Artificial Intelligence 174 (2010) 270–294

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Elicitation strategies for soft constraint problems with missing preferences: Properties, algorithms and experimental studies

Mirco Gelain [a], Maria Silvia Pini [a], Francesca Rossi [a], K. Brent Venable [a,*], Toby Walsh [b]

[a] Dipartimento di Matematica Pura ed Applicata, Università di Padova, Italy
[b] NICTA and UNSW Sydney, Australia

ARTICLE INFO

Article history:
Received 29 September 2008
Received in revised form 29 July 2009
Accepted 12 November 2009
Available online 17 November 2009

Keywords:
Preferences
Soft constraints
Incompleteness
Elicitation

ABSTRACT

We consider soft constraint problems where some of the preferences may be unspecified. This models, for example, settings where agents are distributed and have privacy issues, or where there is an ongoing preference elicitation process. In this context, we study how to find an optimal solution without having to wait for all the preferences. In particular, we define algorithms, that interleave search and preference elicitation, to find a solution which is necessarily optimal, that is, optimal no matter what the missing data will be, with the aim to ask the user to reveal as few preferences as possible. We define a combined solving and preference elicitation scheme with a large number of different instantiations, each corresponding to a concrete algorithm, which we compare experimentally. We compute both the number of elicited preferences and the user effort, which may be larger, as it contains all the preference values the user has to compute to be able to respond to the elicitation requests. While the number of elicited preferences is important when the concern is to communicate as little information as possible, the user effort measures also the hidden work the user has to do to be able to communicate the elicited preferences. Our experimental results on classical, fuzzy, weighted and temporal incomplete CSPs show that some of our algorithms are very good at finding a necessarily optimal solution while asking the user for only a very small fraction of the missing preferences. The user effort is also very small for the best algorithms.

1. Introduction

Traditionally, tasks such as scheduling, planning, and resource allocation have been tackled using several techniques, among which constraint reasoning is one of the most promising. The task is represented by a set of variables, their domains, and a set of constraints, and a solution of the problem is an assignment to all the variables in their domains such that all constraints are satisfied. Preferences or objective functions have been used to extend this formalism and allow for the modelling of constraint optimization, rather than satisfaction, problems. In all these approaches, the data (variables, domains, constraints) are completely known before the solving process starts. However, the increasing use of web services and in general of multi-agent applications demands for the formalization and handling of data that is only partially known when the solving process works, and that can be added later, for example via elicitation [23,24]. In many web applications, data may come from different sources, which may provide their piece of information at different times. Also, in multi-agent

* Corresponding author.
E-mail addresses: mgelain@math.unipd.it (M. Gelain), mpini@math.unipd.it (M.S. Pini), frossi@math.unipd.it (F. Rossi), kvenable@math.unipd.it (K.B. Venable), Toby.Walsh@nicta.com.au (T. Walsh).

0004-3702/$ – see front matter
doi:10.1016/j.artint.2009.11.015

settings, data provided by some agents may be voluntarily hidden due to privacy reasons, and only released if needed to find a solution to the problem.

Here we consider these issues focusing on constraint optimization problems where we look for an optimal solution. In particular, we consider problems where constraints are replaced by soft constraints, in which each assignment to the variables of the constraint has an associated preference coming from a preference set [1]. We assume that variables, domains, and constraint topology are given at the beginning, while the preferences are partially specified and are elicited during the solving process.

There are several application domains where this might be useful. One regards the fact that quantitative preferences, needed in soft constraints, may be difficult and tedious to provide for a user. Another one concerns multi-agent settings, where agents agree on the structures of the problem but they may provide their preferences on different parts of the problem at different times. Finally, some preferences can be initially hidden because of privacy reasons.

Formally, we take the soft constraint formalism and we allow for some preferences to be left unspecified. In our setting, users may know all the preferences but are willing to reveal only some of them at the beginning. Although some of the preferences can be missing, it could still be feasible to find an optimal solution. If not, we ask the user to provide some of the missing preferences and we start again from the new problem. We consider two notions of optimal solution: *possibly optimal* solutions are assignments to all the variables that are optimal in *at least one way* in which the currently unspecified preferences can be revealed, while *necessarily optimal* solutions are assignments to all the variables that are optimal in *all ways* in which the currently unspecified preferences can be revealed. This notation comes from multi-agent preference aggregation [17,20,21], where, in the context of voting theory, some preferences are missing but still one would like to declare a winner.

Given an incomplete soft constraint problem (ISCSP), its set of possibly optimal solutions is never empty, while the set of necessarily optimal solutions can be empty. Of course what we would like to find is a necessarily optimal solution, to be on the safe side: such solutions are optimal regardless of how the missing preferences are specified. Unfortunately, such a set may be empty. In this case there are two choices: either we may be satisfied with a possibly optimal solution, or we can elicit some of the missing preferences from the user and see if the new ISCSP has a necessarily optimal solution.

In this paper we follow this second approach and we repeat the process until the current ISCSP has at least one necessarily optimal solution. In order to do that, we exploit a modified version of the classical branch and bound scheme and we consider different elicitation strategies. In particular, we define a general algorithm scheme that is based on three parameters: *when* to elicit, *what* to elicit, and *who* chooses the value to be assigned to the next variable. For example, we may only elicit missing preferences after running branch and bound to exhaustion, or at the end of every complete branch, or even at every node in the search tree. Also, we may elicit all missing preferences related to the candidate solution, or we might just ask the user for the worst preference among some missing ones. Finally, when choosing the value to assign to a variable, we might ask the user, who knows or can compute (all or some of) the missing preferences, for help.

We test all possible instances of the scheme, obtained by selecting different elicitation strategies, on randomly generated soft constraint problems (fuzzy and weighted). By varying the number of variables, the tightness and density of constraints as well as the percentage of missing preferences, we produce a diversified and meaningful test set. The experiments demonstrate that some of the algorithms are very good at finding necessarily optimal solutions without eliciting too many preferences. We also test some of the algorithms on problems with hard constraints and on fuzzy temporal constraints. Our experimental study on randomly generated problems permits us to filter out algorithms with a poor performance and, thus, to identify those that are more promising for future testing on real-life scenarios.

In our experiments, we compute the elicited preferences, that is, the missing values that the user has to provide to the system because they are requested by the algorithm. Providing these values usually has a cost, either in terms of the computational effort, or in terms of a decrease in privacy, or in terms of the communication bandwidth. Whilst knowing *how many preferences are elicited* is important, we also compute a measure of the *user's effort*. This may be much larger than the number of elicited preferences, as it contains all the preference values the user may have to compute to be able to respond to the elicitation requests. For example, suppose we ask the user for the worst preference value among k missing ones. The user will communicate only one value, but he may have to compute and consider all k of them. Knowing the number of elicited preferences is important when the concern is to communicate as little information as possible. The user effort, on the other hand, measures the hidden work the user has to do to be able to communicate the elicited preferences. This user's effort is therefore also an important measure.

As a motivating example, recommender systems give suggestions based on partial knowledge of the user's preferences. Our approach could improve performance by identifying some key questions to ask before giving recommendations. Privacy concerns regarding the percentage of elicited preferences are motivated by eavesdropping. User's effort is instead related to the burden on the user. Our results show that the choice of the preference elicitation strategy is crucial for the performance of the solver. While the best algorithms need to elicit as little as 10% of the missing preferences, the worst ones need much more. The user's effort is also very small for the best algorithms. The performance of the best algorithms also shows that we only need to ask the user for a very small amount of additional information to be able to solve problems with missing data.

The paper is structured as follows. In Section 2 we define soft constraint problems, known in literature, where all the preferences are given. In Section 3 we introduce soft constraint problems where some preferences are missing (i.e., ISCSPs), we give new notions of optimal solutions, i.e., the possibly and the necessarily optimal solutions, and we characterize them

in Section 4. In Section 5 we present a general algorithmic scheme for ISCSPs with all its possible instances. In Section 6 we describe the problem generator used in the experimental studies and we indicate what we measure in the experiments. Next, in Section 7 we summarize and discuss our experimental comparison of all the algorithms. Finally, in Section 8 we compare our approach to other existing approaches to deal with incompletely specified constraint optimization problems, and in Section 9 we summarize the results contained in this paper, and we give some hints for future work.

