Acquiring both constraint and solution preferences in interactive constraint systems

F. Rossi and A. Sperduti
Dept. of Pure and Applied Mathematics
University of Padova, Italy
Email: {frossi,sperduti}@math.unipd.it

Abstract

Constraints are useful to model many real-life problems. Soft constraints are even more useful, since they allow for the use of preferences, which are very convenient in many real-life problems. In fact, most problems cannot be precisely defined by using hard constraints only.

However, soft constraint solvers usually can only take as input preferences over constraints, or variables, or tuples of domain values. On the other hand, it is sometimes easier for a user to state preferences over entire solutions of the problem.

In this paper, we define an interactive framework where it is possible to state preferences both over constraints and over solutions, and we propose a way to build a system with such features by pairing a soft constraint solver and a learning module, which learns preferences over constraints from preferences over solutions. We also describe a working system which fits our framework, and uses a fuzzy constraint solver and a suitable learning module to search a catalog for the best products that match the user's requirements.

1 Motivation and main idea

Constraints [25], that is, restrictions over the possible combinations of values for some variables, have proven to be very useful in many application domains. However, constraints are also a rather rough way to describe real-life problems, and sometimes it may be more natural to use preferences (also called *soft constraints*) [4, 2, 23, 22, 10, 7, 21, 12] rather than strict requirements. For example, when searching a catalog for the best product that matches some user requirements, using hard constraints to express such requirements, as in [14], could either generate an over-constrained problem (thus no solution will be proposed to the user) or a solvable problem where all solutions are equally preferred by the system. Using soft constraints instead, a solution is always found. Moreover, the solutions are ranked according to the optimization criteria used by the soft constraint system.

Soft constraints allow for preference levels to be associated to either constraints, or variables, or tuples within the constraints. Then, two operators allow for preference combination (to generate the preference of an entire solution from the preferences of the single constraints) and for preference comparison (to decide which preference level is better and which one is worse).

Sometimes, however, one may know his/her preferences over some of the solutions but have no idea on how to code this knowledge into the constraint problem in terms of preferences over constraints or tuples. 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. Another typical situation occurs when one has just a rough estimate of the preferences, either for the tuples or for the constraints, but has some additional information about the solution ranking function. Finally, another related scenario shows a user which could fill all the preferences, but doesn't want to spend too much time in the modelling phase and prefers to just give some examples of solution ratings.

Considering all this, we claim that preferences, both over constraints and over solutions, are certainly very useful and expressive. However, up to now constraint systems allowed only for preferences over constraint or tuples, while they should also allow for preferences over solutions. This would make the constraint system more general, flexible, and user-friendly.

We therefore propose a soft constraint solving and modelling framework where a user can interact with a constraint solver in two ways: either by posting the usual constraint preferences, or by stating preferences over solutions proposed by the system. The solution and modelling processes are thus heavily interleaved, since posting preferences can be seen as part of the modelling phase. Moreover, the two tasks can be partially done at each interaction, thus envisioning an iterative process where better and better solutions (that is, closer to the user's desiderata) are proposed by the system.

Apart from the iterative process just described, to build a constraint system that fits this framework requires having a constraint solver that is capable of handling both constraint and solution preferences. We propose to achieve this by adding to a usual soft constraint solver a learning module, which, taken some examples of solution ratings, induces the appropriate constraint preferences to match the examples. In this way, standard constraint solvers can be used, and the only piece that needs to be built is the learning module.

In [18], machine learning techniques based on gradient descent are used to infer constraint preferences from solution ratings, and in [1] several experimental results exhibit small errors and show that a reasonable number of examples is needed. In [1], we also proposed, besides the usual batch operating mode where all solution rating examples are given at the beginning of the learning phase, another mode where only some examples at a time are considered, leading to an incremental anytime learning algorithm. In [13, 20] the same learning idea has been applied to temporal constraints with preferences, but again all the examples were needed at the beginning.

In this paper we view the learning phase in a more general interactive setting, and we propose to use it not on examples to be given by the user, but on the best solutions proposed so far by the constraint system, as a way to ease the modelling task by successive refining without asking for too much information from the user, while hopefully reaching a good trade-off between quality of learning and efficiency. So the main contribution of this paper is to show how to combine constraint solving and learning in an interactive framework, for a large family of constraint classes.

We also give a description of a working system which instantiates our framework by focussing on fuzzy constraints and using both constraint solving and learning techniques to interact with a user. This system is used to search a catalog of a real estate agency for the houses that best match a user's preferences.

Other constraint learning approaches have been proposed in the literature. In [5], the proposed system learns a constraint from examples of allowed and forbidden assignments, and a constraint is seen as a concept described via a certain language. Thus constraints are not soft and the system builds from scratch a constraint, while in our system we assume the constraint graph to be given and the user defines (via a direct specification or solution feedback) his preferences over the constraints of such a graph. In [17], constraints are given strength levels and, in case of over-constrained problems, the system builds a dialog with the user to agree on the consraints to be relaxed, according to the hierarchy, in order to reach a solvable problem. Hierarchical constraints are a special class of soft constraints, so our approach is more general since it allows for a variety of soft constraint classes.

The paper is organized as follows. Section 2 gives the basic notions about soft constraints. Then, Section 3 introduces the basic concepts of learning via gradient descent, and gives the basic idea in using it for learning soft constraints. Section 4 describes the interactive modelling and solving framework we propose, and how the learning and the solving module cooperate to find the best solutions. Finally, Section 5 describes our system for on-line fuzzy constraint solving and learning, Section 6 goes through the steps of an example of use of our system, and Section 7 summarizes the results of the paper and hints at possible lines for future work.

This paper is an improved and more detailed version of [19].

2 Soft constraints

Standard finite domain constraint satisfaction problems (CSPs) [25] 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. On the contrary, *soft* constraints do not say if a tuple of values for some variables is allowed or not, but rather *at which level* it is allowed. To describe such problems, in this paper we use the paradigm of semiring-based CSPs (SCSPs) [2]: 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. The formal definitions about SCSPs follow. A more extensive treatment of SCSPs can be found in [2].

A *semiring* is a tuple $\langle A, +_s, \times_s, \mathbf{0}, \mathbf{1}\rangle$ such that

- A is a set and $\mathbf{0}, \mathbf{1} \in A$;
- $+_s$, the additive operation, is commutative, associative and $\mathbf{0}$ is its unit element;
- $\times_s$, the multiplicative operation, is associative, distributes over $+_s$, $\mathbf{1}$ is its unit element and $\mathbf{0}$ is its absorbing element.

A *c-semiring* is a semiring in which $+_s$ is idempotent (i.e., $a +_s a = a, a \in A$), $\mathbf{1}$ is its absorbing element, and $\times_s$ is commutative. These additional properties (w.r.t. usual semirings) are required to cope with the usual nature of constraints.

C-semirings allow for a partial order relation $\leq_S$ over A to be defined as $a \leq_S b$ iff $a +_s b = b$. Informally, $\leq_S$ gives us a way to compare tuples of values and constraints, and $a \leq_S b$ can be read *b is better than a*. Moreover, one can prove that:

- $+_s$ and $\times_s$ are monotone on $\leq_S$;
- $\mathbf{0}$ is its minimum and $\mathbf{1}$ its maximum;
- $\langle A, \leq_S\rangle$ is a complete lattice where:
  - for all $a, b \in A$, $a +_s b = lub(a, b)$ (where *lub*=least upper bound);
  - if $\times_s$ is idempotent, then $\langle A, \leq_S\rangle$ is a complete distributive lattice and $\times_s$ is its greatest lower bound (*glb*).

