Aquiring both global and local preferences in interactive constraint solving via machine learning techniques

F. Rossi
Dept. of Pure and Applied Mathematics
University of Padova, Italy
Email: frossi@math.unipd.it and A. Sperduti
Dept. of Computer Science
University of Pisa, Italy
Email:perso@di.unipi.it

No Institute Given

Abstract. Soft constraint solvers 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 can only take as input local preferences, that is, preferences over either constraints, or variables, or tuples of domain values. On the other hand, it is sometimes difficult to code all the knowledge a user has about a problem in terms of local preferences, but it could be easy to do that in terms of global preferences, that is, preferences over entire solutions of the problem.
In general, we think that a constraint system should be able to acquire knowledge both in terms of local preferences and also as global ones, to have a more complete account of the entire knowledge of the user. In this paper, we propose a way to build such a system by pairing a soft constraint solver and a learning module, which learns local preferences from global ones.
We then define an interactive model where a user and a constraint system iterate through information exchange: at each step, the user can provide the system with more constraints, local preferences, or global preferences, and the system returns the best solutions give the current set of soft constraints. Global preferences are allowed only on proposed solutions, to avoid flooding the user with irrelevant information.
We also describe an existing interactive system which uses a fuzzy constraint solver and a suitable learning module to search a catalog for the best product that matches the user requirements.

1 Motivation and main idea

Many real-life search problems are naturally handled by using soft constraints [2, 12, 4]. For example, when searching a catalog for the best product that matches some user requirements, using hard constraints to express such requirements 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 local 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 local 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 local preferences, but doesn't want to spend too much time in the modelling phase and prefers to just give some examples of solution ratings.

In general, we claim that preferences are certainly very useful and expressive, but that up to now constraint systems allowed only for local preferences (that is, preferences over constraint or tuples), while they should also allow for the specification of global preferences (that is, preferences over global 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 local preferences, or by stating some global 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.

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 local and global 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 local preferences to match such global ones. In this way, standard constraint solvers can be used, and the only piece that needs to be built is the learning module.

In [9] we have already studied the use of machine learning techniques base on gradient descent to infer local preferences from global ones, and in [1] we have shown experimental results that exhibit small errors and need a reasonable number of examples. We have also proposed, besides the usual batch operating mode here 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 cite [7] we have applied the learning idea 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.

We also give a description of a system which fits the framework we propose 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 house that best matches a user's preferences.

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, and Section 6 summarizes the results of the paper and hints at possible lines for future work.

2 Soft constraints

Standard finite domain constraint satisfaction problems (CSPs) [13] 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].

Semirings. A *semiring* is a tuple $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ such that

- A is a set and $\mathbf{0}, \mathbf{1} \in A$;

– $+$, the additive operation, is commutative, associative and $\mathbf{0}$ is its unit element;
– $\times$, the multiplicative operation, is associative, distributes over $+$, $\mathbf{1}$ is its unit element and $\mathbf{0}$ is its absorbing element.

A *c-semiring* is a semiring in which $+$ is idempotent (i.e., $a + a = a, a \in A$), $\mathbf{1}$ is its absorbing element, and $\times$ 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 $a + b = b$. 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:

– $+$ and $\times$ 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 $a, b \in A$, $a + b = lub(a, b)$ (where *lub*=least upper bound);
  - if $\times$ is idempotent, then $\langle A, \leq_S \rangle$ is a complete distributive lattice and $\times$ is its greatest lower bound (*glb*).

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

From local to global preferences. 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 $\times$. Then, in order to choose the best among the solutions, the ordering induced by the other operation ($+$) is used. More precisely, we have the operations of *combination* ($\otimes$) and *projection* ($\Downarrow$) over constraints, which are used, respectively, to combine constraints and to project out some variables[1].

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

[1] For the formal definition of these two operations, see [2].

Instances. Many constraint formalisms can be cast in this semiring-based framework: it is enough to choose the appropriate semiring. Classical CSPs are just SCSPs over the c-semiring $\langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$, where $A = \{true, false\}$, $+ = \vee$, $\times = \wedge$, $\mathbf{0} = false$, 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) [4, 10, 11], 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: the c-semiring to use here has $A = \{x \mid x \text{ in } [0, 1]\}$, $+ = max$, $\times = min$, $\mathbf{0} = 0$, and $\mathbf{1} = 1$.

Soft constraint solvers. Usually soft constraint solvers, like the Con'flex solver for fuzzy constraints [3], use a systematic search technique, for example forward checking and branch and bound, to look for the best solution, and 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 for the given set of soft constraints.

