite length, since by assumption I is finite and thus the value associated to each tuple of each constraint may be changed at most size(I) times. □

An interesting special case of the above theorem occurs when the chosen semiring has a finite domain A. In fact, in that case the hypotheses of the theorem hold with $I = A$. Another useful result occurs when $+$ and $\times$ are AD-closed. In fact, in this case one can also compute the time complexity of the k-consistency algorithm by just looking at the given problem. More precisely, if this same algorithm is $O(n^k)$ in the classical CSP case [Freuder, 1978; 1988; Mackworth and Freuder, 1985; Mackworth, 1977], then here it is $O(\text{size}(AD) \times n^k)$ (in [Dubois *et al.*, 1993] they reach the same conclusion for the fuzzy CSP case).

No matter in which order the subsets W of k-1 variables, as well as the additional variables x, are chosen during the k-consistency algorithm, the result is always the same problem. However, this holds in general only if $\times$ is idempotent.

Theorem 13 *Consider a SCSP problem P and two different applications of the k-consistency algorithm to P, producing respectively P' and P''. Then $P' = P''$ if $\times$ is idempotent.* □

5 Instances of the framework

We will now show how several known, and also new, frameworks for constraint solving may be seen as instances of the SCSP framework. More precisely, each of such frameworks corresponds to the choice of a specific constraint system (and thus of a semiring). This means that we can immediately know whether one can inherit the properties of the general framework by just looking at the properties of the operations of the chosen semiring, and by referring to the theorems in the previous section. This is interesting for known constraint solving schemes, because it puts them into a single unifying framework and it justifies in a formal way many informally taken choices, but it is especially significant for new schemes, for which one does not need to prove all the properties that it enjoys (or not) from scratch. Since we consider only finite domain constraint solving, in the following we will only specify the semiring that has to be chosen to obtain a particular instance of the SCSP framework.

5.1 Classical CSPs

A classical CSP problem [Montanari, 1974; Mackworth, 1988] is just a set of variables and constraints, where each constraint specifies the tuples that are allowed for the involved variables. Assuming the presence of a subset of distinguished variables, the solution of a CSP consists of a set of tuples which represent the assignments of the distinguished variables which can be extended to total assignments which satisfy all the constraints.

Since constraints in CSPs are crisp, we can model them via a semiring with only two values, say 1 and 0: allowed tuples will have the value 1, and not allowed ones the value 0. Moreover, in CSPs, constraint combination is achieved via a join operation among allowed tuple sets. This can be modelled here by taking as the multiplicative operation the logical *and* (and interpreting 1 as true and 0 as false). Finally, to model the projection over the distinguished variables, as the k-tuples for which there exists a consistent extension to an n-tuple, it is enough to assume the additive operation to be the logical *or*. Therefore a CSP is just an SCSP where the c-semiring in the constraint system CS is $S_{CSP} = \langle\{0,1\}, \vee, \wedge, 0, 1\rangle$. The ordering $\leq_S$ here reduces to $1 \leq_S 0$. As predictable, all the properties related to k-consistency hold. In fact, $\wedge$ is idempotent. Thus the results of Theorems 11 and 13 apply. Also, since the domain of the semiring is finite, the result of Theorem 12 applies as well.

5.2 Fuzzy CSPs

Fuzzy CSPs (FCSPs) [Rosenfeld *et al.*, 1976; Dubois *et al.*, 1993; Ruttkay, 1994; Schiex, 1992] extend the notion of classical CSPs by allowing non-crisp constraints, that is, constraints which associate a preference level to each tuple of values. Such level is always between 0 and 1, where 1 represents the best value (that is, the tuple is allowed) and 0 the worst one (that is, the tuple is not allowed). The solution of a fuzzy CSP is then defined as the set of tuples of values which have the maximal value. The value associated to n-tuple is obtained by minimizing the values of all its subtuples. Fuzzy CSPs are already a very significant extension of CSPs. In fact, they are able to model partial constraint satisfaction [Freuder and Wallace, 1992], so to get a solution even when the problem is overconstrained, and also prioritized constraints, that is, constraints with different levels of importance [Borning *et al.*, 1989].