We assume to work with a finite set of variables V, a finite set of domain elements D, and a semiring $S = (A, +_s, \times_s, \mathbf{0}, \mathbf{1})$. In this scenario, a *soft constraint* is a pair$\langle def, con\rangle$, where $con \subseteq V$ and $def : D^k \to A$ (where k is the number of variables in con). A *soft constraint problem P* is just a set of soft constraints.

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 operation $\times_s$. Then, in order to choose the best among the solutions, the ordering induced by the other operation ($+_s$) is used. 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 $val(t) = def_1(t \downarrow^V_{con_1}) \times_s \ldots \times_s def_k(t \downarrow^V_{con_k})$, 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 $val(t) +_s val(t') = val(t)$. 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.

Many constraint formalisms can be cast in this semiring-based framework: it is enough to choose the appropriate semiring. Classical CSPs [25] are just SCSPs over the c-semiring $\langle A, +_s, \times_s, \mathbf{0}, \mathbf{1}\rangle$, where $A = \{true, false\}$, $+_s = \vee$, $\times_s = \wedge$, $\mathbf{0} = false$, and $\mathbf{1} = true$. Here constraint combination and projection reduce to the usual operations for the satisfiability of a set of constraints and for the elimination of some variables, respectively. Another example is fuzzy constraint problems (FCSPs) [21, 22], where tuples get assigned real values between 0 and 1 (to be interpreted as their level of preference), and the goal is to maximize the minimum level of preference: the c-semiring to use here has $A = \{x \mid x \text{ in } [0, 1]\}$, $+_s = max$, $\times_s = min$, $\mathbf{0} = 0$, and $\mathbf{1} = 1$.

Figure 1 shows a fuzzy CSP. Variables are inside circles, constraints are represented by undirected arcs, and semiring values are written to the right of the corresponding tuples. Here we assume that the domain D of the variables contains only elements a and b.

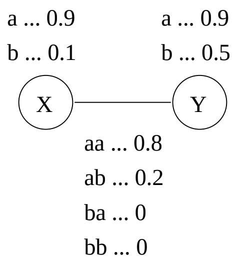


Figure 1: A fuzzy CSP.

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 (here we assume that *con* contains all variables). 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.

Figure 2 shows another example of a fuzzy CSP, its solutions, and its best solutions.

Other useful instances of the semiring-based soft constraint framework are the following ones:

- the so-called probabilistic constraints [8], where semiring values are reals between 0 and 1, combination is achieved via product, and the ordering

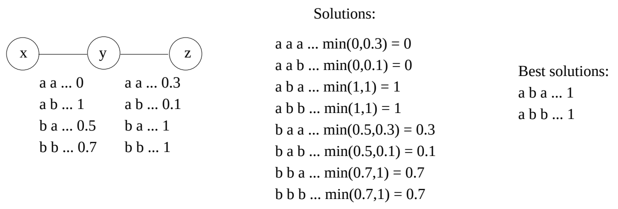


Figure 2: A fuzzy CSP.

is given, as in fuzzy constraints, by max: $A = \{x \mid x \text{ in } [0,1]\}$, $+_s = max$, $\times_s$ = product, $\mathbf{0} = 0$, and $\mathbf{1} = 1$;

- lexicographic constraints [9], where semiring values are tuples of reals in [0,1], combination is pointwise min, and ordering is lexicographic: $A = \{x \mid x \text{ in } [0,1]\}^k$, $+_s$ = lex ordering, $\times_s$ = pointwise min, $\mathbf{0} = 0^k$, and $\mathbf{1} = 1^k$, where k is the number of constraints. This is a way to avoid the so-called drowning effect of fuzzy constraints, where a low preference for one constraint is enough to make the overall solution bad. With lexicographic constraints, two solutions with the same lowest preference, which would be indistinguishable in fuzzy constraints, are now distinguished by the preferences of the other constraints.

- weighted constraints, where semiring values are usually positive reals or naturals, to be interpreted as the cost of a variable instantiation, combination is sum, and the ordering is given by min: $A = \mathbb{R}^+$, $+_s$ = sum, $\times_s$ = min, $\mathbf{0} = +\infty$, and $\mathbf{1} = 0$;

- multi-criteria systems: by taking a c-semiring which is the cartesian product of two or more c-semirings, the framework allows to model multi-criteria scenarios where each of the constituents semirings model one of the optimization criteria [2]. The elements of the multi-criteria semiring are then tuples of elements of the component semirings, and the operations are pointwise applications of the corresponing operations of the component semirings. In such multi-criteria semirings, the resulting ordering is partial even if each of the component semirings is totally ordered.

More complex semirings can also model scenarios where preferences are associated to constraints and not to tuples, as in possibilistic CSPs (where one wants to minimize the maximum preference of violated constraints) or one of the versions of valued CSPs [23].

Usually soft constraint solvers, like the Con'flex solver for fuzzy constraints [6], take as input a set of soft constraints and use a systematic search technique, for example forward checking and branch and bound, to look for the best solution; at each step of the search procedure, they perform some kind of soft constraint propagation (usually full or partial soft arc-consistency). There are also solvers which use local search techniques to find the best solution. In this paper we don't assume any particular search engine, we just assume that the solver knows how to find a best solution given a set of soft constraints.

3 Learning soft constraints

The problem of learning soft constraints can be considered in different scenarios. In a first scenario the user is capable to give precise ratings to solutions presented to him/her by the system. In this case, the learning problem for the system consists in inducing constraint preferences that are consistent with the user's ratings over solutions. In a second scenario, the user is unable to give precise ratings to solutions, however, he/she is able to tell at which degree he/she likes (or dislikes) the solutions, or if he/she is neutral about a specific solution. In this case the learning problem is more difficult since the user is delivering only partial information about his/her desires.

It should be stressed that while in the first scenario the user is perfectly aware of his/her desires, and thus he/she is able to return an absolute ranking value for each solution, in the second scenario, the user is not able to figure out a precise value of ranking to assign to every solution, and thus he/she is only able to produce some non-specific feedback to the system in the form of reward/punishment.

In the first scenario, the learning problem can be cast as a supervised learning problem. Supervised learning [16] 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. Function h is called a hypothesis.

A common approach to supervised 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* [16]. An example of popular error function, that can be used over the reals, is the sum of squares error [16]: $E = \frac{1}{2}\sum_{i=1}^{n}(d(x_i) - h(x_i))^2$, where $(x_i, d(x_i))$ is the i-th example of the training set.

Given a starting hypothesis h_0, the goal of learning is to minimize the error function E by modifying h_0. This can be done by 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 can be formulated in different ways, depending on whether the domain is isomorphic to the reals or not. 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* [16]. 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: $W(\tau + 1) = W(\tau) + \Delta W(\tau)$, where $\Delta W(\tau) = -\eta \frac{\partial E}{\partial W(\tau)}$, and η is the step size used for the gradient descent. Learning is stopped when a minimum of E is reached. Note that, in general, there is no guarantee that the found minimum is global.

Learning can be used to find suitable preferences to be associated to the constraints of a given problem. This problem may be either a hard CSP, or already a soft CSP where the preferences are just a rough estimate of what we want. Here for simplicity we will assume to start from a CSP P, which, we recall, can be seen as an SCSP where the solutions are those n-tuples t such that $val_P(t) = true$.

We assume that for some solutions of P we are given a desired rating $d(t)$ defining the goodness of t. That is, we have the set of *examples* $TR = \{(t_1, d(t_1)), \ldots, (t_m, d(t_m))\}$.

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, and a distance function over such a semiring.

