Learning Solution Preferences in Constraint Problems

Francesca Rossi and Alessandro Sperduti

Computer Science Department, University of Pisa

Corso Italia 40, 56125 Pisa, Italy

Tel.: +39-50-887266

Fax: +39-50-887226

E-mail: {rossi,perso}@di.unipi.it

Short title: Learning Solution Preferences in CSPs

Abstract

Usually, not all the solutions of a finite domain constraint problem (CSP) are equally desirable: some of them may be preferred to others. However, classical CSPs do not allow for this more informative kind of knowledge representation. On the other hand, semiring-based CSPs (SCSPs), where a value is associated with each tuple in each constraint, generate solutions with a corresponding value attached, which can be interpreted as the level of preference of that solution. Sometimes, however, even standard SCSPs are not enough, since one may know his/her preferences over some of the solutions but have no idea on how to code this knowledge into the SCSP. In this paper we consider this situation and propose to address it by first defining a classical CSP and giving some examples of solution preferences, and then *learning* the corresponding SCSP which behaves as the initial CSP (that is, it has the same solutions) and matches the preferences specified in the examples. In other words, we use the examples as the training set, and we employ a learning scheme to adjust the values, to be attached to the constraint tuples, such that the resulting solution preferences coincide with the examples. In this way, we make the SCSP framework more flexible, since it can be used also when it is difficult to assign values to tuples and instead it is easier to rate some of the solutions.

1 Introduction

Classical constraint satisfaction problems (CSPs) [Mon74, Mac88] 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 [Fre78, Fre88, MF85, Mac77, Mon74].

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 this reason, many extensions of the classical CSP framework have been proposed in the literature. For example, fuzzy [DFP93, RHZ76, Rut94, Sch92], partial [FW92], probabilistic [FL93], hierarchical [BMMW89]. More recently, all these extensions have been unified in a general framework [BMR95, BMR97], called SCSP, which uses a semiring (that is, a domain plus two operations satisfying certain properties) to associate with each tuple of values for the variables an appropriate "degree of preference", which can also be interpreted as a cost, or an award, or a probability (of being admissible in the real problem), or others. The two semiring operations define a way to combine constraints together.

The above cited extensions of classical CSPs (fuzzy, partial, probabilistic, ...) are all instances of the SCSP framework, once a suitable semiring is chosen. For example, for classical CSPs we have to choose the semiring with just two values (which can be interpreted as *true* and *false*, or *allowed* and *forbidden*), and *logical and* and *logical or* as its operations.

A typical situation where the features of SCSPs are needed occurs when one has some preferences over the solutions of a real-life problem. In fact, usually not all the solutions of a finite domain constraint problem are equally desirable: some of them may be preferred to others. However, classical CSPs do not allow for this more informative kind of knowledge representation. On the other hand, SCSPs, where a value is associated to each tuple in each constraint, generate solutions with a corresponding value attached, which can be interpreted as the level of preference of that solution.

Sometimes, however, even SCSPs are not expressive enough as a modeling tool, since one may know his/her preferences over some of the solutions but have no idea on how to code this knowledge into the SCSP. That is, one has a global idea about the *goodness* of a solution, but does not know the contribution of each single constraint to such a measure. In such a situation, it is difficult both to associate a preference to the other solutions in a compatible way, and to understand the importance of each tuple and/or constraint.

In this paper we consider this situation and propose to address it by using learning techniques. More precisely, we assume to know the level of preference for some solutions (the *examples*), and we propose to learn, from these examples, values to be associated with each constraint tuple, in a way that is compatible with the examples. In other words, we start with a classical CSP, and some rated solutions, and we end up with an SCSP, whose solutions are the same as those of the classical CSP, and are rated consistently with the examples.

There are many approaches to learning in the literature (see [RN95] for a survey of the major approaches and an introduction to the standard terminology). The problem that we consider is however easily described within a framework for *supervised learning* with a *bias* given by the structure of the given CSP (topology and tuples allowed by the constraints) and an algorithm based on *gradient descent*. That is, we assign an initial value to each tuple in each constraint, compute the solutions and their error w.r.t. the examples, and change the values in a way that the error becomes smaller. Then we repeat this operations until we get a reasonable approximation.

Learning solution preferences may be seen as finding a *rating function* which maps tuples to values in a way compatible with the examples. However, not all computational models are universal, in the sense that they can compute any computable function. Here we show, with an example, that SCSPs where only one value is assigned to each tuple are not a universal framework. In fact, given a set of examples and an SCSP as above, there are rating functions which cannot be computed within this framework. We claim that this problem can be solved by assigning more than one value to each tuple, with no limit in general to the number of these values. Note that this extension is only apparent, since it is just another instance of the SCSP paradigm. This is due to fact that the "vectorization" of several semirings is still a semiring (see [BMR95]).

