In Neural Network World, Vol. 12, Number 5, pp. 421-431, 2002.

Modelling Soft Constraints: A Survey

Roman Barták*

Charles University in Prague, Faculty of Mathematics and Physics
Malostranské nám. 2/25, 118 00, Praha 1, Czech Republic
bartak@kti.mff.cuni.cz

Abstract. Constraint programming is an approach for solving (mostly combinatorial) problems by stating constraints over the problem variables. In some problems, there is no solution satisfying all the constraints or the problem formulation must deal with uncertainty, vagueness, or imprecision. In such a case the standard constraint satisfaction techniques dealing with hard constraints cannot be used directly and some form of soft constraints is required. In the paper we survey four generic models for soft constraints, namely hierarchical, partial, valued, and semiring-based constraint satisfaction.

1 Introduction

Many problems can be expressed in the form of variables with domains - sets of possible values for the variables - and constraints restricting the feasible combinations of variables' values. To solve the problem one needs to find a value for each variable in such a way that all the constraints are satisfied. This is a view of world using the glasses of *constraint programming* (CP) - a modern technology for solving combinatorial (optimisation) problems. In most current applications of CP, the variables' domains are finite, we are speaking about constraint satisfaction [7,8], but the CP framework is applicable to infinite domains as well (even if completely different technology is used for constraint solving over infinite domains).

Usually, the constraints are assumed to be hard, i.e., a tuple of values is either allowed or not. If there are many constraints imposed on the problem variables then it can happen that it is impossible to satisfy them all. Such problems are called over-constrained. In many other situations, we need to model fuzziness, possibilities, preferences, probabilities, costs, etc. so the crisp constraints are not enough to model and to solve such problems. Therefore, soft constraints have been proposed to model originally the over-constrained problems and later the problems with fuzziness, uncertainty etc. Soft constraint can be seen as a preferential constraint whose satisfaction is not required but preferred. There exist several models allowing users to describe how neatly the constraint is satisfied. In the paper we survey four generic models of soft constraints: constraint hierarchies, and partial, valued, and semiring-based constraint satisfaction.

* Supported by the Grant Agency of the Czech Republic under the contracts 201/99/D057 and 201/01/0942.

The models of soft constraints can be applied in several areas. Originally, soft constraints have been proposed to solve *over-constrained problems*, i.e., the problems where there is no solution satisfying all the constraints. In Section 2 we give a small example of the over-constrained problem. By weakening some constraints into preferential/soft constraints, the system may find a solution satisfying all the hard constraints and as many as possible preferential constraints. To distinguish between various levels of soft constraints, a hierarchy of levels can be used. We describe a soft constraints framework based on *constraint hierarchies* in Section 3. Instead of labelling constraints with preferences, it is possible to weaken the original problem in such a way that there exists a solution of the weakened problem. In Section 4 we describe several ways of problem weakening and the general concept based on this idea - *partial constraint satisfaction*. When the idea of soft constraints appeared, the researchers have found it useful not only for modelling the over-constrained problems but also for modelling the problems where the user states some preferences, uncertainty, fuzziness etc. We call these problems simply *soft problems*. In Section 5, we describe a concept of *valued constraint satisfaction* based on idea of adding valuations to constraints. A similar concept is used in *semiring-based constraint satisfaction* (Section 6) but the valuations are added to particular tuples rather than to constraints. Thanks to defining a partial order of valuations, the semiring-based constraint satisfaction can be naturally applied to solve multi-criteria *optimisation problems*.

2 A motivation example

For purposes of this paper we will use a simple, toy problem from [11] describing a robot seeking to choose matching clothes. Even if this is a small problem for pedagogical purposes, it could be regarded more seriously as a simple version of a configuration problem. The problem is to dress a robot using a minimal wardrobe: sneakers and Cordovans for footwear, a red and a white shirt, and three pairs of trousers: blue, denim, and grey. The fashion consultant told the robot that:

- the sneakers only go with the denim trousers,
- the Cordovans only go with the grey trousers and the white shirt,
- the white shirt goes with either denim or blue trousers, and
- the red shirt only goes with the grey trousers.

The fashion rules naturally form the constraints under which we operate. The variables correspond to parts of clothes: footwear, trousers, and shirt, while available wardrobe makes domains of the variables. Figure 1 shows the problem as a constraint graph. Visibly, there is no solution satisfying all the constraints, i.e., it is not possible to dress the robot in such a way that all the fashion rules are satisfied. Thus, it is an over-constrained problem. Still, we need to dress the robot somehow. In the next two sections we show some ways how to solve the robot clothing problem either by adding preferences to fashion rules (constraint hierarchies) or by weakening the original problem (partial constraint satisfaction).