Once we have the above, the learning goal is to define an SCSP P' which has the same semiring and the same topology of P, and 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.

Given the first condition (on the graph topology), the only free parameters that can be arbitrarily chosen in order to satisfy the other condition 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

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

where $\prod$ refers to $\times_s$, k is the number of constraints in P, $\sum$ refers to $+_s$, and subtuple$(t_{ij}, t, i) = \mathbf{1}$ if $t \downarrow^V_{con_i} = t_{ij}$ and $\mathbf{0}$ otherwise. Note that, for each i, there is exactly one j such that subtuple$(t_{ij}, t, i) = \mathbf{1}$. Let l_i be such a j. Thus $\sum_{j=1}^{|S_i|}$ subtuple$(t_{ij}, t, i) \times_s w_{ij} = w_{il_i}$, and therefore $val_{P'}(t) = w_{1l_1} \times_s \ldots \times_s w_{kl_k}$. The values of the free parameters may be obtained via a minimization of the error function, which is defined according to the distance function of the semiring.

Of course, the higher the number of examples, the higher the probability of a successful learning phase. However, it is not feasible to ask the user to provide too many examples. For this reason, in [1] an incremental strategy aims at reducing the number of the examples the user has to provide. Using this strategy, the user just gives an initial small set of examples, over which the system performs the first learning phase. Then, the user checks the resulting system on a set of new solutions, and collects those that are mis-rated by the system, which will be given as new examples for the next learning phase. This process iterates until the user is satisfied with the current state of the system. This approach can also be used in the temporal framework described in [13, 20].

Concerning the universal approximation capability, that is, the ability of the learning algorithm to approximate any preference function, gradient descent is able to learn any preference relation provided that we associate to each tuple a vector of semiring elements, and that we don't put any limit on the length of this vector (see [1, 18]).

Unfortunately, supervised learning cannot be directly applied to the second scenario, where the user is unable to associate a precise desidered value to each presented solution. In this case, a reinforcement learning approach is more appropriate. In reinforcement learning, the system is assumed to interact with an environment (in this case, the user) via a set of actions (in this case, returning a solution in a specific ranking position). The environment is not fully observable and, for each specific action, it returns to the system a feedback which can be neutral or not. If the feedback is not neutral, it can be positive (usually encoded as a positive real value) or negative (usually encoded as a negative real value). The aim of the system is typically to try to maximize the expected (discounted) reward, i.e., the system tries to learn a function (policy) that chooses the actions to be applied in such a way to get (on the average) as much reward as possible (possibly considering future reward/punishment less important than immediate reward/punishment, thus "discounting" future reward/punishment). In the context of this paper, this corresponds to seek for a ranking of the solutions that agrees as much as possible with the user's (hidden) desires. In the following we will discuss a very simple form of learning in this context.

4 The interaction framework

We consider interactive scenarios where a user interacts with a constraint system, which is able to solve the current set of soft constraints. The aim of the interaction is to model a soft constraint problem in a way that matches the users' preferences at best, and then to find its best solutions.

The interaction process can be seen as a sequence of states, each being a set of soft constraints, linked by transitions that allow the user to move from one state to the next one. In each state, the current set of soft constraints is solved by the constraint system and the best solutions (or a subset of them) are presented to the user; on the basis of these solutions, the user decides which transition to activate next.

Allowed transitions. Each transition is the result of applying one or more actions, which can be classified according to the following taxonomy:

- Variable addition: a new variable is added to the current state, plus its domain, in the form of a unary soft constraint.
- Constraint addition: a new soft constraint, which involves a subset of the existing variables, is added to the current state.

- Vertical preference modification: a new preference function is specified for an existing constraint.
- Horizontal preference modification: given a complete variable assignment t, modify the preference value of $t_{\downarrow var(c)}$ for all current constraints c.

Variable and constraint additions allow the user to build a set of constraints from scratch or from an initial state provided by the interactive system, while vertical preference modifications allow for adapting the soft constraints to the users' preferences. For example, by looking at the best solutions of the current state, the user may realize that some soft constraints need to be modified to match his/her desires.

Vertical modifications. In actual interactive systems, vertical modifications may be achieved either by a pointwise specification of a new definition for a soft constraint, or, most probably, by selecting a preference function from a predefined family of functions. Vertical modifications are so called because they involve the preference values of all the tuples of a single constraint, which usually are graphically represented as a vertical list of tuples.

Notice that vertical modifications can be used also to cancel an existing constraint, by setting its preference function to always return the best level of the semiring. It is also possible to simulate a variable removal, by cancelling all the constraints involving it.

Horizontal modifications. Horizontal preference modifications, on the other hand, provide a way to modify the preference value of a single tuple in each constraint. Such tuples are not chosen randomly, but are linked by the fact of being part of one complete assignment of the problem. This modifications allow the user to give his/her feedback over current solutions: in fact, such a feedback can be used to generate an appropriate horizontal preference modification action, which will involve the tuples which are part of the solution. In general, the new preference value for each tuple involved in the modification will be a function of the old preference value, and of the user's feedback. In the following, we will see that such a function can be computed by using machine learning techniques.

Conflicting actions/transitions. As mentioned before, each transition may consist of several actions. However, since such actions may be conflicting, and since they will be applied in parallel to the current state, it is necessary to either avoid conflicting actions in the same transition, or to define an appropriate conflict-resolution policy. In general, two actions are in conflict if they want to modify the preference value of the same tuple in two different ways. For example, it is easy to see that vertical modifications over different constraints are never conflicting. On the contrary, horizontal modifications may be in conflict, if they refer to complete assignments which select the same tuple in some constraint. Using a conflict-resolution policy means choosing a function which combines the conflicting actions and generates a new modification which may be more complex than those listed above.

Since transitions are sets of actions, a conflict may arise also among different transitions. This implies that our transition system is in general not commutative. In fact, performing the same conflicting transitions in different orders may lead to different final states. While we envision a conflict-resolution policy for actions, in our approach we do not consider any way to resolve conflicts among transitions, since the application of this policy would require the history of the interaction, and this could be very costly.

We will now show how this interaction framework can be instantiated to describe different specific interaction models.

The simplest model: only vertical modifications. Most current interaction models allow only for variable and constraint additions, and vertical modifications (see for example [11]). The interaction is then built around the following steps:

1. the user adds soft constraints and/or variables;
2. the solver returns some solutions (probably the best ones);
3. if the user is satisfied with some of the solutions proposed by the solver, the interaction stops; otherwise the user can perform some vertical modifications and go back to step 1 or 2.

Figure 3 shows this interaction model. In this figure, we call local preferences the preferences over constraints.

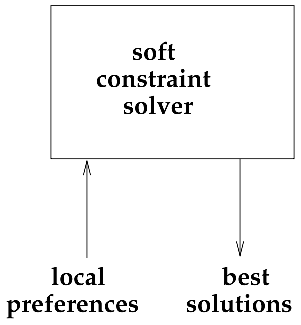


Figure 3: The basic interaction model: only vertical modifications.

A more complex model: also horizontal modifications. However, sometimes it is not easy to model all our knowledge about a problem in terms of soft constraints, that is, preferences over all the assignments of a certain subset of variables. It may instead be easier to express some of our knowledge as preferences over solutions. In this respect, horizontal modification actions come to help, since, as noted above, they provide a way to use the user's feedback over solutions to suitably modify the current state. Thus the interaction model is now as follows:

1. the user adds constraints and/or variables;
2. the system returns some best solutions;
3. if the user is satisfied with some of the solutions proposed by the solver, the interaction stops; otherwise the user can
    (a) perform some vertical modifications and go back to step 1 or 2;
    (b) perform some horizontal modifications and go back to step 1 or 2.