The work that is most related to ours is the one concerned with learning compatibility coefficients for relaxation labeling processes [PR94]. In fact, they use a gradient descent method to learn the coefficient which specify the compatibility of tuples of values for subsets of variables, just like we do. However, there are major differences. First, their approach is tied to the chosen relaxation algorithm (which will then be used to compute the solutions) while ours is independent from it. Second, their training set can only contain optimal solutions, while our examples may concern solutions with any rating. Finally, they associate to each domain a probability distribution of the values in such a domain. This latter feature means that, in a sense, in addition to our two semiring operations, they also have another way to relate constraints to each other. Thus, while the former two differences make them more specific, the latter one is indeed a generalization. We plan to study the meaning of a similar generalization in our framework as well.

This paper is just a first step towards the application of learning techniques within constraint solving frameworks. However, we are confident that this line of research is very promising, and we believe that it can significant advance the practicality of using constraints as a knowledge representation paradigm.

The paper is organized as follows. Section 2 gives the necessary notions about CSPs and SCSPs. Then Section 3 introduces the basic concepts of learning via gradient descent, and Section 4 describes our approach. Finally Section 5 considers the universality problem, and Section 6 discusses the significance of the proposed approach.

2 Standard and Non-Standard Constraint Systems

Standard constraint satisfaction problems (CSPs) [Mon74, Mac88] consist of a set of variables with a finite domain, plus a set of constraints. Each constraint involves a subset of the variables and specifies the tuples of values allowed for those variables. A solution for a CSP is then an assignment of values to all the variables such that all constraints are satisfied.

To achieve a greater expressive power, many non-standard constraint solving frameworks have been proposed in the literature, like fuzzy CSPs [DFP93], partial CSPs [FW92], probabilistic CSPs [FL93], hierarchical CSPs [BMMW89], valued CSPs [SFV95]. The interesting thing is that most of these frameworks can be either cast or easily reduced via an appropriate transformation to the paradigm of semiring-based CSPs (SCSPs) [BMR95]. The basic idea of this paradigm is that each tuple in each constraint is assigned a value (taken from the semiring), to be interpreted as the level of preference for that tuple, or its cost, or any other measurable feature. Then, constraints are combined according to the semiring operations, and the result of such a combination is that each assignment for all the variables has a corresponding semiring value too. Standard CSPs are just SCSPs where the semiring has just two values (to be interpreted as *allowed* and *forbidden*, or, more formally, as *true* and *false*), and the two semiring operations are just *logical and* (to combine constraints together) and *logical or* (to project out some variables). The formal definitions about SCSPs follow. A more extensive treatment of SCSPs can be found in [BMR95, BMR97].

Semirings. A *semiring* S 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., $a \times (b + c) = (a \times b) + (a \times c)$).

Actually, we need to use a structure called *c-semiring*, which is just a semiring where $+$ is idempotent (i.e., $a \in A$ implies $a + a = a$), $\times$ is commutative, and $\mathbf{1}$ is the absorbing element of $+$.

Some properties. The idempotency of the $+$ operation is needed in order to define a partial ordering $\leq_S$ over the set A, which enables us to compare different elements of the semiring. Such a 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 can be used to choose the "best" solution in our constraint problems. Note 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 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 $\mathbf{1} \leq_S a \leq_S \mathbf{0}$.

Constraint systems and problems. 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 number of variables in con) is called the *value* of the constraint. A *constraint problem* P over CS is just a set of constraints over CS.

Combination and projection. The values specified for the tuples of each constraint are used to compute corresponding values for the tuples of values of all the variables, 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 choose the best among all such tuples. More precisely, we can define the operations of *combination* ($\otimes$) and *projection* ($\Downarrow$) over constraints. Consider two constraints $c_1 = \langle def_1, con_1 \rangle$ and $c_2 = \langle def_2, con_2 \rangle$ over CS. Then, 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'. In words, the combination of two constraints c_1 and c_2 generates another constraint which involves all the variables of c_1 and c_2; each tuple of values for such variables, say t, gets a value of the semiring which is obtained by multiplying the values associated by c_1 and c_2 respectively to the two subtuples of t which are the projections of t over the variables of c_1 and c_2 respectively. Moreover, given a constraint $c = \langle def, con \rangle$ over CS, and a subset w of con, its projection over w, written $c \Downarrow_w$, is the constraint $\langle def', con' \rangle$ over CS with $con' = w$ and $def'(t') = \Sigma_{\{t \mid t \downarrow_w^{con} = t'\}} def(t)$. In words, the projection of a constraint c over a subset w of its variables is another constraint which involves only the variables in w; each tuple of values (for the variables in w), say t, gets a value of the semiring which is obtained by summing up all possible extensions of t to the other variables of c not in w.