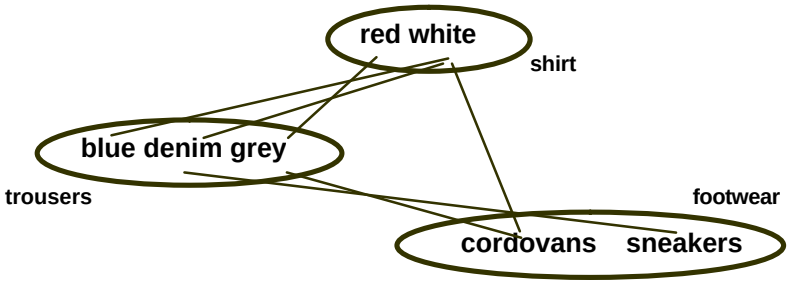

Fig. 1. A robot clothing problem. The ovals/nodes describe the variables with possible values inside them (domains). The edges connect fashion-feasible combinations of wardrobe.

3 Constraint hierarchies

As we mentioned at the end of the previous section, one of the ways how to solve the robot clothing problem is to ask the fashion consultant to put some preferences to individual constraints. Assume that the result of this consultation is that:

- the shirt must match trousers; it is a required constraint,
- it is strongly preferred if footwear matches trousers but it is not required; it is a strong constraint,
- it would be perfect if the shirt matches footwear but it is not a big faux pas if it does not match; it is a weak constraint.

Having the above preferences about the constraints we can reformulate the original problem. Now, the task is to find a solution satisfying all the required constraints and satisfying the preferential constraints as much as possible. Naturally, we prefer satisfaction of a stronger constraint to satisfaction of a weaker constraint.

Assuming the above modification of the robot clothing problem, we can find two solutions to it now:

- the white shirt with the denim trousers and the sneakers, or
- the red shirt with the grey trousers and Cordovans.

Both above solutions satisfy the required and the strong constraint, the weak constraint is violated. This is all right because we do not require satisfaction of all the constraints now and the weak constraint is the weakest constraint in the problem. Note also that if we change the preferences of the constraints, e.g. if we swap the strong and the weak constraints, we may get a completely different solution (white-denim-cordovans, white-blue-cordovans).

The idea of putting preferences to individual constraints is formalised in constraint hierarchies that have been proposed first in [4] and further developed in [5]. In constraint hierarchies, each constraint is labelled by a preference expressing the

strength of the constraint - we are speaking about the *labelled constraint*. Usually names like required, strong, medium, weak, and weakest are used to denote the preferences. The preferences are totally ordered so we can map them to natural numbers. Required constraints are mapped to level 0 and they must hold. The other constraints are merely preferential/soft so their satisfaction is not required. The soft preferential levels are mapped to positive natural numbers; the higher level number indicates the weaker level.

A *constraint hierarchy* is a finite set of labelled constraints defined over some set of values D called domain, e.g. real numbers. Given a constraint hierarchy H, H_0 is a vector of required constraints in H in some arbitrary order with their labels removed. Similarly, H_1 is a vector of the strongest non-required constraints in H etc. up to the weakest level H_n, where n is the number of non-required levels in the hierarchy H. We put $H_k = Æ$ for $k>n$. Recall, that if $i<j$ then the constraints in H_i are stronger (more preferred) than the constraints in H_j. We call the sets H_j *hierarchy levels*.

A *valuation* for the set of constraints is a function that maps variables in the constraints to elements in the domain D over which the constraints are defined. A *solution* to the constraint hierarchy is a set of valuations for the variables in the hierarchy such that any valuation in the solution set satisfies at least the required constraints, i.e., the constraints in H_0, and, in addition, it satisfies the non-required constraints, i.e., the constraints in H_i for $i>0$, at least as well as any other valuation that also satisfies the required constraints. In other words, there is no valuation satisfying the required constraints that is better than any valuation in the solution set. Formally:

$$S_0 = \{\ \theta \mid \forall c \in H_0\ \ c\theta \text{ holds}\ \}$$
$$S = \{\ \theta \mid \theta \in S_0\ \&\ \forall \sigma \in S_0\ \neg\ \text{better}(\sigma,\theta,H)\ \},$$

where S_0 is the set of valuations satisfying all the required constraints, S is the solution set, and "better" is a predicate to be explained below.