Notice that, at each step, the user can perform either vertical or horizontal modifications, but not both. This can be seen as a very simple way to provide a conflict-resolution policy among actions, since we know that vertical modifications are never in conflict. However, there is still the need for a conflict-resolution policy for conflicting horizontal modifications.

To perform the transitions which consist of horizontal modifications, we propose to use a learning module, in a way similar to that described in the learning section, which takes in input the solutions provided by the system in the current state, and the user's preference value over these solutions, and learns the new preference values for the appropriate tuples in the constraints.

From the system point of view, which is now composed of the soft constraint solver and the learning module, the behaviour can be specified as follows:

1. vertical modifications provide the soft constraint solver with new soft constraints;
2. the user's feedback over current solutions is given to the learning module, which will learn appropriate horizontal modifications to be passed to the soft constraint solver;
3. in both cases, the soft constraint solver will take the new soft constraints and return the best solutions of the new state.

Figure 4 shows the architecture and information flow of this interaction model. In this figure, we call global preferences the preferences over complete solutions.

It is obvious that, in this enhanced interaction model, the possibility of asking also for horizontal modifications provides a way to give preferences over solutions, rather than over constraints. This yields a more accurate acquisition of the user's knowledge of the problem to be solved, thus achieving a better model of the problem. Besides accurateness, also a better precision is achieved. In fact, the vertical modifications can represent just rough estimates of what the user has in mind, while the horizontal modifications, supported by the learning module, can be used to make such an estimate more precise at each interaction phase.

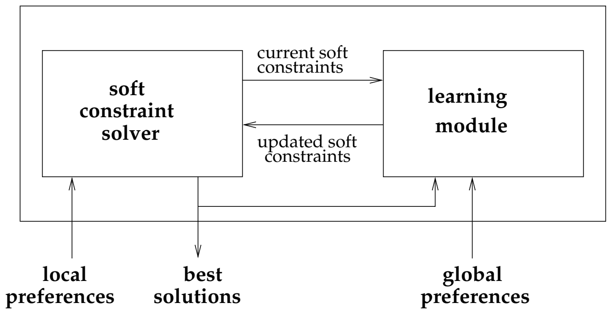


Figure 4: Both vertical and horizontal modifications: solver and learning module.

Notice that the output of the system (that is, the best solutions found so far) is used to improve the current soft constraints, thus providing a feedback mechanism which is used, via the learning module, to improve the model of the problem. It is also important to note that this mechanism can be viewed also as a way to perform on-line within-session personalization of the service to the user, in contrast to other interaction systems where learning or other techniques can be used to perform off-line and between-sessions user profiling, in order to propose more personalized services in the next interaction session. Here we can do this, but we can also personalize the service during a single session.

Since we cannot expect a user to state his opinion on many proposed solutions, we can assume that the learning module receives only a few examples at a time. This is in line with the incremental learning procedure described in [18] and [1], and allows for a more user-friendly interaction. This incremental use of the learning module is in general less efficient than the classical batch mode where all examples are given at once. However, it must be noticed that, in our interaction framework, the soft constraint system has already some knowledge from the user, coded in terms of the current soft constraints. Thus the learning module is not used to learn the whole amount of knowledge to be coded as soft constraints, but it just has to refine the current preferences. This suggests that much fewer examples should be needed to encode the knowledge expressed as preferences over solutions in terms of new soft constraints in such a way that the error is small.

Notice that, although not explicitely defined above, also the solver module can be used in an incremental mode. That is, the user can provide the solver with some constraints and preferences at a time, and interleave the input of preferences over solutions and of new soft constraints. In this way, the system can receive more and more information as the interaction goes on, both in terms of preferences over solutions, and also of preferences over old and new constraints.

One could ask himself if the proposed interaction framework always leads to the user-desired solution. Convergence is assured if the user knows what he wants, and it is derived by the completeness of the soft constraint solver: if a user knows which solution he wants, he can put all its constraints with highest preference, and the system will return (in one step) such a solution if it exists. If it does not exist, then the system will return the closest solution, according to the optimization criteria chosen. However, in the scenario we have in mind, the user does not know exactly what he wants, so the task of of deciding the convergence or the complexity of the system is not well-defined. In the worst case, that is, that the user is completely confused and changes his mind many times, the system could go on forever without satisfying him.

5 A working system

We have developed an interactive system which instantiates the above described interaction framework by making several specific choices. Notice that such choices were made just for simplicity reasons, and not because one cannot have the most general scenario:

- The application field chosen for our system is the search for a house in the catalog of a real estate agency. This application domain is very simple, because houses are monolithic products, that are not described by a set of parts with compatibility constraints among them. Therefore the catalog can be seen as one n-ary hard constraint (where n is the number of the features of a house), and the solver only needs to solve the conjunction of this hard constraint plus the soft constraints representing the user requirements. Other application domains are much more complex. For example, cars are usually described by a collection of single features (like the engine, the type, the color) that can be combined in different ways, and with possibly some preferences on certain combinations over others.

- Fuzzy constraints are used to model the user requirements. In general, one can use any soft constraint class.

- Only unary soft constraints are allowed as user requirements. In general, one can post constraints of any arity.

- No variable or constraint addition is allowed. This means that the set of variables and constraints is chosen at the beginning of the interaction and cannot be changed.

- The learning module uses a very simple learning algorithm, which just reinforces the preferences of solutions with a feedback, inducing and improvement if the feedback is positive and a decrement if it is negative. In general, the learning algorithm can follow the general lines outlined in Section 3.

The system consists of a Java layer, which takes care of the input and output interface, on top of the Con'flex [6] fuzzy constraint solver.

The user can ask for both vertical and horizontal modifications by inserting appropriate data into certain regions of the interface window. In particular, he can specify a vertical modification for a unary constraint in two different ways: either by selecting a preference function out of a certain class, or pointwise.

If the constraint is over variables with numerical values, like the price of a house, the user does not need to define the new preference function on each point, but the system provides a family of preference functions which have two parameters x and y: x states the allowed range for the variable (like "price < x"), and y is interpreted as a level of importance for the whole constraint. This level is then used by the Java layer to generate appropriate preferences for each of the tuples (singletons in this case) which participate in the constraint. More precisely, if the tuple fully satisfies the constraint (that is, it belongs to the specified range), it is given value 1, independently of the level of importance given by the user. If instead a tuple does not satisfy the constraint, then it is given a preference value which depends on the level of importance set by the user, starting from the "dual" of the level from the tuple which is closer to satisfying the constraints, and decreasing the preference as we get far from the allowed range. For example, if we say that the price has to be up to $200,000, with an importance level of 0.8 (actually the user does not type 0.8, but just moves a slider), then all prices up to $200,000 are given preference value 1. Then, assuming that price steps are $50,000, we give preference level 0.2 to $250,000, and lower preference levels for higher prices. Different decreasing functions could be used to model this; we have chosen a function based on an exponential because its behaviour looked better for the domains we have.

If the domain of a variable is not numeric but symbolic (like the type of a house), the user can explicitly set a preference level for each domain element. In fact, in this case it is not possible to derive the new preference function from just a few parameters.