Fuzzy CSPs can be modelled in our framework by choosing the c-semiring $S_{FCSP} = \langle\{x \mid x \in [0,1]\}, max, min, 0, 1\rangle$. The ordering $\leq_S$ here reduces to the $\geq$ ordering on reals. The multiplicative operation of S_{FCSP} (that is, *min*) is idempotent. Thus Theorem 11 and 13 can be applied. Moreover, *min* is AD-closed for any finite subset of [0,1]. Thus, by Theorem 12, any k-consistency algorithm terminates. Thus FCSPs, although providing a significant extension to classical CSPs, can exploit the same kind of local consistency algorithms. An implementation of arc-consistency, suitably adapted to be used over fuzzy CSPs, is given in [Schiex, 1992] (although no formal properties of its behaviour are proved).

5.3 Probabilistic CSPs

Probabilistic CSPs (PCSPs) [Fargier and Lang, 1993] have been introduced to model those situations where each constraint c has a certain independent probability $p(c)$ to be part of the given real problem. This allows one to reason also about problems which are only partially known. The probability of each constraint gives then, to each instantiation of all the variables, a probability that it is a solution of the real problem. This is done by associating to an n-tuple t the probability that all constraints that t violates are in the real problem. This is just the product of all $1 - p(c)$ for all c violated by t. Finally, the aim is to get those instantiations with the maximum probability.

The relationship between PCSPs and SCSPs is complicated by the fact that PCSPs contain crisp constraints with probability levels, while SCSPs contain non-crisp constraints. That is, we associate values to tuples, and not to constraints. However, it is still possible to model PCSPs, by using a transformation which is similar to that proposed in [Dubois *et al.*, 1993] to model prioritized constraints via soft constraints in the FCSP framework. More precisely, we assign probabilities to tuples instead of constraints: consider any constraint c with probability $p(c)$, and let t be any tuple of values for the variables involved in c; then we set $p(t) = 1$ if t is allowed by c, otherwise $p(t) = 1 - p(c)$. The reasons for such a choice are as follows: if a tuple is allowed by c and c is in the real problem, then t is allowed in the real problem; this happens with probability $p(c)$; if instead c is not in the real problem, then t is still alowed in the real problem, and this happens with probability $1-p(c)$. Thus t is allowed in the real problem with probability $p(c) + 1 - p(c) = 1$. Consider instead a tuple t which is not allowed by c. Then it will be allowed in the real problem only if c is not present; this happens with probability $1 - p(c)$.

To give the appropriate value to an n-tuple t, given the values of all the smaller k-tuples, with $k \leq n$ and which are subtuples of t (one for each constraint), we just perform the product of the value of such subtuples. By the way values have been assigned to tuples in constraints, this coincides with the product of all $1 - p(c)$ for all c violated by t. In fact, if a subtuple violates c, then by construction its value is $1 - p(c)$; if instead a subtuple satisfies c, then its value is 1. Since 1 is the unit element of $\times$, we have that $1 \times a = a$ for each a. Thus we get $\Pi(1 - p(c))$ for all c that t violates.

As a result, the c-semiring corresponding to the PCSP framework is $S_{prob} = \langle \{x \mid x \in [0,1]\}, max, \times, 0, 1 \rangle$, and the associated ordering $\leq_S$ here reduces to $\geq$ over reals. Note that the fact that P' is α-consistent means that in P there exists an n-tuple which has probability α to be a solution of the real problem.

The multiplicative operation of S_{prob} (that is, $\times$) is not idempotent. Thus neither Theorem 11 nor Theorem 13 can be applied. Also, $\times$ is not closed on any superset of any non-trivial finite subset of [0,1]. Thus Theorem 12 cannot be applied as well. Therefore, k-consistency algorithms do not make much sense in the PCSP framework, since none of the usual desired properties hold. However, the fact that we are dealing with a c-semiring implies that, at least, we can apply Theorem 8: if a PCSP problem has a tuple with probability α to be a solution of the real problem, then any subproblem has a tuple with probability at least α to be a solution of a subproblem of the real problem. This can be fruitfully used when searching for the best solution in a branch-and-bound search algorithm.