Preliminary versions of parts of this paper have appeared in [12,13].

2. Soft constraints

A soft constraint [1] is just a classical constraint [5] where each instantiation of its variables has an associated value from a (totally or partially ordered) set. This set has two operations, which makes it similar to a semiring, and is called a c-semiring. More precisely, a c-semiring is a tuple $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ where A is a set, called the carrier of the c-semiring, and $\mathbf{0}, \mathbf{1} \in A$; $+$ is commutative, associative, idempotent, $\mathbf{0}$ is its unit element, and $\mathbf{1}$ is its absorbing element; $\times$ is associative, commutative, distributes over $+$, $\mathbf{1}$ is its unit element and $\mathbf{0}$ is its absorbing element. Consider the relation $\leqslant_S$ over A such that $a \leqslant_S b$ iff $a + b = b$. Then: $\leqslant_S$ is a partial order; $+$ and $\times$ are monotone on $\leqslant_S$; $\mathbf{0}$ is its minimum and $\mathbf{1}$ its maximum; $\langle A, \leqslant_S\rangle$ is a lattice and, for all $a, b \in A$, $a + b = lub(a, b)$. Moreover, if $\times$ is idempotent, then $\langle A, \leqslant_S\rangle$ is a distributive lattice and $\times$ is its glb. Informally, the relation $\leqslant_S$ gives us a way to compare (some of the) tuples of values and constraints. In fact, when we have $a \leqslant_S b$, we will say that *b is better than a*. Thus, $\mathbf{0}$ is the worst value and $\mathbf{1}$ is the best one.

Given a c-semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$, a finite set D (the domain of the variables), and an ordered set of variables V, a constraint is a pair $\langle def, con\rangle$ where $con \subseteq V$ is the scope of the constraint and $def : D^{|con|} \to A$ is the preference function of the constraint. Therefore, a constraint specifies a set of variables (the ones in con), and assigns to each tuple of values of D of these variables an element of the semiring set A. A soft constraint satisfaction problem (SCSP) is just a set of soft constraints over a set of variables.

Many classes of satisfaction or optimization problem can be defined in this formalism. A classical CSP is just an SCSP where the chosen c-semiring is: $S_{CSP} = \langle\{false, true\}, \vee, \wedge, false, true\rangle$. On the other hand, fuzzy CSPs [22,11] can be modelled in the SCSP framework by choosing the c-semiring: $S_{FCSP} = \langle[0, 1], max, min, 0, 1\rangle$. For weighted CSPs, the semiring is $S_{WCSP} = \langle\Re^+, min, +, +\infty, 0\rangle$. Here preferences are interpreted as costs from 0 to $+\infty$, which are combined with the sum and compared with min. Thus the optimization criterion is to minimize the sum of costs. For probabilistic CSPs [10], the semiring is $S_{PCSP} = \langle[0, 1], max, \times, 0, 1\rangle$. Here preferences are interpreted as probabilities ranging from 0 to 1, which are combined using the product and compared using max. Thus the aim is to maximize the joint probability.

Given an assignment s to all the variables of an SCSP P, i.e., a solution of P, we can compute its preference value $pref(P, s)$ by combining the preferences associated by each constraint to the sub-tuples of the assignments referring to the variables of the constraint. More precisely, $pref(P, s) = \prod_{\langle def, con\rangle \in C} def(s_{\downarrow con})$, where $\prod$ refers to the $\times$ operation of the semiring and $s_{\downarrow con}$ is the projection of tuple s on the variables in con. For example, in fuzzy CSPs, the preference of a complete assignment is the minimum preference given by the constraints. In weighted constraints, it is instead the sum of the costs given by the constraints.

Definition 1 *(Optimal solution).* An optimal solution of an SCSP P is a complete assignment s such that there is no other complete assignment s' with $pref(P, s) <_S pref(P, s')$. The set of optimal solutions of an SCSP P will be written as $Opt(P)$.

Notice that $Opt(P)$ is always well defined, since the domain D is finite, so there can only be finitely many preference values for an SCSP.

3. Incomplete soft constraint problems (ISCSPs)

Informally, an incomplete SCSP, written ISCSP, is an SCSP where the preferences of some tuples in the constraints, and/or of some of the values in the domains, are not specified. In detail, given a set of variables V with finite domain D, and c-semiring $S = \langle A, +, \times, 0, 1\rangle$, we extend the SCSP framework to incompleteness by the following definitions.

Definition 2 *(Incomplete soft constraint).* Given a set of variables V with finite domain D, and a c-semiring $\langle A, +, \times, 0, 1\rangle$, an incomplete soft constraint is a pair $\langle idef, con\rangle$ where $con \subseteq V$ is the scope of the constraint and $idef : D^{|con|} \to A \cup \{?\}$ is the preference function of the constraint. All tuples mapped into ? by $idef$ are called incomplete tuples.

In an incomplete soft constraint, the preference function can either specify the preference value of a tuple by assigning a specific element from the carrier of the c-semiring, or leave such preference unspecified. Formally, in the latter case the associated value is ?. A soft constraint is a special case of an incomplete soft constraint where all the tuples have a specified preference.

Definition 3 *(Incomplete soft constraint problem (ISCSP)).* An incomplete soft constraint problem is a pair $\langle C, V, D\rangle$ where C is a set of incomplete soft constraints over the variables in V with domain D. Given an ISCSP P, we will denote with $IT(P)$ the set of all incomplete tuples in P.

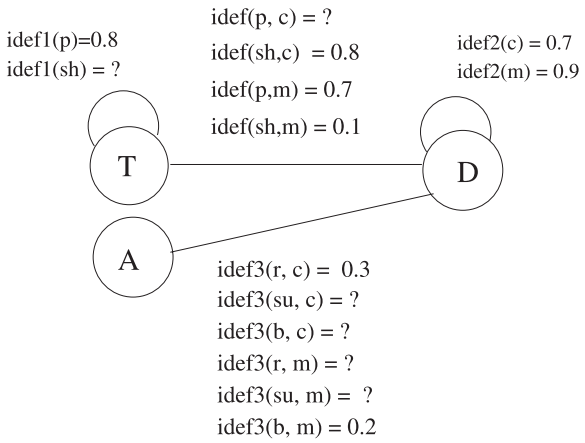


Fig. 1. An ISCSP.

Definition 4 *(Completion).* Given an ISCSP P, a completion of P is an SCSP P' obtained from P by associating to each incomplete tuple in every constraint an element of the carrier of the c-semiring. A completion is partial if some preference remains unspecified. We will denote with $C(P)$ the set of all possible completions of P and with $PC(P)$ the set of all its partial completions.

Example 1. A travel agency is planning Alice and Bob's honeymoon. The candidate destinations are the Maldive islands and the Caribbean, and they can decide to go by ship or by plane. To go to Maldives, they have a high preference to go by plane and a low preference to go by ship. For the Caribbean, they have a high preference to go by ship, and they don't give any preference on going there by plane.

Assume we use the fuzzy c-semiring $\langle [0,1], max, min, 0, 1\rangle$. We can model this problem by using two variables T (standing for *Transport*) and D (standing for *Destination*) with domains $D(T) = \{p, sh\}$ (p stands for *plane* and *sh* for *ship*) and $D(D) = \{m, c\}$ (m stands for *Maldives*, c for *Caribbean*), and an incomplete soft constraint $\langle idef, con\rangle$ with $con = \{T, D\}$ and preference function as shown in Fig. 1. The only incomplete tuple in this soft constraint is (p, c).

Also, assume that for the considered season the Maldives are slightly preferable to the Caribbean. Moreover, Alice and Bob have a high preference for plane as a means of transport, while they don't give any preference to ship. Moreover, as far as accommodations, which can be in a standard room, a suite, or a bungalow, assume that a suite in the Maldives is too expensive while a standard room in the Caribbean is not special enough for a honeymoon. To model this new information we use a variable A (standing for *Accommodation*) with domain $D(A) = \{r, su, b\}$ (r stands for *room*, *su* for *suite* and b for *bungalow*), and three constraints: two unary incomplete soft constraints, $\langle idef1, \{T\}\rangle$, $\langle idef2, \{D\}\rangle$ and a binary incomplete soft constraint $\langle idef3, \{A, D\}\rangle$. The definition of such constraints is shown in Fig. 1. The set of incomplete tuples of the entire problem is $IT(P) = \{(sh), (p, c), (su, c), (b, c), (r, m), (su, m)\}$.