In both cases, the Java layer, after computing the appropriate preferences, generates the updated set of fuzzy constraints and passes them to the Con'flex solver. The solver generates the best solutions, which are then taken again by the Java layer to display them in a new window. In this window, the best n (in our case, $n = 4$) solutions are displayed, with all their features, and an associated satisfaction level, which represents the preference value for each solution (in our case of fuzzy constraints, the minimum of all preferences in all the constraints). The user can either accept one of them, closing the interaction with the system, or he can give his opinion about all or some of the solutions (by using appropriate sliders). This opinion is then taken by the Java layer and handled by the learning module (again written in Java) which generates the appropriate horizontal modifications, that is, new preferences for some tuples in some constraints. The updated set of fuzzy constraints is then passed again to the Con'flex solver, which again returns the best solutions.

More precisely, the user gives preference level p to a certain solution, where p is a semiring value: in our system the interface presents four grades, which correspond to four semiring values: 0.25, 0.5, 0.75, and 1. This value is used to generate a horizontal modification as follows: if $p \leq 0.5$, the current preference value of all tuples selected by the solution is decreased by a constant which is proportional to p (higher for smaller values). Otherwise, the preference values are increased, in a similar way.

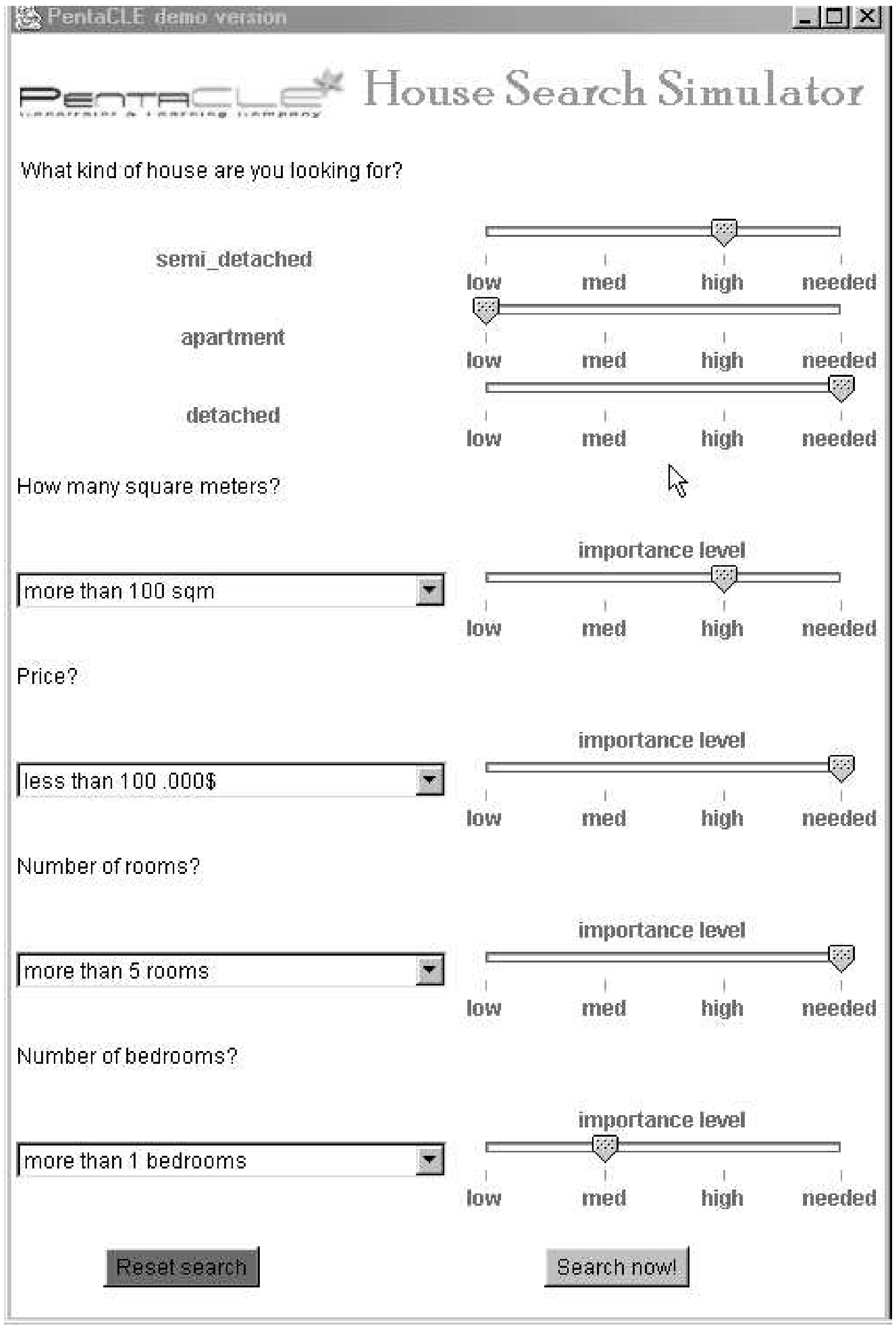
While the handling of vertical modifications does not create conflicts, as noted above, the horizontal modifications asked for in the same step may be in conflict. In our system, the conflict is solved by combining several horizontal modifications via the algebraic sum of the constants associated to each of them. This allows to discover irrelevant features. For example, if two houses have the same price, but they receive a very different preferences from the user, then it means that the price feature was irrelevant for the user's opinion. Therefore its preference value should not be modified in the fuzzy constraints. In our system, the two horizontal modifications generate two constants with the same value but opposite sign, which will result in a null perturbation when the modifications are combined.

The kind of reinforcement learning we use in our working system in general does not guarantee to be able to learn any preference function. This is due to the choices we made for the learning setting, like giving reward or punishments to all constraint preferences whenever a solution feedback is processed. In general, however, it could be possible to define other learning settings (for example, by allowing users to give feedback over a single variable value in a solution) which give the possibility of learning any preference function. However, such settings could be too demanding for the users.

6 Example

Let us go through an example of the interaction between a user and our system. First, the user sets his preferences in a graphical window, as in Figure 5. In this figure, the first constraint is a symbolic one, meaning that it is a unary constraint over a variable (in this case, house-type) whose domain is a set of symbols (in this case: semi-detached, apartment, and detached). On this constraint, preferences can be set directly over each of the variable values, by using the sliders which provide four different levels: low, medium, high, and needed. Each level corresponds to a specific value between 0 and 1 (we recall that we are using fuzzy constraints here). More precisely: low=0.25, medium=0.5, high=0.75, and needed=1.

The next constraints are numeric ones, meaning that they are unary constraints over variables whose domain is an ordered set of integers. On these constraints, preferences can be put over the entire constraint, with the slider method, and they will be automatically induced over all the tuples of variable values. More precisely, if we set, as in Figure 5, that we want more than 100 square meters, with a high preference (that is, 0.75), then all the values of the variable square-meters which are above 100 will have preference 1, since the constraint is satisfied, while the others will have preference starting at 0.25 (that is, the dual of 0.75 with respect to 1) and going down as we move away from 100, with the law described by an exponential as mentioned in the previous section.



PentaCLE demo version

House Search Simulator

What kind of house are you looking for?

semi_detached: high

apartment: low

detached: needed

How many square meters?

more than 100 sqm — importance level: high

Price?

less than 100 .000$ — importance level: needed

Number of rooms?

more than 5 rooms — importance level: needed

Number of bedrooms?

more than 1 bedrooms — importance level: med

Figure 5: Initial preference setting.