3 Learning soft constraints

As noted in the introduction, it is not always easy to specify the preference functions in each temporal constraint in a way that the real-life problem at hand is faithfully modelled. This happens because sometimes it is easier, or possible, to specify only global preference functions, to be associated to entire solutions, rather than local preference functions to be attached to the constraints. For this reason, and since the whole soft constraint machinery is based on local preferences, we propose here a method to induce local preferences from global ones.

Inductive learning. More precisely, the problem to be considered here can now be formally described as an inductive learning problem [8]. 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. 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* [8]. An example of popular error function, that can be used over the reals, is the sum of squares error [8]: $E = \frac{1}{2} \sum_{i=1}^{n} (d(x_i) - h(x_i))^2$, 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* [8]. 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 soft constraints. Learning in our context 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 local 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 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* $TR = \{(t_1, d(t_1)), \ldots, (t_m, d(t_m))\}$.

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, our goal is to define a corresponding SCSP P' over the same semiring such that P and P' have the same graph topology, 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

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

where $\prod$ refers to $\times$, k is the number of constraints in P, $\sum$ refers to $+$, 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$(t_{ij}, t, i) \times w_{ij} = w_{il_i}$, and therefore $val_{P'}(t) = w_{1l_1} \times \ldots \times w_{kl_k}$. 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.

Examples: min-sum and max-min. Assume that the desired ratings are real numbers and that the chosen c-semiring is $\langle \Re^+, min, +, +\infty, 0\rangle$. Soft constraints based on this semiring can be used to represent classical optimization problems. Here $\times$, that is, standard sum, is continuous and derivable. Thus we can use the gradient descent technique. Also, as a distance function it is possible to use here 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 (1) becomes

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

and the error function is

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

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.

Another example concerns fuzzy constraints, which use a max-min framework. In this case, the multiplicative operation of the semiring, that is, min, is not continuous and derivable, thus it seems that we cannot apply the proposed learning scheme in this case. However, it is possible to use an approximation of the min operation which is continuous and derivable. This is what we did to learn temporal constraints in [7].

Incremental training set. 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] we proposed an incremental strategy which 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 [7].

4 The overall interaction model

We consider scenarios where we want to use soft constraints to model preferences. Therefore we have a soft constraint solver which takes as input a set of constraints with associated local preferences (over constraints and/or over tuples) and returns solutions.

The simplest interaction model for a soft constraint solver could then consist of the following steps:

1. the user posts constraints and local preferences;
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 can stop; otherwise the user can change its local preferences and thus go back to step 1.

Figure 1 shows this simple interaction model.

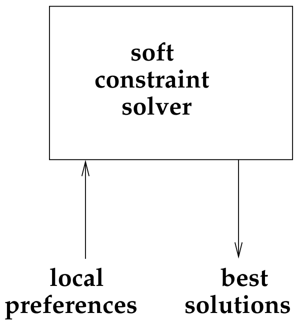

Fig. 1. The simplest interaction model.

However, sometimes it is not easy to model all our knowledge about a problem in terms of local preferences, but it may be easier to express some of our knowledge as preferences over solutions. For this reason, we need to provide the soft constraint system with a way to handle global preferences. The way we propose to do that is by pairing the soft constraint solver with a learning module, which takes the global preferences over the best solutions found so far by the constraint system, and learns the local preferences to be passed to the constraint system for the next interaction phase. In this way, the user interacts with a system which can handle both local and global preferences. Thus the new interaction model is now:

1. the user posts constraints and local preferences;
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) change its local preferences and go back to step 1;
(b) state some global preferences and give them to the system, and go back to step 2.

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. local preferences are given to the soft constraint solver;
2. global preferences (plus the current set of local preferences) are given to the learning module, which will learn appropriate local preferences to be passed to the soft constraint solver;
3. in both cases, the soft constraint solver will take the new preferences and return the best solutions according to the updated soft constraints.

Figure 2 shows the architecture and information flow of this interaction model.

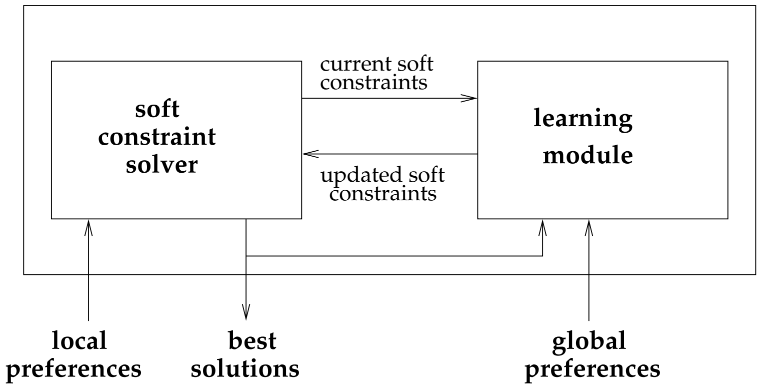

Fig. 2. The new interaction model, with a solver and a learning module.