Definition 5 *(Preference of an assignment, incomplete tuples).* Given an ISCSP $P = \langle C, V, D\rangle$ and an assignment s to all its variables, we denote with $pref(P, s)$ the preference of s in P and with $DEF(P, s)$ the set of soft constraints with no s-related missing preferences, that is, $DEF(P, s) = \langle idef, con\rangle \in C \mid idef(s_{\downarrow con}) \neq ?$. In detail, $pref(P, s) = \prod_{\langle idef, con\rangle \in DEF(P,s)} idef(s_{\downarrow con})$. Moreover, we denote by $it(s)$ the set of all the projections of s over constraints of P which have an unspecified preference.

The preference of an assignment s in an incomplete problem is thus obtained by combining the known preferences associated with the projections of the assignment, that is, of the appropriated sub-tuples in the constraints. The projections which have unspecified preferences, that is, those in $it(s)$, are simply ignored.

Example 2. Consider the two assignments $s_1 = (p, m, b)$ and $s_2 = (p, m, su)$ for the problem in Fig. 1. We have that $pref(P, s_1) = min(0.8, 0.7, 0.9, 0.2) = 0.2$, while $pref(P, s_2) = min(0.8, 0.7, 0.9) = 0.7$. However, while the preference of s_1 is fixed, since none of its projections is incomplete, the preference of s_2 may become lower than 0.7 depending on the preference of the incomplete tuple (su, m).

As shown by the example, the presence of incompleteness partitions the set of assignments into two sets: those which have a certain preference which is independent of how incompleteness is resolved, and those whose preference is only an upper bound, in the sense that it can be lowered in some completions. Given an ISCSP P, we will denote the first set of assignments as $Fixed(P)$ and the second with $Unfixed(P)$. In Example 2, $Fixed(P) = \{s_1\}$, while all other assignments belong to $Unfixed(P)$.

In SCSPs, an assignment is an optimal solution if its global preference is undominated. This notion can be generalized to the incomplete setting. In particular, when some preferences are unknown, we will speak of necessarily and possibly optimal solutions, that is, assignments which are undominated in all (resp., some) completions.

Definition 6 *(Necessarily and possibly optimal solution).* Given an ISCSP $P = \langle C, V, D\rangle$, an assignment $s \in D^{|V|}$ is a necessarily (resp., possibly) optimal solution iff $\forall Q \in C(P)$ (resp., $\exists Q \in C(P)$) $\forall s' \in D^{|V|}$, $pref(Q, s') \not> pref(Q, s)$.

Given an ISCSP P, we will denote with $NOS(P)$ (resp., $POS(P)$) the set of necessarily (resp., possibly) optimal solutions of P. Notice that, while $POS(P)$ is never empty, in general $NOS(P)$ may be empty. In particular, $NOS(P)$ is empty whenever the available preferences are not sufficient to establish the relationship between an assignment and all others.

Example 3. In the ISCSP P of Fig. 1, we can easily see that $NOS(P) = \emptyset$ since, given any assignment, it is possible to construct a completion of P in which it is not an optimal solution. On the other hand, $POS(P)$ contains all assignments not including tuple (sh, m).

4. Characterizing $POS(P)$ and $NOS(P)$

In this section we characterize the set of necessarily and possibly optimal solutions of an ISCSP given the preferences of the optimal solutions of two of the completions of P. All the results are given for ISCSPs defined on totally ordered c-semirings. In particular, given an ISCSP P defined on the c-semiring $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$, we consider:

- the SCSP $P_0 \in C(P)$, called the $\mathbf{0}$-completion of P, obtained from P by associating preference $\mathbf{0}$ to each tuple of $IT(P)$;
- the SCSP $P_1 \in C(P)$, called the $\mathbf{1}$-completion of P, obtained from P by associating preference $\mathbf{1}$ to each tuple of $IT(P)$.

Let us indicate respectively with $pref_0$ and $pref_1$ the preference of an optimal solution of P_0 and P_1. Due to the monotonicity of $\times$, and since $\mathbf{0} \leqslant \mathbf{1}$, we have that $pref_0 \leqslant pref_1$.

Example 4. Consider the problem shown in Fig. 1. We have that $pref_0 = 0.2$ and $pref_1 = 0.7$.

We will now give some lemmas that will be useful to show the following theorems.

Lemma 1. *Given an ISCSP P and the completion $P_1 \in C(P)$ as defined above, we have that $pref(P, s) = pref(P_1, s)$.*

Proof. Follows immediately from the definition of $pref(P, s)$ and from the fact that in a c-semiring $\mathbf{1}$ is the unit element. □

Lemma 2. *Given an ISCSP P and the completion $P_1 \in C(P)$ as defined above, there always exists an assignment s such that $pref(P, s) = pref_1$.*

Proof. Follows from Lemma 1 and choosing any $s \in Opt(P_1)$. □

Lemma 3. *Given an ISCSP P, the completions $P_0, P_1 \in C(P)$ as defined above, and another completion $P' \in C(P)$, then, $\forall s \in Opt(P')$, $pref_0 \leqslant pref(P', s) \leqslant pref_1$.*

Proof. Due to monotonicity, for any solution s we have that $pref(P', s) \leqslant pref(P_1, s) \leqslant pref_1$, since $pref_1$ is the optimal preference of P_1. Assume there is a solution $s \in Opt(P')$ such that $pref(P', s) < pref_0$. Then, for any solution $s_0 \in Opt(P_0)$, we have, by monotonicity, $pref(P', s) < pref_0 = pref(P_0, s_0) \leqslant pref(P', s_0)$. Thus, we have a contradiction, since s is an optimal solution of P'. □

Lemma 4. *Given an ISCSP P and the completion $P_1 \in C(P)$ as defined above, if $pref_1 > pref_0$, then $Opt(P_1) \subseteq Unfixed(P)$.*

Proof. Assume there is a fixed solution s such that $s \in Opt(P_1)$. Then we would have that $pref(P_0, s) = pref(P_1, s) = pref_1$ and thus $pref(P_0, s) > pref_0$ which is a contradiction, since $pref_0$ is the optimal preference in P_0. □

Lemma 5. *Given an ISCSP P, we have that $NOS(P) = \bigcap_{P' \in C(P)} Opt(P')$.*

Proof. Any solution $s \in \bigcap_{P' \in C(P)} Opt(P')$ satisfies the definition of necessarily optimal. Consider now $s \in NOS(P)$ and a completion P' of P. Then, by Definition 6, s cannot be dominated by another solution s', and thus $s \in Opt(P')$. □

In the following theorem we will show that, if $pref_0 > \mathbf{0}$, there is a necessarily optimal solution of P iff $pref_0 = pref_1$, and in this case $NOS(P)$ coincides with the set of optimal solutions of P_0.

Theorem 1. *Given an ISCSP P and the two completions $P_0, P_1 \in C(P)$ as defined above, if $pref_0 > \mathbf{0}$ we have that $NOS(P) \neq \emptyset$ iff $pref_1 = pref_0$. Moreover, if $NOS(P) \neq \emptyset$, then $NOS(P) = Opt(P_0)$.*

Proof. Since we know that $pref_0 \leqslant pref_1$, if $pref_0 \neq pref_1$ then $pref_1 > pref_0$. We prove that, if $pref_1 > pref_0$, then $NOS(P) = \emptyset$. Let us consider any assignment s of P. Due to the monotonicity of $\times$, for all $P' \in C(P)$, we have $pref(P', s) \leqslant pref(P_1, s) \leqslant pref_1$.