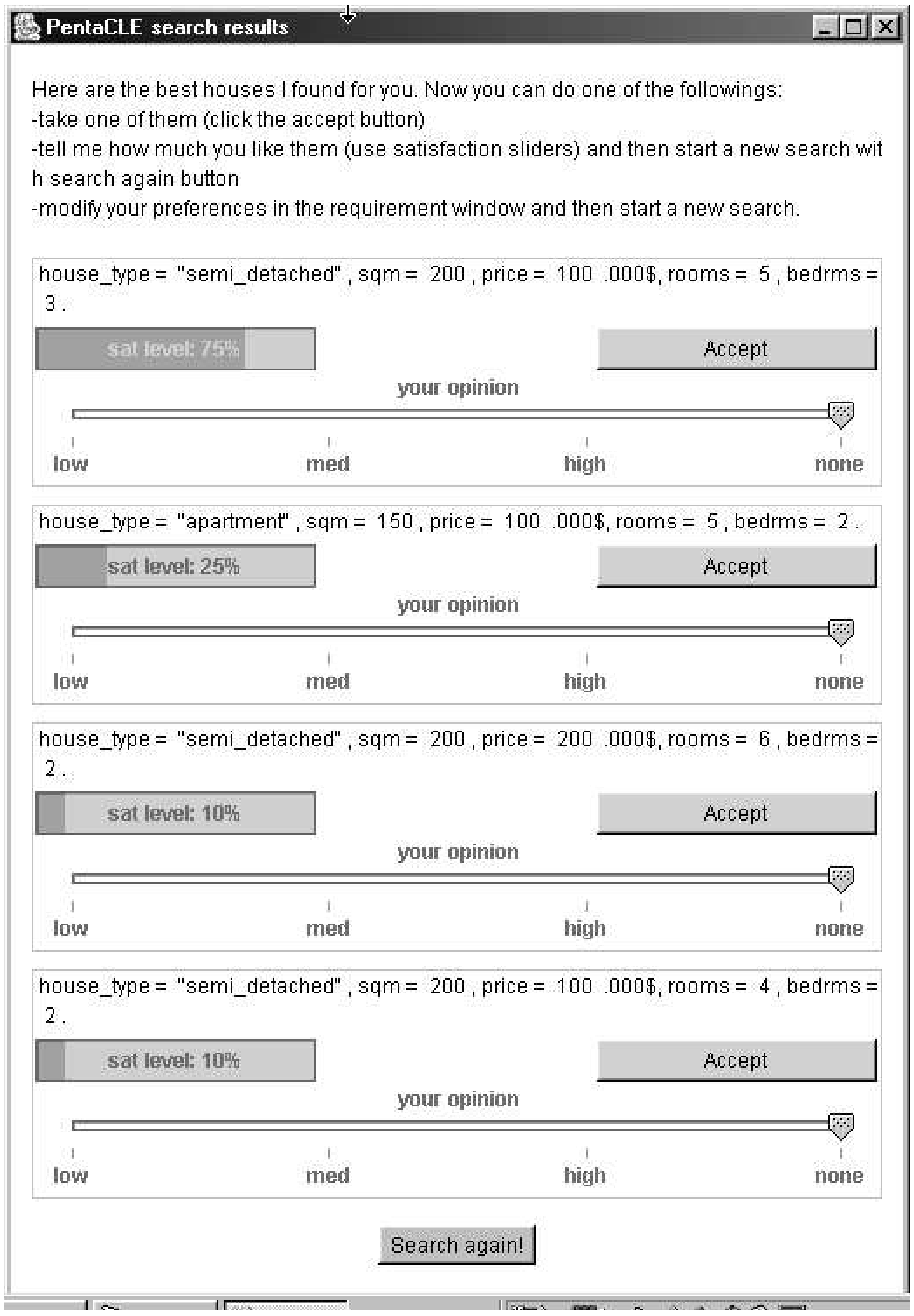
By clicking on the search button, the user asks for some solutions to the constraints and preferences he has posed. For our example, the result of clicking over this button can bee seen in Figure 6.

Figure 6 shows the best four solutions according to the given preferences. We recall that the preference value of each solution is the minimum preference over all the constraints. The best solution in Figure 6 has preference value 0.75 (written as "sat level 75%" in the result window). This means that no house in the catalog satisfies all constraints at the best level (which is 1). This solution has level 0.75 because of the constraint on the type of the house, which is satisfied with level 0.75 (which was our preference level for value "semi-detached"). All the other constraints are satisfied at level 1. The second house has preference level 0.25 because of the same constraint: in fact, we had set to 0.25 the preference over apartments. The third house has level 0.1 because its price is 200K: we recall that we had set 100K as the price limit, and 1 as its preference. This puts to 0.1 the preference level of the non-satisfying values. Note that we cannot put 0 because otherwise the minimum would be 0 and so there would be no solution, which we want to avoid. The fourth house has preference level 0.1 since it has 4 rooms while we had set the room threshold to 5 and the preference of this constraint to 1.

Notice that, at this point of the interaction, both windows (the one in Figure 5 and the one in Figure 6) are visible on the screen. Now the user can decide to either accept one of the proposed houses, or to change the preferences over the constraints, or to give feedback over the solutions. This last choice can be made by using the sliders below each of the proposed solutions. Let's continue the example assuming this last alternative. The user sets the opinions over the solutions as can be seen in Figure 7.

We recall that a positive feedback (medium, or high) induces an increase over the values of the variables, while a negative feedback (low) induces a decrement. For example, for the first solution, giving a high opinion means that the preference values of all variables values which constuitute this solution (that is, house-type=semi-detached, square-meters=200, price=100K, rooms=5, bedrooms=3) are increased by a certain quantity (0.2 in our system). The same for all those solutions whose opinion is not "none".

Notice that opinions over different solutions can compensate each other, and thus achieve a conflict-resolution mechanism. For example, the preference for price value 100K is increased because of the opinion on the first and fourth solution, but it is also decreased because of the opinion on the second solution. This means that in some cases the modifications can result in no change. This is the reason why the preference value of the first solution remains unchanged, as can be seen in Figure 8, which is obtained by clicking on the "search again" button after having set the opinions. In fact, all the features of the first solution are increased, except for the room feature, whose preference value is increased because of the feedback over the first solution, and decreased of the same quantity because of the feedback over the second solution. Thus the minimum preference remains the same, that is, 0.75.


PentaCLE search results

Here are the best houses I found for you. Now you can do one of the followings:
-take one of them (click the accept button)
-tell me how much you like them (use satisfaction sliders) and then start a new search wit
h search again button
-modify your preferences in the requirement window and then start a new search.