Example (constraint combination and projection). Consider two constraints $c_1 = \langle def_1, con_1 \rangle$ and $c_2 = \langle def_2, con_2 \rangle$ such that c_1 connect the variables x and y (thus $con_1 = \{x, y\}$) and c_2 connects the variables y and z (thus $con_2 = \{y, z\}$). Assume also that each of the variables has values over the domain $D = \{a, b\}$. Thus functions def_1 and def_2 map pairs of elements of D to elements of the semiring set. In particular, let def_1 assign value v_1 to tuple $\langle a, a \rangle$, value v_2 to tuple $\langle a, b \rangle$, value v_3 to tuple $\langle b, a \rangle$, and value v_4 to tuple $\langle b, b \rangle$. Similarly, function def_2 assigns value v_5 to tuple $\langle a, a \rangle$, value v_6 to tuple $\langle a, b \rangle$, value v_7 to tuple $\langle b, a \rangle$, and value v_8 to tuple $\langle b, b \rangle$. Of course the semiring set must contain all values v_i, for $i = 1, \ldots, 8$, but these values are not necessarily all different (we have given them different names for sake of clarity). Constraint $c_1 \otimes c_2$ connects all three variables x, y, and z, and assigns semiring values to triples of values from D in the following way. Tuple $t = \langle a, a, a \rangle$ gets the value $v_1 \times v_5$, because the subtuple of t which refers to the variables connected by c_1 (that is, $\langle a, a \rangle$) gets the value v_1 from c_1, and the subtuple which refers to the variables connected by c_2 (that is, $\langle a, a \rangle$ again) gets the value v_5 from c_2. Similarly, tuple $\langle a, a, b \rangle$ gets the value $v_1 \times v_6$, $\langle a, b, a \rangle$ gets the value $v_2 \times v_7$, $\langle a, b, b \rangle$ gets the value $v_2 \times v_8$, $\langle b, a, a \rangle$ gets the value $v_3 \times v_5$, $\langle b, a, b \rangle$ gets the value $v_3 \times v_6$, $\langle b, b, a \rangle$ gets the value $v_4 \times v_7$, $\langle b, b, b \rangle$ gets the value $v_4 \times v_8$. As an example of constraint projection, consider instead the projection of constraint c_1 over the variable x. That is, we want to project out variable y from c_1. Then, constraint $c_1 \Downarrow_x$ has $\{x\}$ as its type, and will assign a semiring value to each tuple of one element of the domain. More precisely, tuple $\langle a \rangle$ will get the value $v_1 + v_2$, because there are two extensions of such a tuple to the other variable of c_1 (that is, y), one with assigned value v_1 and the other one with assigned value v_2. Similarly, tuple $\langle b \rangle$ will get the value $v_3 + v_4$.

Solutions. Given a constraint problem C over a constraint system CS, the *solution* of P is a constraint defined as $Sol(P) = (\bigotimes C)$, 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 all the variables by the whole problem. Such constraint provides, for each tuple of values of D for all the variables, an associated value of A. More precisely, consider any tuple t with as many elements of D as the number of all variables (in the following such tuples will be called *n-tuples*). Then the corresponding value $val(t)$ can be obtained by (see definition of combination)

$$val(t) = def_1(t \downarrow^V_{con_1}) \times \ldots \times def_k(t \downarrow^V_{con_k})$$

where V is the set of all variables, the set of all constraints is $C = \{c_1, \ldots, c_k\}$, and $c_i = \langle def_i, con_i \rangle$, for $i = 1, \ldots, k$. Given two n-tuples t and t', we say that t is better than t' if $val(t) \leq_S val(t')$. Note that this, by definition of $\leq_S$, means that $val(t) + val(t') = val(t)$. Note also that t and t' could be incomparable, since $\leq_S$ is a partial order. Given an SCSP P, we will call n-tuples(P) the set of all n-tuples of P. Then we will consider the function f :n-tuples(P) $\rightarrow A$ such that, for each n-tuple t, $f(t) = val(t)$. That is, the function which assigns to each n-tuple the corresponding value. We will call this f the *solution-rating function* of the given SCSP.

Example (concrete semirings). Let us recall the general form of an SCSP, which is $\langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$. As noted above, classical CSPs are just SCSPs where $A = \{true, false\}$, $+ = \vee$, $\times = \wedge$, $\mathbf{0} = false$, and $\mathbf{1} = true$. It is easy to check that the above described operations of constraint combination and projection reduce indeed in this case to the usual operations for the satisfiability of a set of constraints and for the elimination of some variables, respectively.

Other interesting semirings can be thought. For example, the paradigm of fuzzy constraint problems (FCSPs) [DFP93, RHZ76, Rut94, Sch92], where tuples get assigned values between 0 and 1 (to be interpreted as their level of preference), and the goal is to maximize the minimum level of preference, can be described by using the c-semiring where $A = \{x \mid x \text{ in } [0, 1]\}$, $+ = max$, $\times = min$, $\mathbf{0} = 0$, and $\mathbf{1} = 1$.

Another useful instance is the one describing probabilistic CSPs [FL93], where each tuple has a probability (to be allowed in the real problem), and the goal is to maximize the probability that we are modeling exactly the given real-life problem. In this case the semiring to be used is $\langle \{x \mid x \text{ in } [0, 1]\}, max, \times, 0, 1\rangle$. This instance is very useful because it allows to treat situations where only partial knowledge about the real problem is available.

Partial constraint problems [FW92] are CSPs which are overconstrained and thus would not have a solution if treated via the classical paradigms for constraint solving. In this case one may want to get a partial solution which satisfies as many constraints as possible. In this case the semiring to be used is $\langle N, max, +, 0, +\infty\rangle$.