- If $pref(P_1, s) < pref_1$, then s is not in $NOS(P)$ since P_1 is a completion of P where s is not optimal.
- If instead $pref(P_1, s) = pref_1$, then $s \in Opt(P_1)$ and, by Lemma 4 we have that $s \in Unfixed(P)$. Thus we can consider completion P'_1 obtained from P_1 by associating preference $\mathbf{0}$ to the incomplete tuples of s. In P'_1 the preference of s is $\mathbf{0}$ and the preference of an optimal solution of P'_1 is, due to the monotonicity of $\times$, at least that of an optimal solution of P_0, that is $pref_0 > \mathbf{0}$. Thus $s \notin NOS(P)$.

Next we consider when $pref_0 = pref_1$. From Lemma 5 it follows that $NOS(P) \subseteq Opt(P_0)$. We will show that $NOS(P) \neq \emptyset$ by showing that any $s \in Opt(P_0)$ is in $NOS(P)$. Let us assume, on the contrary, that there is $s \in Opt(P_0)$ such that $s \notin NOS(P)$. Thus there is a completion P' of P with an assignment s' with $pref(P', s') > pref(P', s)$. By construction of P_0, any assignment $s \in Opt(P_0)$ must be in $Fixed(P)$. In fact, if it had some incomplete tuple, its preference in P_0 would be $\mathbf{0}$, since $\mathbf{0}$ is the absorbing element of $\times$. Since $s \in Fixed(P)$, $pref(P', s) = pref(P_0, s) = pref_0$. By construction of P_1 and monotonicity of $\times$, we have $pref(P_1, s') \geqslant pref(P', s')$. Thus the contradiction $pref_1 \geqslant pref(P_1, s') \geqslant pref(P', s') > pref(P', s) = pref_0$. This allows us to conclude that $s \in NOS(P) = Opt(P_0)$. □

In the theorem above we have assumed that $pref_0 > \mathbf{0}$. The case in which $pref_0 = \mathbf{0}$ needs to be treated separately. We consider it in the following theorem.

Theorem 2. *Given ISCSP $P = \langle C, V, D\rangle$ and the two completions $P_0, P_1 \in C(P)$ as defined above, assume $pref_0 = \mathbf{0}$. Then, if $pref_1 = \mathbf{0}$, $NOS(P) = D^{|V|}$. If $pref_1 > \mathbf{0}$, $NOS(P) = \{s \in Opt(P_1) \mid \forall s' \in D^{|V|}$ with $pref(P_1, s') > \mathbf{0}$ we have $it(s) \subseteq it(s')\}$.*

Proof. We prove the two items separately.

- If $pref_0 = pref_1 = \mathbf{0}$, then, from Lemma 3 it follows that the preference level of the optimal solution of SCSP P' is $\mathbf{0}$. Thus all assignments have always the same preference equal to 0. Thus they are all necessarily optimal solutions.
- Let us now assume that $\mathbf{0} = pref_0 < pref_1$. From Lemma 5, only assignments in $Opt(P_1)$ can be in $NOS(P)$ since all other assignments are not optimal in P_1. Let us now consider $s \in Opt(P_1)$. By Lemma 4 we have that $it(s) \neq \emptyset$. If there exists $s' \in D^{|V|}$, with $pref(P_1, s') > \mathbf{0}$, such that $it(s) \not\subseteq it(s')$ then we can construct a completion of P, say P' where s is not optimal. It is sufficient to set the preference of the tuples in $it(s')$ to $\mathbf{1}$ and the tuples in $it(s) \setminus it(s')$ to $\mathbf{0}$. We have that $pref(P', s) = \mathbf{0}$, since $\mathbf{0}$ is the absorbing element of $\times$, and $pref(P', s') = pref(P_1, s')$. Thus, in P' we have $pref(P', s') = pref(P_1, s') > pref(P', s) = \mathbf{0}$.
  We will now show that, if given $s \in Opt(P_1)$ there is no $s' \in D^{|V|}$ with $pref(P_1, s') > \mathbf{0}$ such that $it(s) \not\subseteq it(s')$, then $s \in NOS(P)$.
  First notice that, since $\mathbf{1}$ is the unit element of $\times$, $\forall P' \in C(P)$ $pref(P', s) = pref(P_1, s) \times it\text{-}pref(P', s)$ and $pref(P', s') = pref(P_1, s') \times it\text{-}pref(P', s')$ where $it\text{-}pref(P', s)$ (resp. $it\text{-}pref(P', s')$) is the combination of the preferences associated in P' to the incomplete tuples in $it(s)$ (resp. $it(s')$).
  Since for every $s' \in D^{|V|}$ with $pref(P_1, s') > \mathbf{0}$ we are assuming that $it(s) \subseteq it(s')$, then $\forall P' \in C(P)$, $it\text{-}pref(P', s) \geqslant it\text{-}pref(P', s')$, due to the intensive property of $\times$. Moreover, since $s \in Opt(P_1)$, $pref(P_1, s) = pref_1 > pref(P_1, s')$. Thus, for every $P' \in C(P)$, $\forall s' \in D^{|V|}$ (trivially for those with $pref(P_1, s') = \mathbf{0}$) we have that $pref(P', s) \geqslant pref(P', s')$. This allows us to conclude that $s \in NOS(P)$. □

Intuitively, if the tuples of s are not a subset of the incomplete tuples of some assignment s', then we can make s' dominate s in a completion by setting all the incomplete tuples of s' to $\mathbf{1}$ and all the remaining incomplete tuples of s to $\mathbf{0}$. In such a completion s is not optimal. Thus s is not a necessarily optimal solution. However, if the tuples of s are a subset of the incomplete tuples of all other assignments, then it is not possible to lower s without lowering all other tuples even further. This means that s is a necessarily optimal solution.

We now turn our attention to possible optimal solutions. Given a c-semiring $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$, it has been shown in [2] that idempotency and strict monotonicity of the $\times$ operator are incompatible, that is, at most one of these two properties can hold. In the following two theorems we show that the presence of one or the other of such two properties plays a key role in the characterization of $POS(P)$ where P is an ISCSP. In particular, if $\times$ is idempotent, then the possibly optimal solutions are the assignments with preference in P between $pref_0$ and $pref_1$. If, instead, $\times$ is strictly monotonic, then the possibly optimal solutions have preference in P between $pref_0$ and $pref_1$ and dominate all the assignments which have as set of incomplete tuples a subset of their incomplete tuples.

Theorem 3. *Given an ISCSP P defined on a c-semiring with idempotent $\times$ and the two completions $P_0, P_1 \in C(P)$ as defined above, if $pref_0 > \mathbf{0}$ we have that*: $POS(P) = \{s \in D^{|V|} \mid pref_0 \leqslant pref(P,s) \leqslant pref_1\}$.

Proof. First we show that any s such that $pref_0 \leqslant pref(P,s) \leqslant pref_1$ is in $POS(P)$. Let us consider the completion of P, P', obtained by associating preference $pref(P,s)$ to all the incomplete tuples of s and $\mathbf{0}$ to all other incomplete tuples of P. For any other assignment s' we can show that it never dominates s:

- $s' \in Fixed(P)$ and thus $pref(P',s') = pref(P_0,s') \leqslant pref_0 \leqslant pref(P,s)$;
- $s' \in Unfixed(P)$ and
  - $it(s') \not\subseteq it(s)$, then $pref(P',s') = \mathbf{0}$ since in P' the incomplete tuples in $it(s')$ which are not in $it(s)$ have been associated with preference $\mathbf{0}$;
  - $it(s') \subseteq it(s)$. By construction of P' and since $\times$ is idempotent and associative we have that: $pref(P',s) = (pref(P,s) \times (\prod_{|it(s)|} pref(P,s))) = pref(P,s)$ and $pref(P',s') = (pref(P,s') \times (\prod_{|it(s')|} pref(P,s))) = pref(P,s') \times pref(P,s)$. Since $\times$ is intensive, $pref(P',s') = (pref(P,s') \times pref(P,s)) \leqslant pref(P,s) = pref(P',s)$.

Thus in P' no assignment dominates s. This means that $s \in POS(P)$.