5.4 Weighted CSPs

While fuzzy CSPs associate a level of preference to each tuple in each constraint, in weighted CSPs (WCSPs) tuples come with an associated cost. This allows one to model optimization problems where the goal is to minimize the total cost (time, space, number of resources, ...) of the proposed solution. Therefore, in WCSPs the cost function is defined by summing up the costs of all constraints (intended as the cost of the chosen tuple for each constraint). Thus the goal is to find the n-tuples (where n is the number of all the variables) which minimize the total sum of the costs of their subtuples (one for each constraint).

According to this informal description of WCSPs, the associated c-semiring is $S_{WCSP} = \langle \mathcal{R}^+, \min, +, +\infty, 0 \rangle$, with ordering $\leq_S$ which reduces here to $\leq$ over the reals. This means that a value is preferred to another one if it is smaller.

The multiplicative operation of S_{WCSP} (that is, $+$) is not idempotent. Thus the k-consistency algorithms cannot be used (in general) in the WCSP framework, since none of the usual desired properties hold. However, again, the fact that we are dealing with a c-semiring implies that, at least, we can apply Theorem 8: if a WCSP problem has a best solution with cost α, then the best solution of any subproblem has a cost smaller than α. This can be convenient to know in a branch-and-bound search algorithm.

5.5 Egalitarianism and Utilitarianism: FCSP + WCSP

The FCSP and the WCSP systems can be seen as two different approaches to give a meaning to the notion of optimization. The two models correspond in fact, respectively, to two definitions of social welfare in utility theory [Moulin, 1988]: *egalitarianism*, which maximizes the minimal individual utility, and *utilitarianism*, which maximizes the sum of the individual utilities.

In this section we show how our framework allows also for the combination of these two approaches. In fact, we construct an instance of the SCSP framework where the two approaches coexist, and allow us to discriminate among solutions which otherwise would result indistinguishable. More precisely, we first compute the solutions according to egalitarianism (that is, using a $max - min$ computation as in FCSPs), and then discriminate more among them via utilitarianism (that is, using a $max - sum$ computation as in WCSPs). The resulting c-semiring is $S_{ue} = \langle \{\langle l, k \rangle \mid l, k \in [0,1]\}, \underline{max}, \underline{min}, \langle 0, 0 \rangle, \langle 1, 0 \rangle \rangle$, where $\underline{max}$ and $\underline{min}$ are defined as follows:

$$\langle l_1, k_1 \rangle \underline{max} \langle l_2, k_2 \rangle = \begin{cases} \langle l_1, max(k_1, k_2) \rangle & \text{if } l_1 = l_2 \\ \langle l_1, k_1 \rangle & \text{if } l_1 > l_2 \end{cases}$$

$$\langle l_1, k_1 \rangle \underline{min} \langle l_2, k_2 \rangle = \begin{cases} \langle l_1, k_1 + k_2 \rangle & \text{if } l_1 = l_2 \\ \langle l_2, k_2 \rangle & \text{if } l_1 > l_2 \end{cases}$$

That is, the domain of the semiring contains pairs of values: the first element is used to reason via the *max-min* approach, while the second one is used to further discriminate via the *max-sum* approach. More precisely, given two pairs, if the first elements of the pairs differ, then the $\underline{max} - \underline{min}$ operations behave like a normal $max - min$, otherwise they behave like *max-sum*. This can be interpreted as the fact that, if the first element coincide, it means that the *max-min* criteria cannot discriminate enough, and thus the $max - sum$ criteria is used.

Since $\underline{min}$ is not idempotent, k-consistency algorithms cannot in general be used meaningfully in this instance of the framework.

A kind of constraint solving similar to that considered in this section is the one presented in [Fargier *et al.*, 1993], where Fuzzy CSPs are augmented with a finer way of selecting the preferred solution. More precisely, they employ a lexicographic ordering to improve the discriminating power of FCSPs and avoid the so-called *drowing effect*. This approach can be rephrased in our framework as well, yielding an instance where k-consistency algorithms can be applied.