There is a number of reasonable candidates for the predicate *better*, which are called *comparators*. The comparator formally describes the idea that satisfaction of a stronger constraint is strictly preferred to satisfaction of an arbitrary number of weaker constraints. For example the *locally-better* comparator considers each constraint individually and it can be defined by the following way:

$$\text{locally-better}(\sigma,\theta,H) \equiv_{\text{def}}$$
$$\exists k>0\ \forall i \in \{1,\ldots,k-1\}\ \forall c \in H_i\ e(c,\sigma)=e(c,\theta)\ \&$$
$$\exists c' \in H_k\ e(c',\sigma)<e(c',\theta)\ \&\ \forall c \in H_k\ e(c,\sigma)=e(c,\theta),$$

where $e(c,q)$ is an *error function* indicating how nearly a constraint c is satisfied for a valuation q (a lower number means better satisfaction of the constraint). There exist other comparators like globally-better comparator that combines the errors of individual constraint at each hierarchy level using weighted-sum, least-squares and similar methods and then it compares the combined errors for two valuations.

Constraint hierarchies belong to the first and the most popular approaches handling preferences in constraint systems. A survey of constraint hierarchy framework and solving algorithms can be found in [1].

4 Partial Constraint Satisfaction

Let us go back to the robot clothing problem from Section 2. If a woman solves a similar problem and she finds that there is no matching wardrobe (which is usually very surprising assuming that wardrobe of women is much larger than our robot's wardrobe)[1] then she immediately has a solution: "I need to buy a new shirt, shoes etc." In constraint terminology, it corresponds to enlarging the domain of some variable so we can add new connections between new and old wardrobe and thus make the problem solvable. Or we can use a "men's approach" that is based on combining less elegant but existing wardrobe, e.g., we can decide that a white shirt goes with the sneakers after all (Figure 2). In constraint terminology, this corresponds to enlarging the domain of the constraint[2] by adding more compatible tuples to the constraint domain. Again, we made the problem weaker and hopefully solvable.

In addition to above light problem weakening, there exist two strong weakenings: we can decide to remove a constraint completely (e.g. do not care at all about matching of the shirt and footwear) or we can remove a variable (e.g., do not wear shoes).

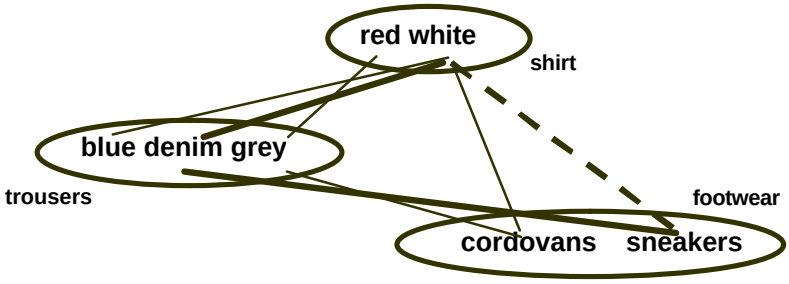

Fig. 2. By weakening the problem via extending the constraint domain (the dashed line), we make the robot clothing problem solvable (the bold lines indicate the solution).

In the above paragraph, we proposed four ways to weaken the constraint problem:

- enlarge the domain of a variable (i.e., add more values),
- enlarge the domain of a constraint (i.e., add more compatible tuples),
- remove the constraint from the problem,
- remove the variable from the problem.

Assume that the variable domain is defined as a unary constraint over the variable. Then, enlarging the domain of the variable is equivalent to enlarging the domain of the unary constraint over the variable. Moreover, if we enlarge the domain of the constraint to full extend, i.e., all the tuples are feasible according to the constraint, then the constraint does not restrict the solution - this is equivalent to removing the constraint. Finally, removing all the constraints binding a given variable is equivalent

[1] Please do not blame the author from sexism, this is just to make reading more relaxed (the role of man and woman in the above examples can be swapped, if the reader prefers it).

[2] A constraint domain is a set of tuples satisfying the constraint.

to removing the variable. Thus, we showed that the above ways of problem weakening collapse to enlarging the domain of a constraint.

By weakening the constraints we get alternative versions of the original problem. Now instead of solving the original problem we can look for a solution of some alternative problem. Naturally, we prefer this alternative problem to be close to the original problem so we can declare the crisp solution of the alternative problem to be the solution of the original problem. This idea has been formalised by Freuder and Wallace in the concept of partial constraint satisfaction [11].