We will now show that if $s \in POS(P)$, $pref_0 \leqslant pref(P,s) \leqslant pref_1$. If $s \in POS(P)$, then $s \in Opt(Q)$ form some $Q \in C(P)$. Thus we can conclude by Lemma 3. □

Informally, given a solution s such that $pref_0 \leqslant pref(P,s) \leqslant pref_1$, it can be shown that it is an optimal solution of the completion of P obtained by associating preference $pref(P,s)$ to all the incomplete tuples of s, and $\mathbf{0}$ to all other incomplete tuples of P. On the other hand, by construction of P_0 and due to the monotonicity of $\times$, any assignment which is not optimal in P_0 cannot be optimal in any other completion. Also, by construction of P_1, there is no assignment s with $pref(P,s) > pref_1$.

Theorem 4. *Given an ISCSP P defined on a c-semiring with a strictly monotonic $\times$ and the two completions $P_0, P_1 \in C(P)$ as defined above, if $pref_0 > \mathbf{0}$ we have that*: *$s \in POS(P)$ iff $pref_0 \leqslant pref(P,s) \leqslant pref_1$ and $pref(P,s) = max\{pref(P,s') \mid it(s') \subseteq it(s)\}$.*

Proof. Let us first show that if assignment s is such that $pref_0 \leqslant pref(P,s) \leqslant pref_1$ and $pref(P,s) = max\{pref(P,s') \mid it(s') \subseteq it(s)\}$ it is in $POS(P)$. We must show there is a completion of P where s is undominated. Let us consider completion P' obtained by associating preference $\mathbf{1}$ to all the tuples in $it(s)$ and $\mathbf{0}$ to all the tuples in $IT(P) \setminus it(s)$. First we notice that $pref(P',s) = pref(P,s)$, since $\mathbf{1}$ is the unit element of $\times$. Let us consider any other assignment s'. Then we have one of the following:

- $it(s') = \emptyset$, which means that $s' \in Fixed(P)$ and thus $pref(P',s') = pref(P_0,s') \leqslant pref_0 \leqslant pref(P,s) = pref(P',s)$;
- $it(s') \not\subseteq it(s)$, which means that there is at least one incomplete tuple of $it(s')$ which is associated with $\mathbf{0}$. Since $\mathbf{0}$ is the absorbing element of $\times$, $pref(P',s') = \mathbf{0}$ and thus $pref(P',s') < pref_0 \leqslant pref(P',s)$;
- $it(s') \subseteq it(s)$, in this case $pref(P',s') = pref(P,s')$ since all tuples in $it(s')$ are associated with $\mathbf{1}$ in P'. However since $pref(P,s) = max\{pref(P,s') \mid it(s') \subseteq it(s)\}$, $pref(P',s') \leqslant pref(P',s)$.

We can thus conclude that s is not dominated by any assignment in P'. Hence $s \in POS(P)$.

Let us now prove the other direction by contradiction. If $pref(P,s) < pref_0$ then we can conclude by Lemma 2. We must prove that if $pref_0 \leqslant pref(P,s) \leqslant pref_1$ and $pref(P,s) < max\{pref(P,s') \mid it(s') \subseteq it(s)\}$ then s is not in $POS(P)$. In any completion P' of P we have that $pref(P',s) = pref(P,s) \times it\text{-}pref(P',s)$ and $pref(P',s') = pref(P,s') \times it\text{-}pref(P',s')$ where $it\text{-}pref(P',s)$ (resp. $it\text{-}pref(P',s')$) is the combination of the preferences associated to the incomplete tuples in $it(s)$ (resp. $it(s')$). Since $it(s') \subseteq it(s)$, for any completion P' we have that $it\text{-}pref(P',s) \leqslant it\text{-}pref(P',s')$. Moreover, let s'' be such that $pref(P,s'') = max\{pref(P,s') \mid it(s') \subseteq it(s)\}$. Then we have that for any completion P', $pref(P',s'') > pref(P',s)$ since $pref(P,s'') > pref(P,s)$ and $it\text{-}pref(P',s'') \geqslant it\text{-}pref(P',s)$ and $\times$ is strictly monotonic. Thus, if $pref_0 \leqslant pref(P,s) \leqslant pref_1$ and $pref(P,s) < max\{pref(P,s') \mid it(s') \subseteq it(s)\}$, then s is not in $POS(P)$. □

The intuition behind the statement of this theorem is that, if assignment s is such that $pref_0 \leqslant pref(P,s) \leqslant pref_1$ and $pref(P,s) = max\{pref(P,s') \mid it(s') \subseteq it(s)\}$, then it is optimal in the completion obtained associating preference $\mathbf{1}$ to all the tuples in $it(s)$ and $\mathbf{0}$ to all the tuples in $IT(P) \setminus it(s)$. On the contrary, if $pref(P,s) < max\{pref(P,s') \mid it(s') \subseteq it(s)\}$, there must be another assignment s'' such that $pref(P,s'') = max\{pref(P,s') \mid it(s') \subseteq it(s)\}$. It can then be shown that, in all completions of P, s is dominated by s''.

In contrast to $NOS(P)$, when $pref_0 = \mathbf{0}$ we can immediately conclude that $POS(P) = D^{|V|}$, independently of the nature of $\times$, since all assignments are optimal in P_0.

Corollary 4.1. *Given an ISCSP $P = \langle C, V, D\rangle$, if $pref_0 = \mathbf{0}$, then $POS(P) = D^{|V|}$.*

For ease of clarity, the results shown in this section can be summarized as follows:

- when $pref_0 = pref_1 = \mathbf{0}$
  - $NOS(P) = D^{|V|}$ (by Theorem 2);
  - $POS(P) = D^{|V|}$ (by Corollary 4.1);
- when $\mathbf{0} = pref_0 < pref_1$
  - $NOS(P) = \{s \in Opt(P_1) \mid \forall s' \in D^{|V|}$ with $pref(P_1, s') > \mathbf{0}$ we have $it(s) \subseteq it(s')\}$ (by Theorem 2);
  - $POS(P) = D^{|V|}$ (by Corollary 4.1);
- when $\mathbf{0} < pref_0 = pref_1$
  - $NOS(P) = Opt(P_0)$ (by Theorem 1);
  - if $\times$ is idempotent: $POS(P) = \{s \in D^{|V|} \mid pref_0 \leqslant pref(P, s) \leqslant pref_1\}$ (by Theorem 3);
  - if $\times$ is strictly monotonic: $POS(P) = \{s \in D^{|V|} \mid pref_0 \leqslant pref(P, s) \leqslant pref_1, pref(P, s) = max\{pref(P, s') \mid it(s') \subseteq it(s)\}\}$ (by Theorem 4);
- when $\mathbf{0} < pref_0 < pref_1$
  - $NOS(P) = \emptyset$ (by Theorem 1);
  - $POS(P)$ as for the case when $\mathbf{0} < pref_0 = pref_1$.

5. Solving ISCSPs

In this section we first describe a general schema for solving ISCSPs based on interleaving a branch and bound search with elicitation. Such a general schema is instantiated to different elicitation strategies generating several concrete algorithms. A computational analysis of the algorithms is provided both in terms of the number of elicited preferences and of the user effort for revealing some of the missing preferences.

5.1. The solver schema and its instances

The solving strategy which we propose for ISCSPs is based on the idea of combining a branch and bound search (B&B) with elicitation steps in which the user is asked to provide some type of missing information. In general, B&B proceeds by considering the variables in some order, by choosing a value for each variable in the order, and by computing, using some heuristics, an upper bound on the global preference of any completion of the current partial assignment. B&B also stores the highest preference (assuming the goal is to maximize) of a complete assignment found so far. If at any step the upper bound is lower than the preference of the current best solution, the search backtracks.

When some of the preferences are missing, as in ISCSPs, the agent may be asked for some preferences or other information regarding the preferences in order to know the true preference of a partial or complete assignment or in order to choose the next value for some variable. Preferences can be elicited after each run of B&B (as in [12]) or during a B&B run while preserving the correctness of the approach. For example, we can elicit preferences at the end of every complete branch (that is, regarding preferences of every complete assignment considered in the branch and bound algorithm), or at every node in the search tree (thus considering every partial assignment). Moreover, when choosing the value for the next variable to be assigned, we can ask the user (who knows the missing preferences) for help. Finally, rather than eliciting all the missing preferences in the possibly optimal solution, or the complete or partial assignment under consideration, we can elicit just some of the missing preferences.