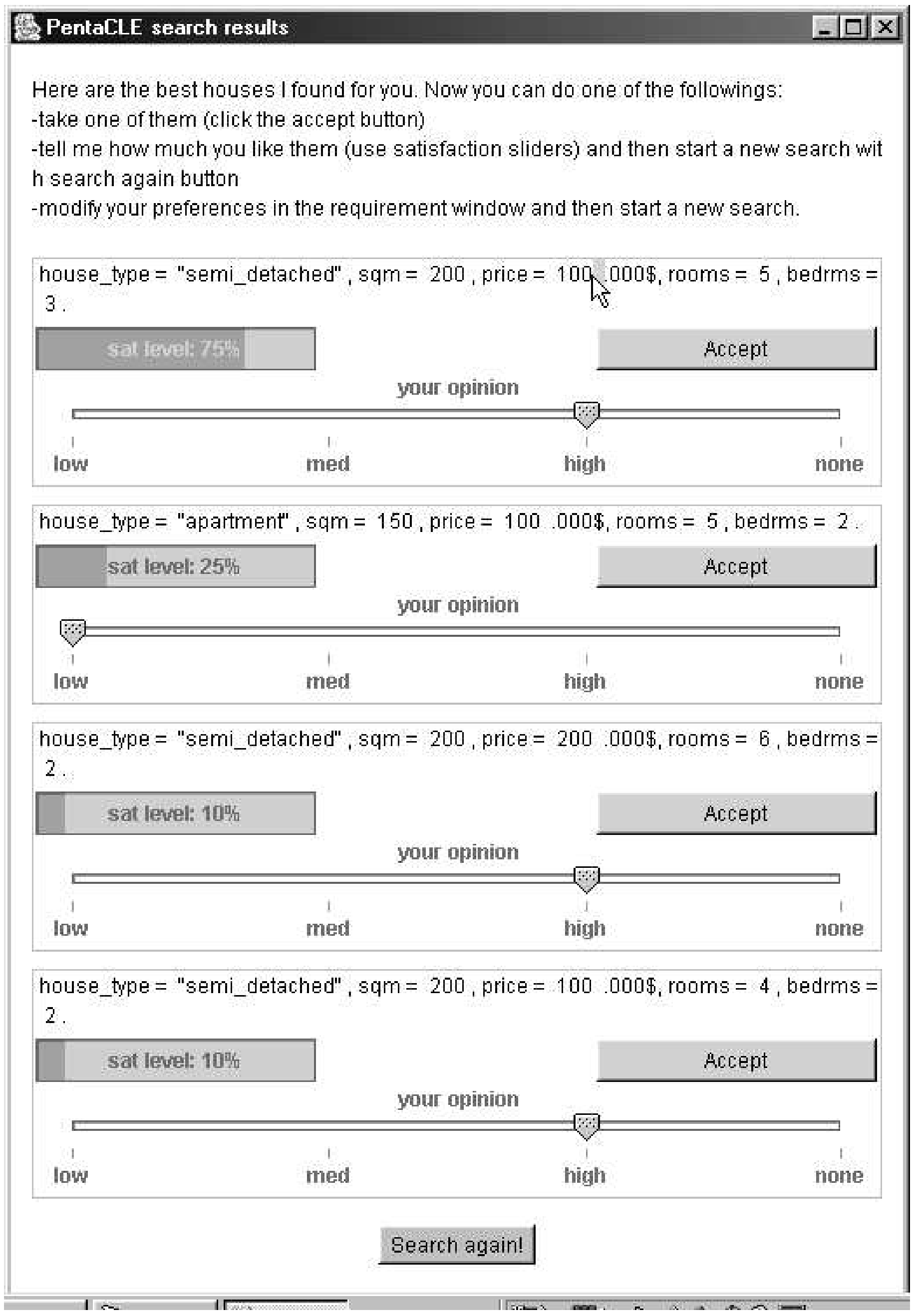
house_type = "semi_detached" , sqm = 200 , price = 100 .000$, rooms = 5 , bedrms = 3 .
sat level: 75%

house_type = "apartment" , sqm = 150 , price = 100 .000$, rooms = 5 , bedrms = 2 .
sat level: 25%

house_type = "semi_detached" , sqm = 200 , price = 200 .000$, rooms = 6 , bedrms = 2 .
sat level: 10%

house_type = "semi_detached" , sqm = 200 , price = 100 .000$, rooms = 4 , bedrms = 2 .
sat level: 10%

Figure 6: The best 4 solutions.


PentaCLE search results

Here are the best houses I found for you. Now you can do one of the followings:
-take one of them (click the accept button)
-tell me how much you like them (use satisfaction sliders) and then start a new search wit h search again button
-modify your preferences in the requirement window and then start a new search.

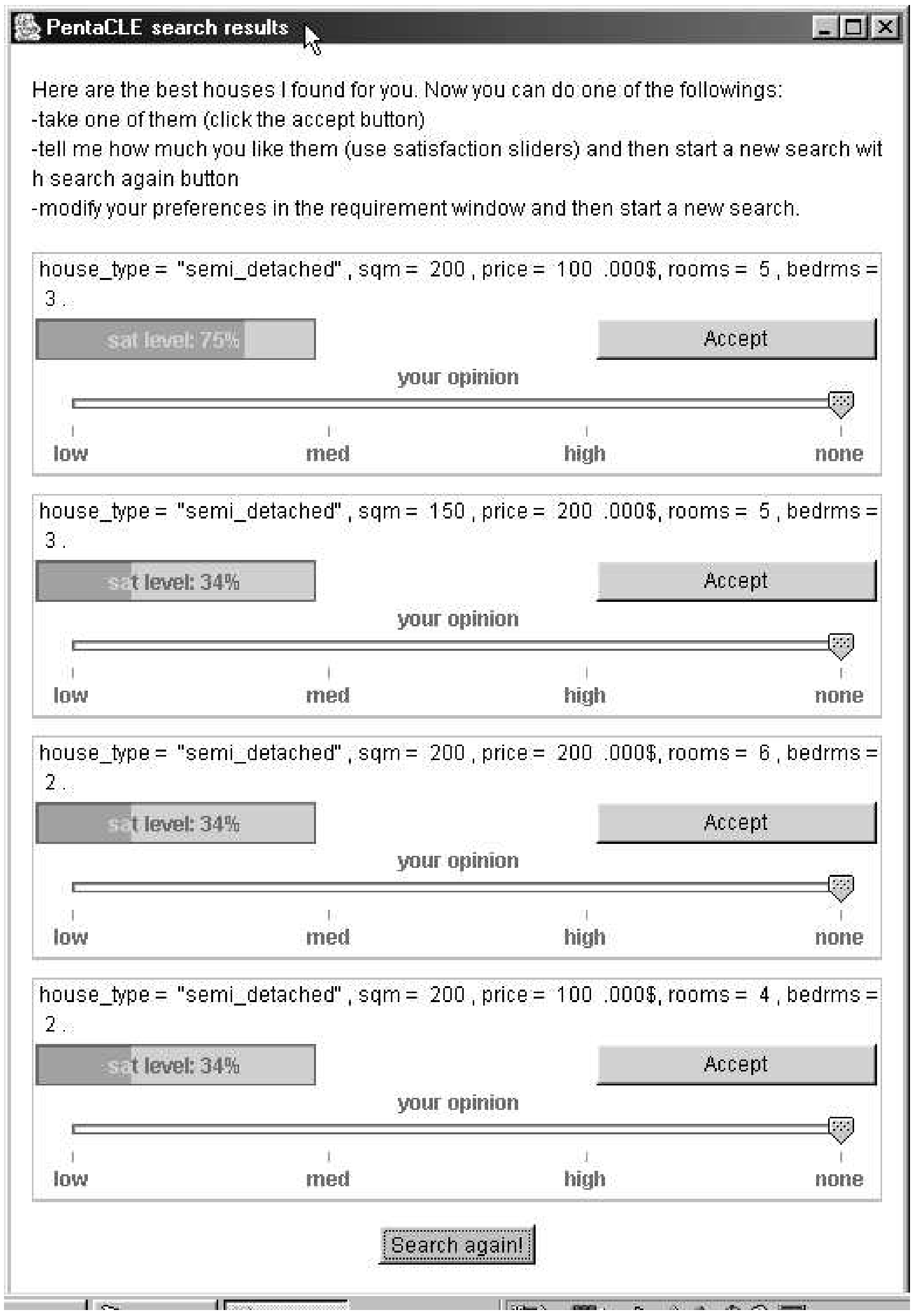
house_type = "semi_detached" , sqm = 200 , price = 100 .000$, rooms = 5 , bedrms = 3 .

house_type = "apartment" , sqm = 150 , price = 100 .000$, rooms = 5 , bedrms = 2 .

house_type = "semi_detached" , sqm = 200 , price = 200 .000$, rooms = 6 , bedrms = 2 .

house_type = "semi_detached" , sqm = 200 , price = 100 .000$, rooms = 4 , bedrms = 2 .