The last instance we cite is the one describing optimization problems, where each tuple has an associated cost and the goal is to minimize the global cost. Here the semiring is $\langle \Re^+, min, +, +\infty, 0\rangle$.

3 Learning

In this paper we consider *inductive learning* (see [RN95] for a description of many learning paradigms), since it turns out to be best suited for solving our problem. Inductive learning can be defined as the ability of a system to induce the correct structure of a map d which is known only for particular inputs. More formally, defining an example as a pair $(x, d(x))$, the computational task is as follows: *given a collection of examples of d, i.e., the training set, return a function h that approximates d.* The function h is called a hypothesis.

A common approach to inductive learning, especially in the context of neural networks, is to evaluate the quality of a hypothesis h (on the training set) through an *error function*. An example of popular error function, that can be used over the reals, is the sum of squares error [HKP91]:

$$E = \frac{1}{2} \sum_{i=1}^{n} (d(x_i) - h(x_i))^2, \tag{1}$$

where $(x_i, d(x_i))$ is the i-th example of the training set. When instead the domain is a (totally or partially ordered) discrete set, then a suitable notion of metric has to be built, so that a distance between pairs of elements (i.e., $d(x_i)$ and $h(x_i)$) can be derived and an error function formula defined.

Given a starting hypothesis h_0, the goal of learning is to minimize the error function E by modifying h_0. There are several approaches which can be followed in order to minimize E. A very simple and popular technique consists in using a definition of h which depends on a set of internal parameters W, i.e., $h \equiv h_W$, and then adjusting these parameters. This adjustment, again, can be formulated in different ways, depending on whether the domain is isomorphic to the reals or not. In fact, the usual way to be used over the reals, and if h_W is continuous and derivable, is to follow the negative of the *gradient* of E with respect to W. This technique is called *gradient descent* [HKP91]. Specifically, the set of parameters W is initialized to small random values at time $\tau = 0$ and updated at time $\tau + 1$ according to the following equation

$$W(\tau + 1) = W(\tau) + \Delta W(\tau), \tag{2}$$

where

$$\Delta W(\tau) = -\eta \frac{\partial E}{\partial W(\tau)}, \tag{3}$$

and η is the step size used for the gradient descent. If instead we cannot use the gradient descent, then we may resort to some form of local search, in order to pass from the current values to new values which are adjacent in the domain and make the value of the error function smaller.

Learning is stopped when a minimum of E is reached. Note that, in general, there is no guarantee that the found minimum is global.

4 Our approach

Learning in our context can be used to find a suitable set of values to be assigned to the constraint tuples of a given CSP. In this way we get a corresponding SCSP where the value associated to each solution is consistent with the desired rating for that solution. More precisely, consider a CSP P. We recall that a CSP can be seen as an SCSP where the semiring is $\langle\{true, false\}, \vee, \wedge, false, true\rangle$. That is, allowed tuples are assigned the value *true*, and forbidden ones the value *false*. Given such a CSP, its solutions are those n-tuples t such that $val_P(t) = true$.

We assume now that for some solutions of P we are given a desired rating $d(t)$ defining the goodness of t. Let m be the number of such solutions. That is, we have the set of *examples* $TR = \{(t_1, d(t_1)), \ldots, (t_m, d(t_m))\}$. We recall that such a function d : {n-tuples} $\rightarrow D$, where $D = \{d(t_i)$ such that $(t_i, d(t_i)) \in TR\}$, is partial since only some solution ratings are specified. However, giving the examples is not enough, because we must also have an idea of how to combine and compare the values given as examples. Therefore, together with the examples, we must also be given the following two objects:

- a semiring containing the values in the examples. That is, a semiring $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ such that $D \subseteq A$.
- a distance function over such a semiring. That is, a function $dist : A^2 \rightarrow \Re$,

which gives the distance between two elements of the semiring. A distance function must have the following features: for all $x, y, z \in A$, $dist(x, y) \geq 0$, $dist(x, y) = 0$ iff $x = y$, $dist(x, y) = dist(y, x)$, and $dist(x, y) \leq dist(x, z) + dist(z, y)$.

Once we have the above (initial given CSP P, examples, semiring S, and distance function), our goal is to define a corresponding SCSP P' over the given semiring S such that

- P and P' have the same graph topology (that is, they have the same variables and corresponding constraints have the same type);
- for each n-tuple t such that $(t, d(t))$ is an example, $dist(val_{P'}(t), d(t)) < \epsilon$, where $\epsilon > 0$ and small.

Usually, if the chosen semiring set is the set of reals, then the distance function is implicitly defined in the obvious way as $dist(val_{P'}(t), d(t)) =\mid val_{P'}(t) - d(t) \mid$. We recall also that, if we want to use the gradient descent technique to learn the SCSP P', then the $\times$ operation of the chosen semiring must be continuous and derivable.