For example, with incomplete fuzzy constraint problems (IFCSPs), eliciting just the worst preference among the missing ones is sufficient, since only the worst value is important to the computation of the overall preference value. Instead, with incomplete weighted constraint problems (IWCSPs), we need to elicit as many preference values as needed to decide whether the current assignment is better than the best one found so far.

More precisely, the algorithm schema we propose is based on the following parameters:

1. **WHO chooses the value of a variable:**
   (a) The algorithm, using one of the following heuristics:
      i. values are picked in decreasing order w.r.t. their preference values in the **1**-completion. The order is maintained dynamically. We denote this heuristic with *dp*;
      ii. values are picked in decreasing order w.r.t. the preferences in the **0**-completion of the initial ISCSP. The order is thus static. We denote this heuristic with *dpi*.
   (b) The user, revealing the value that he prefers according to one of the following criteria:
      i. the value is the most preferred among those in the domain which haven't been considered yet (*lazy user*, *lu* for short);
      ii. the value is the most preferred among those which haven't been considered yet given the constraints involving the current variable and the past variables in the search order (*smart user*, *su* for short).
2. **WHAT must be elicited:**
   (a) the preferences of all the incomplete tuples of the current assignment (denoted with *all*);

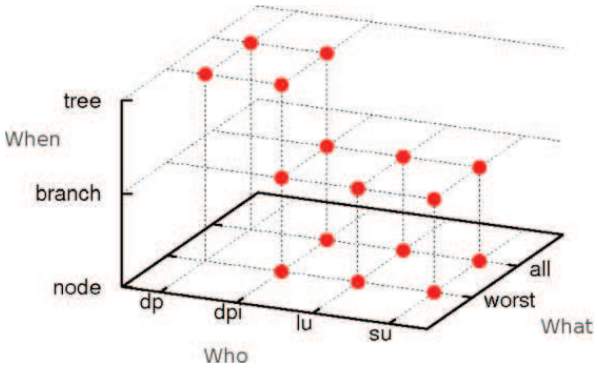


Fig. 2. The algorithms for IFCSPs.

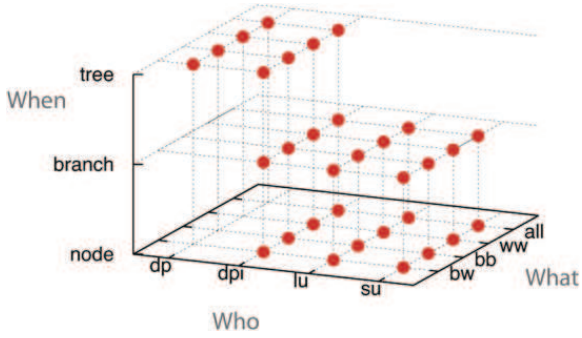


Fig. 3. The algorithms for IWCSPs.

(b) for IFCSPs, only the preference of the worst tuple of the current assignment, if it is worse than the known ones (denoted with *worst*);
(c) for IWCSPs:
- the worst missing cost (that is, the highest) until either all the costs are elicited or the current global cost of the (possibly partial) assignment is higher than the optimum found so far. This strategy is denoted by WW;
- the best (i.e. the minimum) cost until either all the costs are elicited or the current global cost of the (possibly partial) assignment is higher than the optimum found so far. This strategy is denoted by BB;
- the best and the worst cost in turn. This strategy is denoted by BW.

3. **WHEN elicitation should take place:**
(a) at the end of the branch and bound search (at *tree* level);
(b) during the search, when we have a complete assignment to all the variables (i.e., when we have reached a leaf of the search tree, and thus when we are at the end of a *branch*). We will refer to such a heuristics, by saying "at *branch* level";
(c) during search, whenever a new value is assigned to a variable. We will refer to such a heuristics, by saying "at the *node* level".

Summarizing, we have three features which we call *who, what* and *when*. There are four possible choices for *who*: *dp*, *dpi*, *lu*, and *su*. If we work with IFCSPs, there are two possibilities for *what*: *all* and *worst*. Instead, with IWCSPs, there are four options: *all*, WW, BB, and BW. Finally, there are three options for *when*: *tree*, *branch*, and *node*. If $when = tree$, elicitation takes place only when the search is completed. This means that the B&B search can be performed more than once. In contrast, if $when = branch$ or $when = node$, the B&B search is performed only once and the elicitation is done either at every node of the search tree or at every leaf.

By choosing a value for each of the three parameters above in a consistent way, we obtain, for IFCSPs, a total of 16 different algorithms, as summarized in Fig. 2.

If instead we work with IWCSPs, we have a total of 32 algorithms, as can be seen in Fig. 3.

The pseudo-code of our general solver, which we call ISCSP-SCHEME, is shown in Algorithms 1 and 2. Every point in Figs. 2 and 3 represents an instantiation of ISCSP-SCHEME to specific values for parameters *who*, *what* and *when*.

ISCSP-SCHEME takes in input an ISCSP P and the values for the three parameters: *who*, *what*, and *when*. It returns an ISCSP Q, a complete assignment s and a preference p. In Theorems 5 and 7 we will show that Q is a partial completion of P and s is a necessarily optimal solution of Q with preference p. As a first step, ISCSP-SCHEME computes the $\mathbf{0}$-completion of P, called P_0, and finds one of its optimal solutions, say s_{max}, and its preference, say $pref_{max}$, by applying a standard branch and bound procedure (denoted by B&B). Next, procedure BBE is called. If BBE succeeds, it returns a partial completion of P, one of its necessarily optimal solutions, and its associated preference. Otherwise, it returns a solution equal to *nil*. In the first

Algorithm 1: ISCSP-SCHEME

Input: an ISCSP P, a parameter *who* indicating the method of values instantiation, a parameter *what* indicating the elicitation policy, a parameter *when* indicating the level at which the elicitation must be done
Output: an ISCSP Q, an assignment s, a preference p

```
compute P_0
Q ← P_0
s_max, pref_max ← B&B(P_0, −)
Q′, s_1, pref_1 ← BBE(P, 0, who, what, when, s_max, pref_max)
if s_1 ≠ nil then
    s_max ← s_1
    pref_max ← pref_1
    Q ← Q′
return Q, s_max, pref_max
```

Algorithm 2: BBE

Input: an ISCSP P, the number of currently instantiated variables *nInstVar*, a parameter *who* indicating the method of values instantiation, a parameter *what* indicating the elicitation policy, a parameter *when* indicating the level at which the elicitation must be done, a reference to a solution *sol*, *lb* lower bound
Output: an ISCSP P, a solution *sol* and its preference *pref*

```
sol′ ← sol
pref′ ← lb
currentVariable ← nextVariable(P[?/1])
while nextValue(currentVariable, who) do
    if when = node then
        P, pref ← Elicit@Node(what, P, currentVariable, lb)
    ub ← UpperBound(P[?/1], currentVariable)
    if ub >_S lb then
        if nInstvar = number of variables in P then
            if when = branch then
                P, pref ← Elicit@branch(what, P, lb)
            if pref >_S lb then
                sol ← getSolution(P[?/1])
                lb ← pref(P[?/1], sol)
        else
            BBE(P, nInstVar + 1, who, what, when, sol, lb)
if when = tree and nInstVar = 0 then
    if sol = nil then
        sol ← sol′
        pref ← pref′
    else
        P, pref ← Elicit@tree(what, P, sol, lb)
        if pref >_S pref′ then
            BBE(P, 0, who, what, when, sol, pref)
        else
            BBE(P, 0, who, what, when, sol′, pref′)
```

case the output of ISCSP-SCHEME coincides with that of BBE, otherwise ISCSP-SCHEME returns P_0 and one of its optimal solutions with the corresponding preference.

Procedure BBE takes as input the same values as ISCSP-SCHEME and, in addition, a solution *sol* and a preference *lb* representing the current lower bound of the optimal preference level. Solution sol' and preference $pref'$ are initialized to such values at the beginning of BBE. Procedure *nextVariable* applied to the **1**-completion of the ISCSP in input (denoted by $P[?/1]$) allows to assign to *currentVariable* the next variable to be assigned. The algorithm then assigns a value to this variable. If the Boolean function *nextValue* returns true (if there is a value in the domain), we select a value for *currentVariable* according to the value of parameter *who*.