6 Conclusions

In some of the instances considered, we have shown that the sufficent conditions for the application of k-consistency algorithms do not hold. However, this does not mean that problems defined according to such schemes are not easily solvable in some special cases. In fact, one can show that our general framework is always compatible with the notion of *dynamic programming*, which, in the case of classical CSPs with a thick-tree structure, reduces to that of perfect relaxation [Montanari and Rossi, 1991].

We plan to extend our framework in order to take into account several dimensions of "preference". In fact, in practice one could want, for example, to minimize cost and time, while maximizing the level of confidence and some notion of probability. Each of these desires could be cast in one c-semiring as shown in this paper, and then the combination of the four c-semirings (which is again a c-semiring) can be used to model the entire solving scheme. However, more attention has to be put now in the choice of the "best" solutions, since the presence of several dimensions obviously increases the probability of tuple incomparability.

References

[Borning *et al.*, 1989] A. Borning, M. Maher, A. Martindale, and M. Wilson. Constraint hierarchies and logic programming. In Martelli M. Levi G., editor, *Proc. 6th ICLP*. MIT Press, 1989.

[Dubois *et al.*, 1993] D. Dubois, H. Fargier, and H. Prade. The calculus of fuzzy restrictions as a basis for flexible constraint satisfaction. In *Proc. IEEE Int. Conf. on Fuzzy Systems*. IEEE, 1993.

[Fargier and Lang, 1993] H. Fargier and J. Lang. Uncertainty in constraint satisfaction problems: a probabilistic approach. *Proc. ECSQARU*. Springer-Verlag, LNCS 747, 1993.

[Fargier *et al.*, 1993] H. Bargier and J. Lang and T. Schiex. Selecting Preferred Solutions in Fuzzy Constraint Satisfaction Problems. *Proc. 1st European Congress on Fuzzy and Intelligent Technologies* (EUFIT). 1993.

[Freuder, 1978] E. Freuder. Synthesizing constraint expressions. *CACM*, 21(11), 1978.

[Freuder, 1988] E. Freuder. Backtrack-free and backtrack-bounded search. In Kanal and Kumar, editors, *Search in Artificial Intelligence*. Springer-Verlag, 1988.

[Freuder and Wallace, 1992] E. C. Freuder and R. J. Wallace. Partial constraint satisfaction. *AI Journal*, 58, 1992.

[Kumar, 1992] V. Kumar. Algorithms for constraint satisfaction problems: a survey. *AI Magazine*, 13(1), 1992.

[Mackworth, 1977] A.K. Mackworth. Consistency in networks of relations. *AI Journal*, 8(1), 1977.

[Mackworth, 1988] A.K. Mackworth. *Encyclopedia of AI*, chapter Constraint Satisfaction, pages 205–211. Springer Verlag, 1988.

[Mackworth and Freuder, 1985] A.K. Mackworth and E.C. Freuder. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *AI Journal*, 25, 1985.

[Montanari, 1974] U. Montanari. Networks of constraints: Fundamental properties and application to picture processing. *Information Science*, 7, 1974.

[Montanari and Rossi, 1991] U. Montanari and F. Rossi. Constraint relaxation may be perfect. *AI Journal*, 48:143–170, 1991.

[Moulin, 1988] H. Moulin. *Axioms for Cooperative Decision Making*. Cambridge University Press, 1988.

[Rosenfeld *et al.*, 1976] A. Rosenfeld, R.A. Hummel, S.W. Zucker. Scene Labelling by Relaxation Operations. *IEEE Trans. on Sys., Man, and Cyb.*, vol. 6. n.6, 1976.

[Ruttkay, 1994] Zs. Ruttkay. Fuzzy constraint satisfaction. *Proc. 3rd Int. Conf. on Fuzzy Systems*, 1994.

[Schiex, 1992] T. Schiex. Possibilistic constraint satisfaction problems, or "How to handle soft constraints?". *Proc. 8th Conf. of Uncertainty in AI*, 1992.

[Zadeh, 1975] L. A. Zadeh. Calculus of fuzzy restrictions. In K. Tanaka L.A. Zadeh, K.S. Fu and M. Shimura, editors, *Fuzzy sets and their applications to cognitive and decision processes*. Academic Press, 1975.