Given the first condition (on the graph topology), the only free parameters that can be arbitrarily chosen in order to satisfy the other conditions are the values to be associated to each constraint tuple. For each constraint $c_i = \langle def_i, con_i \rangle$ in P, consider $S_i = \{t_{ij}$ such that $def_i(t_{ij}) = true\}$ (that is, the set of tuples allowed by that constraint). The idea is to associate, in P', a free parameter w_{ij} (note that w_{ij} must belong to the set of the chosen semiring) to each t_{ij} in S_i. With the other tuples, we associate the constant $\mathbf{0}$, which, we recall, is the worst element of the semiring (w.r.t. $\leq_S$). Given this association, the value assigned to each n-tuple t in P' is

$$val_{P'}(t) = \prod_{i=1}^{k}(\sum_{j=1}^{|S_i|} \text{subtuple}(t_{ij}, t, i) \times w_{ij}), \tag{4}$$

where

- $\prod$ refers to the $\times$ operator of the semiring of P' and k is the number of constraints in P,
- $\sum$ refers to the $+$ operation of the semiring, and
- $\text{subtuple}(t_{ij}, t, i) = \mathbf{1}$ if $t \downarrow^{V}_{con_i} = t_{ij}$, $\mathbf{0}$ otherwise.

Note that, for each i, there is exactly one j such that $\text{subtuple}(t_{ij}, t, i) = \mathbf{1}$. Let l_i be such a j. Thus $\sum_{j=1}^{|S_i|} \text{subtuple}(t_{ij}, t, i) \times w_{ij} = w_{il_i}$, and therefore $val_{P'} = w_{1l_1} \times \ldots \times w_{kl_k}$.

The values of the free parameters w_{ij} may be obtained via a minimization of the error function, which will be defined according to the distance function of the semiring. In the case of a generic semiring, it will be a function $f_{error} : \Re^m \rightarrow \Re$, where m is the size of the training set, which takes the m distances between the examples and the current values of the n-tuples, and computes the total error given by such distances.

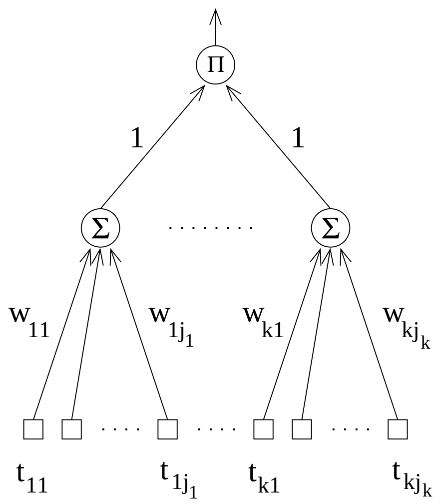

Figure 1: The neural network which computes the value of an n-tuple.

When minimizing such a function, one has to assure that, every time the values of the free parameters are changed in order to make the error function smaller, the new values are in the semiring set as well. In other words, no change can be made to a value which is already **1** (resp., **0**) and the learning technique would like to make even smaller (resp., bigger).

Note that equation (4) can be cast in a structure similar to a two-layer feed-forward neural network [HKP91] as in Figure 1, where the input units are as many as the constraint tuples in the whole problem and the hidden units are as many as the constraints. Each hidden unit hu_i receives inputs from the units representing the tuples of the corresponding constraint c_i. The connection from unit u_{ij} representing tuple t_{ij} to hu_i has weight w_{ij}. Given an n-tuple t, the output value of each input unit u_{ij} corresponds to the value returned by function $subtuple(t_{ij}, t, i)$. Each hidden unit performs the weighted sum of its inputs, where the sum uses the $+$ operation of the semiring. The connections between the hidden units and the output unit all have weight **1**, and the output unit computes the product of its input (where the product uses the $\times$ operator of the semiring).

Example (over the reals). Assume that the examples are real numbers and that the chosen c-semiring is $\langle \Re^+, min, +, +\infty, 0\rangle$. This corresponds to the case of the so-called weighted CSPs where the values associated to the constraint tuples are interpreted as weights and the goal is to minimize the sum of the weights. As noted above, SCSPs based on this semiring can be used to represent classical optimization problems. Note that the $\times$ operation of the semiring, that is, standard sum, is a function with two arguments which is continuous and derivable. Thus we can use the gradient descent technique. Also, we recall that the distance function to be used here is the usual one over the reals, that is $dist(val_{P'}(t), d(t)) = \mid val_{P'}(t) - d(t) \mid$.

In this case equation (4) becomes:

$$val_{P'}(t) = \sum_{i=1}^{k} (min_{j=1,\ldots,|S_i|} \text{ subtuple}(t_{ij}, t, i) + w_{ij})$$

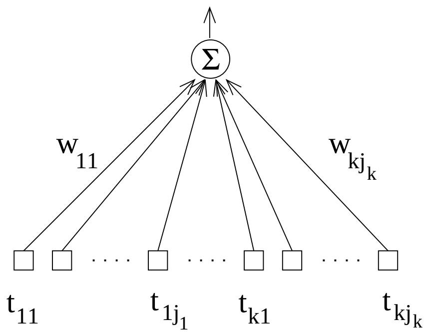

Figure 2: The neural network which computes the value of an n-tuple in the example.

and the error function is

$$E = \frac{1}{2}\sum_{i=1}^{m}(d(t_i) - val_{P'}(t_i))^2.$$

Note that, by definition of subtuple and min, the network of Figure 1 can be reduced to a one-layer net (see Figure 2), if the inputs are transformed according to the following formula: output$(u_{ij}) := e^{-subtuple(t_{ij},t,i)}$, where u_{ij} is an input unit. In this way, when function subtuple has value $+\infty$, we get the value 0, while when it has the value 0, we get 1. This allows us to remove the min operation, thus making the function computed by the net continuous and derivable. Therefore one can apply a gradient descent technique to minimize the error function.