The computation of the upper bound for the preference that can be obtained by any completion of the current partial assignment is performed by procedure *UpperBound*. In general, any kind of upper bound can be used. However, we have chosen to estimate it by combining the preferences of the constraints involving only variables that have already been instantiated. Formally, let t be the current partial assignment to variables in $\{v_1, \ldots, v_k\} \subseteq V$, and let $c_i = \langle def_i, con_i \rangle$ be a constraint, such that $con_i \subseteq \{v_1, \ldots, v_k\}$. Then, the value ub returned by *UpperBound* is:

$$ub = \prod_{i=1}^{k} def_i(t \downarrow_{con_i}),$$

where $\prod$ is the combination operator of the semiring.

We will now describe procedure BBE by considering the various values for parameter *when*. This corresponds to consider the algorithms in Figs. 2 and 3 divided into the three horizontal planes obtained fixing the value on the *when*-axis.

- If *when* = *tree*, elicitation is handled by procedure *Elicit@tree* and takes place only at the end of the search over the **1**-completion. The user is not involved in the value assignment steps within the search and thus there are only two possible values for variable *who*, i.e. *dp* and *dpi*. At the end of the search, if a solution is found, the user may be asked to reveal all the preferences of the incomplete tuples in the solution (if *what* = *all*). If we work with IFCSPs, we could also ask for just the worst one among the missing preferences if it is worst than the known ones (if *what* = *worst*). If instead we work with IWCSPs, preferences can be asked in decreasing (*what* = BB), increasing (*what* = WW), or alternating order (*what* = BW) until we have enough information. If the preference of this solution is better than the best found so far, BBE is called recursively with the new best solution and preference, otherwise the recursive call is done with the old solution and preference.
- If *when* = *branch*, B&B is performed only once and not several times as in the previous case. The user may be asked to choose the next value for the current variable being instantiated. Preference elicitation, which is handled by function *Elicit@branch*, takes place during search, whenever all variables have been instantiated. As above, the user can be asked either to reveal the preferences of all or some of the incomplete tuples depending on the value of *what*. In all cases the information gathered is sufficient to compare the preference of the current assignment against the current lower bound.
- If *when* = *node*, preferences are elicited every time a new value is assigned to a variable, and it is handled by procedure *Elicit@node*. The tuples to be considered for elicitation are those involving the value which has just been assigned and belonging to constraints between the current variable and already instantiated variables. The value of *what* determines whether one or all or some preference values involving the new assignment are asked to the user. With the information given by the user, the preference of the current partial assignment is updated in order to determine if the subtree rooted at the current node can be pruned.

5.2. Termination and correctness

We will now prove that algorithm ISCSP-SCHEME, when given an ISCSP in input, always terminates generating a completion of the ISCSP and one of its necessarily optimal solution.

Theorem 5. *Given an ISCSP P and when = tree, if*

- *what = all, or*
- *what = worst and P is an IFCSP, or*
- *(what = WW or what = BB or what = BW) and P is an IWCSP,*

algorithm ISCSP-SCHEME *always terminates and returns an ISCSP Q such that $Q \in PC(P)$, an assignment $s \in NOS(Q)$, and its preference in Q.*

Proof. Clearly ISCSP-SCHEME terminates if and only if BBE terminates. If we consider the pseudo-code of procedure BBE shown in Algorithm 2, we see that if *when* = *tree*, BBE terminates when *sol* = *nil*. This happens only when the search fails to find a solution of the current problem with a preference strictly greater than the current lower bound, i.e., when the condition $pref >_S lb$ is never satisfied. Let us denote with Q^i and Q^{i+1} respectively the ISCSPs given in input to the i-th and $(i+1)$-th recursive call of BBE. First we notice that only procedure *Elicit@tree* modifies the ISCSP in input by possibly adding new elicited preferences. Moreover, whatever the value of parameter *what* is, the returned ISCSP is either the same as the one in input or it is a (possibly partial) completion of the one in input. Thus we have $Q^{i+1} \in PC(Q^i)$ and $Q^i \in PC(P)$. Since the search is always performed on the **1**-completion of the current ISCSP, we can conclude that for every solution s, $pref(Q^{i+1}, s) \leqslant_S pref(Q^i, s)$. Let us now denote with lb^i and lb^{i+1} the lower bounds given in input respectively to the i-th and $(i+1)$-th recursive call of BBE. It is easy to see that $lb^{i+1} \geqslant_S lb^i$. Thus, since at every iteration the preferences of solutions cannot increase and the bound cannot decrease, and since we have a finite number of solutions, we can conclude that BBE always terminates.

The reasoning that follows relies on the fact that value *pref* returned by function *Elicit@tree* is the final preference after elicitation of assignment *sol* given in input. This is true since either *what* = *all* and thus all preferences have been elicited and the overall preference of *sol* can be computed, or only the *worst* preference has been elicited but in a fuzzy context where the overall preference coincide with the worst one, or we are in an IWCSP and we have elicited enough preferences to discover that the current solution is worst then the optimum found so far or we have elicited all its costs.

If called with *when* = *tree* ISCSP-SCHEME exits when the last branch and bound search has ended returning *sol* = *nil*. In such a case *sol* and *pref* are updated to contain the best solution and associated preference found so far, i.e. sol' and $pref'$. Then, the algorithm returns the current ISCSP, say Q, and *sol* and *pref*. Following the same reasoning as above done for Q^i, we can conclude that $Q \in PC(P)$.

At the end of a while loop execution of the first call of BBE (the bottom of the call stack), assignment *sol* either contains an optimal solution *sol* of the **1**-completion of the current ISCSP or $sol = nil$. $sol = nil$ iff there is no assignment with preference higher than *lb* in the **1**-completion of the current ISCSP. In this situation, sol' and $pref'$ are an optimal solution and preference of the 1-completion of the current ISCSP. However, since the preference of sol', $pref'$ is fixed and since, due to monotonicity, the optimal preference value of the **1**-completion is always better than or equal to that of the **0**-completion, we have that sol' and $pref'$ are an optimal solution and preference of the 0-completion of the current ISCSP as well.

By Theorems 1 and 2, we can conclude that $NOS(Q)$ is not empty. If $pref = \mathbf{0}$, then $NOS(Q)$ contains all the assignments and thus also *sol*. The algorithm correctly returns the same ISCSP given in input, assignment *sol* and its preference *pref*. If instead $\mathbf{0} < pref$, again the algorithm is correct, since by Theorem 1 we know that $NOS(Q) = Opt(Q[?/0])$, and we have shown that $sol \in Opt(Q[?/0])$. □

Moreover, if parameter $when = tree$, then no useless work is done since we only elicit preferences of assignments that occur in an optimal solution of some completion of the current incomplete problem.

Theorem 6. *If* ISCSP-SCHEME *is given in input* $when = tree$*, then only preferences of tuples of solutions in* $POS(P)$ *are elicited.*

Proof. If $when = tree$ then, during the execution of ISCSP-SCHEME, preferences are elicited only by procedure *Elicit@tree*. A call to such a procedure, such as $Elicit@tree(what, P, sol, lb)$, depending on the value of parameter *what*, elicits all or a subset of the preferences of the incomplete tuples of assignment *sol*, returning the (eventually) new global preference of *sol*, *pref* and the completion of *P* obtained adding the new elicited preferences. During the execution of ISCSP-SCHEME, *Elicit@tree* is called on the current partial completion of the ISCSP given in input, *P* and on an optimal solution of its **1**-completion, *sol*. By Theorems 3 and 4, any optimal solution of the **1**-completion of the current partial completion of *P* is a possibly optimal solution of such a partial completion. □

We will now consider other values for parameter *when*.