Figure 7: Feedback over the solutions.



Here are the best houses I found for you. Now you can do one of the followings:
-take one of them (click the accept button)
-tell me how much you like them (use satisfaction sliders) and then start a new search wit h search again button
-modify your preferences in the requirement window and then start a new search.

house_type = "semi_detached" , sqm = 200 , price = 100 .000$, rooms = 5 , bedrms = 3 .

house_type = "semi_detached" , sqm = 150 , price = 200 .000$, rooms = 5 , bedrms = 3 .

house_type = "semi_detached" , sqm = 200 , price = 200 .000$, rooms = 6 , bedrms = 2 .

house_type = "semi_detached" , sqm = 200 , price = 100 .000$, rooms = 4 , bedrms = 2 .

Figure 8: New best solutions.

7 Conclusions and Future work

We have described an interaction framework, based on a soft constraint solver, where users can post preferences both over constraints and over solutions, and we have proposed a way of building working instances of such a framework via the association of a standard soft constraint solver and a learning module. The acquisition of knowledge is incremental and more complete and precise than with standard soft constraint solvers. The learning module provides also a feedback and an on-line profiling mechanism.

A specific instance of the proposed framework is defined by choosing a semiring for the soft constraint solving part, and a learning setting for the learning module. The fact that an instance is able to converge to and to learn any preference function depends on the choices made: the framework is general enough to have both properties. For example, the kind of reinforcement learning we use in our working system in general does not guarantee to be able to learn any preference function. It is possible to define other learning settings which give the possibility of learning any preference function. However, such settings could be too demanding for the users. Therefore, as expected, one has to consider a trade-off between the expressivity of the model, its efficiency, and the amount of information that has to be provided by the user.

Semiring-based CSPs can model partial preference relations. If one chooses a semiring which is partially ordered, solutions will be ranked on such a partially ordered set. Working with partially ordered semiring will not change the properties of the framework: there are instances of the framework which are able to learn any preference functions (whether total or partial) and others which are not.

The working system we describe in this paper is just one of the many possible instances of the general framework we propose. In this instance, a catalog of houses is described via an n-ary constraint and many unary soft constraints, so search is limited. However, in other catalog scenarios there many be products consisting of many parts and such parts could be combined together subject to several hard and/or soft constraints. Thus in those cases we need a separate constraint system which is able to search intelligently the solution space.

We are planning to extend our working system to deal also with more complex domains, other soft constraint classes, and non-unary constraints as user requirements. We also plan to propose not just the best n solutions, but, in case of ties, to propose solutions which are reasonably different from each other, in order to maximize the amount of information that the user can give us with his opinion over the solutions.

We also plan to assess the usefulness of our approach with real users. In particular, we will study the behaviour of our working system on some classes of problems, to check whether it behaves better (in terms of number of iterations before an acceptable solution is found) than standard soft constraint solvers.

Other interactive iterative constraint systems have been proposed, such as the matchmaking system in [11]. Although the learning part is not present in that system, and it deals with hard constraints, the authors made a careful study of the behaviour of the system in the presence of different strategies, which follow different optimization criteria (minimize the number of iterations, or maximize the number of discovered user constraints, ...). We plan to perform a similar analysis in our context.

Our system for catalog search proposes to use both soft constraint technology and machine learning, in the spirit of the interaction framework we have presented in this paper, to allow for an improved interaction beetween a user and a catalog search system. Other techniques have been used to search in catalogs, such as recommender systems [15], collaborative filtering [24], and case-based reasoning [3]. Also, [14] proposes to use constraint technology together with abstraction, clustering, and semplification of constraints. All these techniques can be used to extend our approach in order to make the interaction more successful.

Acknowledgments

We would like to thank the company Aiglos for supporting the development of the first version of our system, the PentaCLE group for continuing to develop and improve the implementation, and the many people who gave us valuable suggestions on how to improve both our system and our interactive framework. This research was partially supported by Italian MIUR project 2003091149_005 and by ASI project ARISCOM.

References

[1] A. Biso, F. Rossi, and A. Sperduti. Experimental Results on Learning Soft Constraints. *Proc. KR 2000*, Morgan Kaufmann, 2000.

[2] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based Constraint Solving and Optimization. *Journal of the ACM*, 44(2):201–236, March 1997.

[3] R. Burke. The Wasabi personal shopper: a case based recommender system. Proc. Conf. on innovative applications in AI.

[4] P. Codognet and F. Rossi, eds. Special issue of the Constraints journal on soft constraints, vol. 8, n. 1, Kluwer, 2003.

[5] Remi Coletta, Christian Bessiere, Barry O'Sullivan, Eugene C. Freuder, Sarah O'Connell, Jol Quinqueton. Semi-automatic Modeling by Constraint Acquisition. Proc. CP 2003 (short paper), Springer Verlag LNCS 2833, 2003.

[6] Con'flex web site. http://www.inra.fr/bia/T/conflex/

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

[8] H. Fargier, J. Lang. Uncertainty in Constraint Satisfaction Problems: a Probabilistic Approach. Proc. European Conference on Symbolic and Qualitative Approaches to Reasoning and Uncertainty (ECSQARU), Springer-Verlag, LNCS 747, pp.97–104, 1993.

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

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

[11] E. Freuder and R. Wallace. Suggestion strategies for constraint-based matchmaker agents. Proc. CP 1998, Springer Verlag, LNCS 1520, 1998.

[12] L. Khatib, P. Morris, R. Morris, F. Rossi. Temporal Constraint Reasoning With Preferences. *Proc. IJCAI 2001.*

[13] L. Khatib, P. Morris, R. Morris, F. Rossi, A. Sperduti. Learning Preferences on Temporal Constraints: A Preliminary Report. Proc. TIME 2001, IEEE Computer Society Press, June 2001.

[14] F. Laburthe, Y. Caseau. Using constraints for exploring catalogs. Proc. CP 2003, Springer-Verlag, LNCS 2833, 2003.

[15] P. Maes, R. H. Guttman, A. G. Moukas. Agents that buy and sell. Communication of the ACM, 42(3), 1999.

[16] T. Mitchell. Machine learning. McGraw Hill, 1998.

[17] Y. Qu, S. Beale. A constraint-based model for cooperative response generation in information dialogues. Proc. AAAI'99, pp. 148-155, Morgan Kauffmann, 1999.

[18] F. Rossi and A. Sperduti. Learning solution preferences in constraint problems. *Journal of Experimental and Theoretical Artificial Intelligence*, 1998. Vol 10.

[19] F. Rossi and A. Sperduti. Aquiring both global and local preferences in interactive constraint solving via machine learning techniques. Proc. CP 2001 workshop on User-Interaction in Constraint Satisfaction. Cyprus, December 2001.

[20] F. Rossi, A. Sperduti, K. B. Venable, L. Khatib, P. Morris, R. Morris. Learning and Solving Soft Temporal Constraints: An Experimental Study. Proc. CP 2002, Springer Verlag, LNCS, 249-263, 2002.

[21] Zs. Ruttkay. Fuzzy Constraint Satisfaction. Proc. 3rd IEEE International Conference on Fuzzy Systems, 1994.

[22] T. Schiex. Possibilistic constraint satisfaction problems, or “how to handle soft constraints?”. In *Proc. 8th Conf. of Uncertainty in AI*, pages 269–275, 1992.

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

[24] U. Shardanand, P. Maes. Social information filtering algorithms for automating ”word of mouth”. Proc. Human factors in computer science, ACM, 1995.

[25] Rina Dechter. Constraint processing. Morgan Kauffmann, 2003.