By deriving the error function w.r.t. one free parameter w_{ij} we get

$$\Delta w_{ij}(\tau) = -\eta\frac{\partial E}{\partial w_{ij}(\tau)} = -\eta \cdot \sum_{i=1}^{m}((d(t_i) - val_{P'}(t_i)) \cdot output(u_{ij}))$$

Thus only those weights which are associated to tuples which are contained in at least one n-tuple in an example will be changed. We recall that one has to make sure that all weights remain positive. This can be done by modifying the learning rule in such a way that weights that would become negative are not modified. See [PR94] for an example of this technique.

If the examples are not real numbers and/or the given semiring is not the one considered above, then we have already said that we must use a local search technique which, using the given distance function, changes the free parameters by moving from one value of the semiring to the closest one which is better. However, this may require having a library of (possibly very different) local search techniques, among which to choose depending on the semiring to be used. An alternative to this, which is much more efficient in terms of space and development time, but may lack precision, is to always use the gradient descent technique over the reals (thus, as noted above, we may implement the search for the minimum via a neural net), and to inject any given semiring into the semiring above over the reals, or any other semiring over the reals where the two operations are continuous and derivable. In other words, given a semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$, we need a mapping from S into a semiring of choice $S' = \langle \Re, +', \times', \mathbf{0}', \mathbf{1}' \rangle$ which injects A into $\Re$ (that is, the mapping must not map any two distinct elements of A into the same real number), and transforms $+$ in $+'$, $\times$ in $\times'$, $\mathbf{0}$ in $\mathbf{0}'$, and $\mathbf{1}$ in $\mathbf{1}'$ accordingly. Of course the transformation of the two operations may introduce some imprecision, since some functions, like min or max, which are not continuous and derivable, may only be approximated by continuous and derivable functions (like $+'$ and $\times'$ must be). There is also another issue to consider: when the minimum is found, and thus the weights for the tuples are computed, such weights are real numbers and not elements of A as the user would expect. Thus we need a subsequent phase in which they are transformed back into elements of A. For examples, if A is the set of natural numbers, then we may approximate the obtained real numbers by taking the closest natural number. More formally, one must define both a mapping from S to S' and a mapping from S' to S. Since S can be thought of as an abstraction of S', it is possible that classical abstract interpretation techniques [CC92] may be of some help in defining and using such mappings.

5 On the Representational Power

The net in Figure 2 can only compute linear functions. Thus, if the desired rating function cannot be expressed as a linear function of the values associated to the constraint tuples, it cannot be computed within this framework. Thus one can either be content of the approximation that this net structure can obtain, or it is necessary to change the model in order to make it universal, i.e., where every desired continuous function can be computed. General neural networks with hidden units are indeed universal.

This universality problem depends on the fact that only a single value is associated to each tuple in each constraint. This can be seen also in the following example.

Take an SCSP with two binary constraints sharing one variable, one allowing two tuples and the other one three tuples. If all the tuples in one constraint are compatible with all those in the other constraint, there will be six solutions and a solution-rating function can assign six different values to them. However, the values to be assigned to the tuples in the constraints are only five. If the combination is obtained via standard sum, then we have a system of six linear equations with only five free parameters, which not necessarily has a solution.

It is easy to see that this problem can be eliminated if one assigns more than one value to each tuple, with no limit in general to the number of values. In fact, by using a suitable number of values, one can always obtain a number of free parameters which matches the number of equations, thus assuring that a solution always exists.

The important thing is that this extension does not lead us outside SCSPs, since the vectorization of several c-semirings is again a c-semiring (see [BMR95] for the formal proof). Such a vectorization can be defined as follows.

Consider n c-semirings $S_i = \langle A_i, +_i, \times_i, \mathbf{0_i}, \mathbf{1_i} \rangle$, for $i = 1, \ldots, n$. Then we define the structure

$$Comp(S_1, \ldots, S_n) = \langle \langle A_1, \ldots, A_n \rangle, +, \times, \langle \mathbf{0_1}, \ldots, \mathbf{0_n} \rangle, \langle \mathbf{1_1} \ldots \mathbf{1_n} \rangle \rangle.$$

Given $\langle a_1, \ldots, a_n\rangle$ and $\langle b_1, \ldots, b_n\rangle$ such that $a_i, b_i \in A_i$ for $i = 1, \ldots, n$, the two semiring operations are defined as follows:

- $\langle a_1, \ldots, a_n\rangle + \langle b_1, \ldots, b_n\rangle = \langle a_1 +_1 b_1, \ldots, a_n +_n b_n\rangle$, and
- $\langle a_1, \ldots, a_n\rangle \times \langle b_1, \ldots, b_n\rangle = \langle a_1 \times_1 b_1, \ldots, a_n \times_n b_n\rangle$.