It is obvious that, in this enhanced interaction model, the possibility of stating also global preferences provides a more complete acquisition of the user's knowledge of the problem to be solved, thus achieving a better model of the problem. Besides completeness, also a better precision is achieved. In fact, the local preferences can be rough estimates of what the user has in mind, and the learning module can be used to make such an estimate more precise at each interaction phase.

Notice also that the output of the system (that is, the best solutions found so far) is used to improve the current local preferences, 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, the learning module in our interaction framework receives only a few examples at a time. This is in line with the incremental learning procedure described in [9] 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 model the soft constraint problem has already some knowledge from the user, coded in terms of the current local preferences. Thus in our model the learning module is not used to learn the whole amount of knowledge to be coded as local preferences, but it just has to refine the current local preferences. This suggests that much fewer examples should be needed to encode the knowledge expressed as global preferences in terms of new local preferences in such a way that the error is small.

Notice that also the solver module can be used in an incremental mode. That is, the user can provide the solver with some constraints and local preferences at a time, and interleave the input of global preferences and of new constraints and local preferences. In this way, the system can receive more and more information as the interaction goes on, both in terms of global preferences over solutions, and also of local preferences over old and new constraints.

5 Our interactive system

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

- 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.
- The application field chosen for our system is the search for a house in the catalog of 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 is just 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 combination over others.

Our system consists of a Java layer, which takes care of the input and output interface, on top of the Con'flex [3] fuzzy constraint solver.

The user can post his constraints and preferences by inserting information into an appropriate window. In particular, he can post unary constraints over variables with numerical values, like the price of a house, and set an allowed range for such variables. Also, he can associate to each of such "numerical" constraints a level of importance. 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 satisfies fully 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.

The user can also post unary constraints over variables with a symbolic domain, like the type of the house. In this case, he can explicitly set a preference level for each domain element.

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 global 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 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.

The learning module in the prototype is very simple, but at least it behaves in such a way to discover irrelevant features. For example, if two houses have the same price, but they receive a very different global preference from the user, then it means that the price feature was irrelevant for the global preference. Therefore its preference value should not be modified in the fuzzy constraints. This is currently achieved by our system.

6 Conclusion and future work

We have described an interaction model where users can post both local and global preferences to a constraint system, which is able to handle both kinds of preferences via the association of a standard soft constraint solver and a learning module. The acquisition of knowledge is thus 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.

We are planning to extend our 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 the global preferences.

We will also study the behaviour of our 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 [5]. 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.

Acknowledgments

We would like to thank the company Aiglos, which supported the development of our interactive system, and the many people who gave us valuable suggestions on how to improve both our system and our interactive framework.

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. Con'flex web site: http://www.inra.fr/bia/T/conflex/Logiciels/adressesConflex.html
4. 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.
5. E. Freuder and R. Wallace. Suggestion strategies for constraint-based matchmaker agents. Proc. CP 1998, Springer Verlag, LNCS 1520, 1998.
6. L. Khatib, P. Morris, R. Morris, F. Rossi. Temporal Constraint Reasoning With Preferences. *Proc. IJCAI 2001.*

7. 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.
8. J. Hertz, A. Krogh, R. G. Palmer. Introduction to the Theory of Neural Computation. Addison-Wesley, 1991.
9. F. Rossi and A. Sperduti. Learning solution preferences in constraint problems. *Journal of Experimental and Theoretical Computer Science*, 1998. Vol 10.
10. Zs. Ruttkay. Fuzzy Constraint Satisfaction. Proc. 3rd IEEE International Conference on Fuzzy Systems, 1994.
11. T. Schiex. Possibilistic constraint satisfaction problems, or "how to handle soft constraints?". In *Proc. 8th Conf. of Uncertainty in AI*, pages 269–275, 1992.
12. T. Schiex and H. Fargier and G. Verfaille. Valued Constraint Satisfaction Problems: Hard and Easy Problems. Proc. IJCAI95, Morgan Kaufmann, 1995.
13. Edward P. K. Tsang. Foundations of Constraint Satisfaction. Academic Press, 1993.