Let PS be a set of constraint satisfaction problems with a partial order, $\leq$, on PS defined as follows: $P_1 \leq P_2$ iff the set of solutions to P_2 is a subset of the set of solution to P_1. If $P_1 \leq P_2$ but the two sets of solutions are not equal, we will write $P_1 < P_2$ and say that P_1 is weaker than P_2. We call the pair (PS, $\leq$) a *problem space*. The problem space can be obtained by considering all the ways of weakening the constraints in the original problem by allowing additional consistent combinations of values - as we described at the beginning of this chapter.

A *partial constraint satisfaction problem* (PCSP) can now be described more formally by supplying the original constraint satisfaction problem P, a problem space PS containing P, a metric on that space, and necessary and sufficient solution distances, N and S. A *solution* to a partial constraint satisfaction problem is defined as a problem P' from the problem space along with a solution to that problem where the metric distance of P' from P is less than or equal to N. A solution is sufficient if the metric distance is less than or equal to S; a solution is optimal if the metric distance of P' from P is minimal over the problem space.

The metric compares the problems from PS. Ideally, we might like to define it in terms of the partial order, i.e., the distance between P and P' equals to the number of solutions not shared by P and P'. Of course, computing such heuristic is not likely to be easy so we may consider other heuristics as well. Another natural heuristic is a count of the number of permitted value combinations not shared by the constraints of P and P'.

Partial constraint satisfaction is a generic framework for solving over-constrained problems but it can be also applied to solving hard problems where it is easier to find a solution of some relaxed alternative problem. Alternatively, PCSP can be seen as a generalisation of constraint satisfaction optimisation problem where the quality of the solution is measured by some objective function. In facts, the techniques used for solving PCSP, like branch and bound and local consistency count [11] are derived from the techniques applied to solving optimisation problems.

5 Valued Constraint Satisfaction

While partial constraint satisfaction defines the solution of the problem via weakening the problem, various other extensions of the original constraint satisfaction framework associates some valuation (usually a number) to each constraint. These extensions use a specific mathematical operator (+, max, etc.) to aggregate constraint violations. In [14] Schiex, Fargier, and Verfaillie proposed a generic framework that, rather than choosing a specific set for expressing valuations and a specific operator, uses an

ordered commutative monoid. The valuations are taken from the set of the monoid, combined using its operator and compared using the order. The other frameworks can then be specified by choosing a particular monoid as we show later in this chapter (Table 1).

To express the fact that the constraint may eventually be violated, the valued CSP annotates each constraint with a valuation taken from a set of valuations E equipped with the *valuation structure* $(E,\otimes,>,\bot,\top)$ where E is set of valuations totally ordered by >, with a minimum element $\bot$ and a maximum element $\top$. $\otimes$ is a commutative, associative binary operation on E with the unit element $\bot$ ($\bot\otimes a=a$), the absorbing element $\top$ ($\top\otimes a=\top$) and preserving monotonicity ($a\geq b \Rightarrow a\otimes c \geq b\otimes c$). The ordered set E allows different levels of constraint violations to be expressed. The element $\top$ corresponds to unacceptable violation, i.e., to express hard constraints. The element $\bot$ corresponds to complete satisfaction.

Let C be a set of constraints in the problem and φ be a mapping of the constraints to E, i.e. φ: $C\rightarrow E$. $\varphi(c)$ is called a valuation of the constraint *c*. The valuation of a particular assignment *A* of values to variables can be acquired by aggregating the valuations of violated constraints in C via $\otimes$:

$$v(A) = \bigotimes_{\substack{c\in C \\ \text{A violates c}}} \varphi(c)$$

Note that commutativity and associativity guarantee that the valuation of an assignment depends only on the set of the valuations of the violated constraints, and not on the way they are aggregated.

The *valued constraint satisfaction problem* is defined formally by a classical constraint satisfaction problem with the set of variables X, their domains D, and the set of constraints C, a valuation structure $(E,\otimes,>,\bot,\top)$, and a mapping φ from C to E. The task is to find an assignment *A* with a minimal valuation $v(A)$.

Valued CSP has been defined as an abstract framework to provide general solving algorithms and properties [14]. Some other extensions of CSP can be described as an instance of Valued CSP by choosing an appropriate algebraic structure (Table 1), details of conversion can be found in [12,14].