According to the definition of $\leq_S$, such an ordering for $S = Comp(S_1, \ldots, S_n)$ is as follows. Given $\langle a_1, \ldots, a_n\rangle$ and $\langle b_1, \ldots, b_n\rangle$ such that $a_i, b_i \in A_i$ for $i = 1, \ldots, n$, we have $\langle a_1, \ldots, a_n\rangle \leq_S \langle b_1, \ldots, b_n\rangle$ if and only if $\langle a_1 +_1 b_1, \ldots, a_n +_n b_n\rangle = \langle b_1, \ldots, b_n\rangle$. Note that, since the tuple elements are completely independent, $\leq_S$ is in general a partial order, even though each of the $\leq_{S_i}$ is a total order.

The problem now is that, if we associate a vector of values to any tuple, then also solutions will have vectors of values associated to them. This is in contrast with the assumption that the examples assign only one value of the semiring to each solution in the training set. Consider an n-tuple t whose associated vector of values is $\langle a_1, \ldots, a_n\rangle$. In order to obtain a single value for t (since we assumed that we have been given a single value for some n-tuples), one has to define a function $g : A^n \to A$ which combines the n values a_i into a single element of the semiring. Function g has to be chosen in such a way that, combined with equation (4), allows one to approximate any function. For example, in the case of the reals, and to remain within continuous and derivable functions, we may compute the square of the modulus of the vector, that is, $\sum_{i=1}^{n}(a_i)^2$.

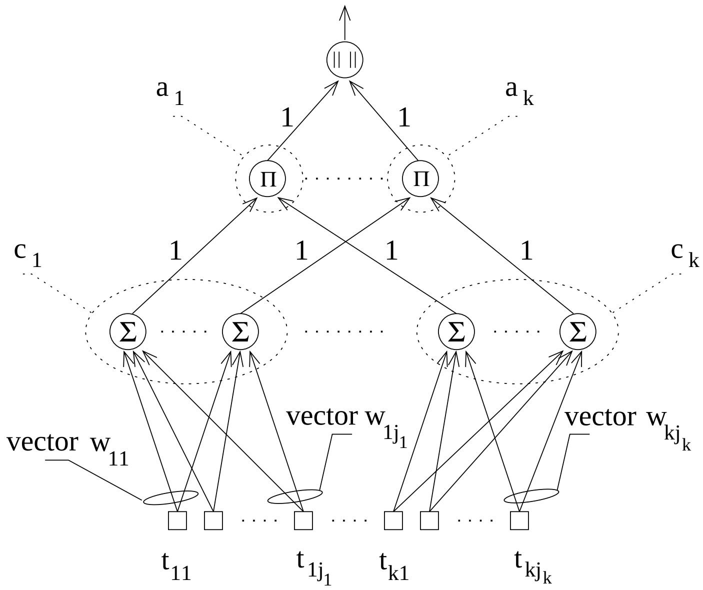

Figure 3: The neural network which computes the value of an n-tuple in the extended framework.

The neural network which computes the value of an n-tuple in this extended framework for the special case of the reals has two layers of hidden units and the same input units as in Figure 1. For each constraint c_i there are as many hidden units, in the lower hidden layer, as the dimension of the vector. Each input unit corresponding to tuple t_{ij} sends its output to all the hidden units corresponding to constraint c_i, and the weights on these connections correspond to the components of the vector. Thus the lower layer of hidden units can be partitioned in k subsets, each with n units, where k is the number of constraints and n is the dimension of the vector. Moreover, each of these n units in a subset are associated to a component of the vector. In the higher layer of hidden units, there are as many units as the dimension of the vector, and the i-th unit of this layer takes input from all the i-th units of the subsets of the lower layer, with weight 1. These upper-layer hidden units compute the product of their inputs, where by product we mean the $\times$ operator. Finally, one output unit, connected to each upper-layer hidden unit with weight 1, computes the sum of the squares of its inputs (see Figure 3). Thus the choice of function g as the square of the modulus of the vector allowed us to obtain a network with hidden units, which is able to approximate any function (given a suitable dimension of the vectors).

6 Discussion and Conclusions

In this paper we have considered the problem of assigning values to constraint tuples of a CSP in a way compatible with a set of examples of solution ratings. The approach we have proposed is based on an inductive learning algorithm implementing either a local search or a gradient-descent technique. We have also discussed the relationship between this approach and a class of feed-forward neural nets (in reality, the nets we considered are not standard neural nets, since not all units compute the same function, which usually is a sigmoid) where learning may be realized via back-propagation

[RM86]. The interesting aspect of such a relationship is that these nets directly match the topology of the SCSP, and thus the weights obtained by the learning algorithm can directly be used in the SCSPs as values to be assigned to the constraint tuples. We also noted that SCSPs where only one value is associated with each tuple have a limited expressive power, since they cannot model any desired rating function. However, we have claimed that by using more than one value for each tuple, without any bound on the number of these values, we can get a universal framework within the SCSP paradigm.

We believe that adding the capability of learning solution preferences to the SCSP framework makes it more widely applicable to real problems. In fact, in general it is not easy to find the values to be associated to all the tuples (which may be many) in order to get an SCSP which models the real-life problem at hand in a faithful way. Instead, it could be much easier to rate only some solutions, and then let the system find reasonable values for the tuples automatically, in a way consistent with the given ratings.

Obviously, this paper represents just a first step towards the application of learning techniques within SCSPs. In fact, it is definitely necessary to obtain some experimental results which show how this approach behaves on real and/or randomly generated constraint problems to be able to assess the significance of the proposed method. Moreover, there are still many issues regarding learning in SCSPs which need to be addressed in the experimentation phase (like generalization, learning speed, size of training set, and so on) and which could lead to more or less satisfactory trade-offs. However, all such problems can be tackled by taking advantage of the neural network literature, where they have been already deeply considered. We do not have the experimental results yet but we believe that nevertheless this paper, which presents the first formalized and general approach to learning in constraint problems, has at least the value of introducing these ideas in the constraint solving field and to point out the crucial issues in learning constraint problems. This can be useful to define in the future several specialized learning algorithms which will probably be efficient in their (possibly narrow) field of application and thus will help the user in modeling its real-life problems as constraint problems.

References

[BMR95] S. Bistarelli, U. Montanari, and F. Rossi. Constraint Solving over Semirings. In *Proc. IJCAI95*. Morgan Kaufman, 1995.

[BMR97] S. Bistarelli, U. Montanari and F. Rossi. Semiring-based Constraint Solving and Optimization. To appear in Journal of the ACM.

[BMMW89] A. Borning, M. Maher, A. Martindale, and M. Wilson. Constraint hierarchies and logic programming. In Martelli M. Levi G., editor, *Proc. 6th*

International Conference on Logic Programming. MIT Press, 1989.

[CC92] P. Cousot and R. Cousot. Abstract Interpretation and Application to Logic Programming. In The Journal of Logic Programming. Volume 13 (2 and 3), 1992.

[DFP93] D. Dubois, H. Fargier, and H. Prade. The calculus of fuzzy restrictions as a basis for flexible constraint satisfaction. In *Proc. IEEE International Conference on Fuzzy Systems.* IEEE, 1993.

[FL93] H. Fargier and J. Lang. Uncertainty in constraint satisfaction problems: a probabilistic approach. In *Proc. European Conference on Symbolic and Qualitative Approaches to Reasoning and Uncertainty (ECSQARU).* Springer-Verlag, LNCS 747, 1993.

[Fre78] E. C. Freuder. Synthesizing constraint expressions. *Communication of the ACM*, 21(11), 1978.

[Fre88] E. C. Freuder. Backtrack-free and backtrack-bounded search. In Kanal and Kumar, editors, *Search in Artificial Intelligence.* Springer-Verlag, 1988.

[FW92] E. C. Freuder and R. J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58, 1992.

[HKP91] J. Hertz, A. Krogh, and R. G. Palmer. Introduction to the Theory of Neural Computation. Addison-Wesley, 1991.

[Mac77] A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1), 1977.

[Mac88] A.K. Mackworth. *Encyclopedia of AI*, chapter Constraint Satisfaction, pages 205–211. Springer Verlag, 1988.

[MF85] A. K. Mackworth and E. C. Freuder. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, 25, 1985.

[Mon74] U. Montanari. Networks of constraints: Fundamental properties and application to picture processing. *Information Science*, 7, 1974.

[PR94] M. Pelillo and M. Refice. Learning Compatibility Coefficients for Relaxation Labeling Processes. *IEEE Transactions on Patterns Analysis and Machine Intelligence*, vol.16, n.9, 1994.

[RM86] D. E. Rumelhart and J. L. McClelland. Parallel Distributed Processing: Explorations in the Microstructure of Cognition. MIT Press, 1986.

[RHZ76] A. Rosenfeld, R.A. Hummel, and S.W. Zucker. Scene labelling by relaxation operations. *IEEE Transactions on Systems, Man, and Cybernetics*, 6(6), 1976.

[RN95] S. Russell and P. Norvig. Artficial Intelligence: A Modern Approach. Prentice Hall, 1995.

[Rut94] Zs. Ruttkay. Fuzzy constraint satisfaction. In *Proc. 3rd International Conference on Fuzzy Systems*, 1994.

[Sch92] T. Schiex. Possibilistic constraint satisfaction problems, or "how to handle soft constraints?". In *Proc. 8th Conf. of Uncertainty in AI*, 1992.

[SFV95] T. Schiex, H. Fargier, and G. Verfaille. Valued Constraint Satisfaction Problems: Hard and Easy Problems. In *Proc. IJCAI95*. Morgan Kaufmann, 1995.