Framework	E	$\otimes$	>	$\bot$	$\top$
Classical CSP [7,8]	{true,false}	$\wedge$	>	true	false
Weighted CSP [15]	$N \cup \{+\infty\}$	+	>	0	$+\infty$
Probabilistic CSP [9]	$\langle 0,1\rangle$	$\times$	<	1	0
Possibilistic CSP [13]	$\langle 0,1\rangle$	max	>	0	1
Lexicographic CSP [10]	$N^{\langle 0,1)} \cup \{\top\}$	$\cup$	$>_{lex}$	$\varnothing$	$\top$

Table 1. Various (soft) constraint satisfaction frameworks can be expressed by using a particular valuation structure in Valued CSP.

6 Semiring-based Constraint Satisfaction

Semiring-based constraint satisfaction proposed by Bistarelli, Montanary, and Rossi [2] is another meta-approach for modelling problems with preferences. Instead of using the monoid, this framework uses a semiring structure, where the set of semiring specifies the preference associated to each tuple of values. The two semiring operations (+ and $\times$) then model constraint projection and combination respectively (to be defined below).

In semiring-based constraint satisfaction, each tuple in the constraint is marked by a preference level expressing how good the tuple satisfies the constraint. The preference level is taken from a set A equipped with the *c-semiring structure* $(A,+,\times,0,1)$. A is a set of preferences, + is a commutative, associative, idempotent (a+a=a) binary operation on A with the unit element 0 (0+a=a) and the absorbing element 1 (1+a=1), $\times$ is a commutative, associative binary operation on A with the unit element 1 (1×a=a) and the absorbing element 0 (0×a=0) and $\times$ distributes over +.

The multiplication operation $\times$ is used to combine constraints. Let $vars(c)$ be a set of variables over which the constraint c is defined, δ_c be a mapping of all tuples over $vars(c)$ to A, i.e., $\delta_c(V)$ is a preference of the tuple V in the constraint c, and let $U{\downarrow}Y$ be a projection of some tuple U to variables Y. Then we can describe a preference of some tuple V by combining preferences of this tuple (its projection) in all the constraints C:

$$p(V) = \times_{c \in C} \delta_c\left(V \downarrow vars(c)\right)$$

To compare the preferences of tuples we need some ordering on A. This ordering can be defined using the additive operation + in the following way: $a \le b \Leftrightarrow a+b=b$. If $a \le b$ then we say that b is better than a. Note that the relation $\le$ defines a partial ordering on A opposite to the total ordering used in the valued constraint satisfaction.

The *semiring-based constraint satisfaction problem* is defined formally by the c-semiring structure $(A,+,\times,0,1)$, the set of variables X, their domains D, and the set of constraints C described via δ_c. The task is to find an assignment V with the best preference $p(V)$.

Similarly to the valued CSP, it is possible to see many existing extensions of CSP as instances of the semiring-based CSP over a certain semiring (Table 2); for details look at [2,12].

Framework	A	+	$\times$	1	0
Classical CSP [7,8]	{false,true}	$\wedge$	$\vee$	true	false
Weighted CSP [15]	$\mathbb{N} \cup \{+\infty\}$	min	+	0	$+\infty$
Probabilistic CSP [9]	$\langle 0,1 \rangle$	max	$\times$	1	0
Possibilistic CSP [13]	$\langle 0,1 \rangle$	min	max	0	1
Fuzzy CSP [6]	$\langle 0,1 \rangle$	max	min	1	0
Lexicographic CSP [10]	$\mathbb{N}^{\langle 0,1 \rangle} \cup \{\top\}$	$\max_{lex}$	$\cup$	$\varnothing$	$\top$

Table 2. Various (soft) constraint satisfaction frameworks can be expressed as an instance of the semiring-based CSP.

Both semiring-based constraint satisfaction (SCSP) and valued constraint satisfaction (VCSP) are meta-frameworks based on some generic algebraic structure to describe preferences, costs, etc. There are two main differences between them:

- VCSP annotates constraints while SCSP puts preferences to individual tuples,
- VCSP works with a total order of preferences while SCSP uses a partial order.

If a total order of preferences is assumed then the frameworks can be converted each to another. A deep comparison of Semiring-based CSP and Valued CSP can be found in [3].

7 Are we ready for soft constraints?

In the paper we survey four generic models dealing with soft constraints, namely hierarchical, partial, valued, and semiring-based constraint satisfaction. Valued and semiring-based constraint satisfaction frameworks have been proposed for discrete (finite) domains only and so the partial constraint satisfaction. The idea of problem weakening in partial constraint satisfaction can be extended to infinite domains as well but we are not aware about any work on such extension. The constraint hierarchies are independent on domains and they are mostly applied to domains of real numbers.

There exist many other models that can be seen as instances of the above generic frameworks, like fuzzy [6], lexicographic [10], weighted [15], probabilistic [9], or possibilistic [13] constraint satisfaction (for a survey see [12]). Thus, the users have a broad range of models of soft constraints to choose from. The question is: “Do the users really use the soft constraints?” If we look at the main stream of today constraint satisfaction, the answer is probably no. The focus of constraint community is in solving large-scale combinatorial (optimisation) problems and in integration with other techniques like the methods of operations research. In fact the only alive, i.e., still being developed frameworks of soft constraints are constraint hierarchies and semiring-based constraint satisfaction.

In our opinion, the main reason for lukewarm acceptance of models for soft constraints is poor efficiency of the solving algorithms. The constraint hierarchy solvers are not incremental (after adding a new constraint, the solution must be in general recomputed from scratch) and the efficient algorithms for finite domain constraint hierarchies are still under development. The semiring-based solvers are close to optimisation algorithms and modelling using semiring-based constraints is not very practical due to necessity of adding preference to each tuple. Finally, the soft constraints are not included in any leading constraint packages like ILOG Solver, SICStus Prolog, or ECLiPSe.

Currently, it seems that the users solving problems with preference constraints turn attention to modelling such problems as optimisation problems with penalties etc. This is a pragmatic approach based on strong results of optimisation community so we believe that a closer integration of optimisation technology and soft constraints may be a good way of future development. Using optimisation techniques together with soft constraint propagation may improve efficiency both of optimisation and soft

constraint solvers. Moreover, in addition to optimisation problems the soft constraints frameworks deal with uncertainty, vagueness, or imprecision so the same solving algorithms for soft constraints can be applied to these problems as well.

References

[1] Barták, R.: Expertní systémy založené na omezujících podmínkách (in Czech), Ph.D. Thesis, Charles University, Prague, 1997.

[2] Bistarelli, S., Montanary, U., Rossi, F., Semiring-based Constraint Satisfaction and Optimisation, *Journal of ACM*, 1997.

[3] Bistarelli, S., Fargier, H., Montanary, U., Rossi, F., Schiech, T., Verfaillie, G., Semiring-based CSPs and Valued CSPs: Frameworks, properties, and comparison, *Constraints: An international journal*, 4(3), 1999.

[4] Borning, A., Duisberg, R., Freeman-Benson, B., Kramer, A., Woolf, M., Constraint Hierarchies, in *Proceedings of the 1987 ACM Conference on Object Oriented Programming Systems, Languages, and Applications*, pp.48-60, 1987.

[5] Borning, A., Maher, M., Martindale, A., Wilson, M., Constraint Hierarchies and Logic Programming, in *Proceedings of the Sixth International Conference on Logic Programming*, pp. 149-164, Lisbon, 1989.

[6] Dubois, D., Fargier, H., Prade, H., Propagation and satisfaction of flexible constraints, in *Fuzzy Sets, Neural Networks and Soft Computing*, pp. 166-187, New York, 1994.

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

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

[9] Fargier, H., Lang, J., Uncertainty in constraint satisfaction problems: a probabilistic approach, in *Proceedings of European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, Springer Verlag LNCS 747, 1993.

[10] Fargier, H., Lang, J., Schiex, T., Selecting preferred solutions in fuzzy constraint satisfaction problems, in the *First European Congress on Fuzzy and Intelligent Technologies*, Volume 3, pp. 1128-1134, 1993.

[11] Freuder, E.C., Wallace R.J., Partial Constraint Satisfaction, *Artificial Intelligence*, 58:21-70, 1992.

[12] Rudová, H., *Constraint Satisfaction with Preferences*, Ph.D. Thesis, Masaryk University, Brno, 2001.

[13] Schiex, T., Possibilistic constraint satisfaction problems or "How to handle soft constraints?", in *Proceedings of the Eighth International Conference on Uncertainty in Artificial Intelligence*, pp. 268-275, Stanford, 1992.

[14] Schiex, T., Fargier, H., Verfaillie, G., Valued Constraint Satisfaction Problems: Hard and Easy Problems, in *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp. 631-639, San Mateo, 1995.

[15] Shapiro, L.G., Haralick R.M., Structural descriptions and inexact matchings, in *IEEE Transactions Pattern Analysis Machine Intelligence*, 3:504-519, 1981.