Theorem 7. *Given a fuzzy or weighted ISCSP P and* ($when = branch$ *or* $when = node$)*, algorithm* ISCSP-SCHEME *always terminates, and it returns an ISCSP Q such that* $Q \in PC(P)$*, an assignment* $s \in NOS(Q)$*, and its preference in Q.*

Proof. In order to prove that the algorithm terminates, it is sufficient to show that BBE terminates. Since the domains are finite, the labelling phase produces a number of finite choices at every level of the search tree. Moreover, since the number of variables is limited, then, we have also a finite number of levels in the tree. Hence, BBE considers at most all the possible assignments, that are a finite number. At the end of the execution of ISCSP-SCHEME, *sol*, with preference *pref* is one of the optimal solutions of the current $P[?/1]$. Thus, for every assignment s', $pref(P[?/1], s') \leqslant_S pref(P[?/1], sol)$. Moreover, for every completion $Q' \in C(P)$ and for every assignment s', $pref(Q', s') \leqslant_S pref(P[?/1], s')$. Hence, for every assignment s' and for every $Q' \in C(P)$, we have that $pref(Q', s') \leqslant_S pref(P[?/1], sol)$. In order to prove that $sol \in NOS(P)$, now it is sufficient to prove that for every $Q' \in C(P)$, $pref(P[?/1], sol) = pref(Q', sol)$. This is true, since $sol \in Fixed(P)$ both when eliciting all the missing preferences, and when eliciting only the worst one for fuzzy ISCSPs, and when eliciting via BB, BW, or WW in weighted ISCSPs. In fact, in both cases, the preference of *sol* is the same in every completion. To show that the final problem *Q* returned by BBE is in $PC(P)$, it is sufficient to note that only the procedures *Elicit@node* and *Elicit@branch* modify the ISCSP in input by possibly adding some missing preferences. Thus, the returned ISCSP is in $PC(P)$. □

6. Problem generator and experimental design

To test the performance of these different algorithms, we created fuzzy ISCSPs using a generator which is a simple extension of the standard random model for hard constraints to soft and incomplete constraints. The generator has the following parameters:

- n: number of variables;
- m: cardinality of the variable domains;
- d: density, that is, the percentage of binary constraints present in the problem w.r.t. the total number of possible binary constraints that can be defined on n variables;
- t: tightness, that is, the percentage of tuples with preference 0 in each constraint and in each domain w.r.t. the total number of tuples (m^2 for the constraints, since we have only binary constraints, and m in the domains);
- i: incompleteness, that is, the percentage of incomplete tuples (that is, tuples with preference ?) in each constraint and in each domain.

Given values for these parameters, we generate IFCSPs as follows. We first generate n variables and then d% of the $n(n-1)/2$ possible constraints. Then, for every domain and for every constraint, we generate a random preference value in $(0, 1]$ for

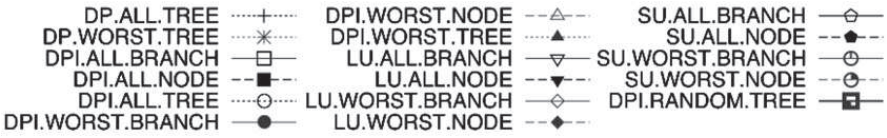


Fig. 4. Algorithm names and corresponding line symbols.

each of the tuples (that are m for the domains, and m^2 for the constraints); we randomly set t% of these preferences to 0; and we randomly set i% of the preferences as incomplete.

For example, if the generator is given in input $n = 10$, $m = 5$, $d = 50$, $t = 10$, and $i = 30$, it generates a binary IFCSP with 10 variables, each with 5 elements in the domain, 22 constraints (that is 50% of $45 = 10(10-1)/2$), 2 tuples with preference 0 (that is, 10% of $25 = 5 \times 5$) and 7 incomplete tuples (that is, 30% of $25 = 5 \times 5$) in each constraint, and 1 missing preference (that is, 30% of 5) in each domain. Notice that we use a model B generator: density and tightness are interpreted as percentages, and not as probabilities [14].

We also generate random IWSCSPs using the same parameters as for IFCSPs, with costs in $[0, 10] \cup \{+\infty\}$.

Our experiments measure the *percentage of elicited preferences* (over all the missing preferences) as the generation parameters vary. Since some of the algorithm instances require the user to suggest the value for the next variable, or ask for the worst value among several, we also show the *user's effort* in the various solvers, formally defined as the percentage of missing preferences the user has to consider to give the required help.

Besides the 16 instances of the scheme for IFCSPs described above, we also considered a "baseline" algorithm that elicits preferences of randomly chosen tuples every time branch and bound ends. All algorithms are named by means of the three parameters. For example, algorithm DPI.WORST.BRANCH has parameters $who = dpi$, $what = worst$, and $when = branch$. For the baseline algorithm, we use the name DPI.RANDOM.TREE.

For every choice of parameter values, 100 problem instances are generated. The results shown are the average over the 100 instances. Also, when it is not specified otherwise, we set $n = 10$ and $m = 5$. However, for IFCSPs, we have similar results for $n = 5$, 8, 11, 14, 17, and 20. All our experiments have been performed on an AMD Athlon 64x2 2800+, with 1 Gb RAM, Linux operating system, and using JVM 6.0.1.

7. Results

In this section we summarize and discuss our experimental comparison of the different algorithms. We first focus on fuzzy ISCSPs. We then consider two special cases: incomplete CSPs where all constraints are hard, and incomplete fuzzy temporal problems. Finally, we consider incomplete weighted CSPs. In all the experimental results, the association between an algorithm name and a line symbol is shown in Fig. 4.

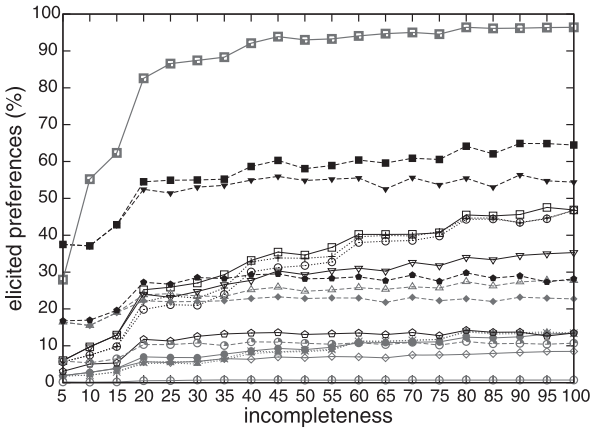
7.1. Incomplete fuzzy CSPs

We first described the experiments on incomplete fuzzy CSPs. Fig. 5 shows the percentage of elicited preferences when we vary the incompleteness, the density, and the tightness, respectively. We show only the results for specific values of the parameters. However, the trends observed here hold in general. It is easy to see that the best algorithms are those that elicit at the branch level. In particular, algorithm SU.WORST.BRANCH elicits a very small percentage of missing preferences (less than 5%), no matter the amount of incompleteness in the problem, and also independently of the density and the tightness. This algorithm outperforms all others, but relies on help from the user. The best algorithm that does not need such help is DPI.WORST.BRANCH. This never elicits more than about 10% of the missing preferences. Notice that the baseline algorithm is always the worst one, and needs nearly all the missing preferences before it finds a necessarily optimal solution. Notice also that the algorithms with $what = worst$ are almost always better than those with $what = all$, and that $when = branch$ is almost always better than $when = node$ or $when = tree$.

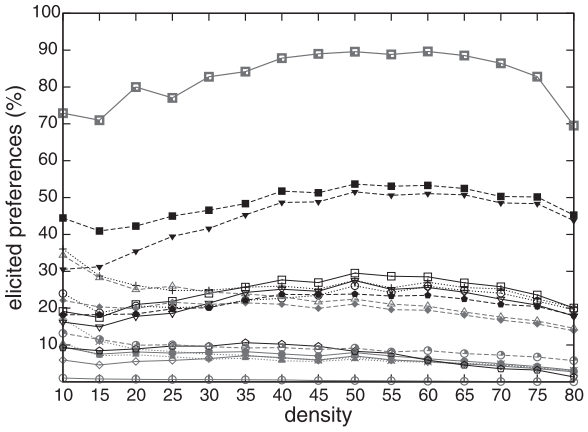
Fig. 6(a) shows the user's effort as incompleteness varies. As could be predicted, the effort grows slightly with the incompleteness level, and it is equal to the percentage of elicited preferences only when $what = all$ and $who = dp$ or dpi. For example, when $what = worst$, even if $who = dp$ or dpi, the user has to consider more preferences than those elicited, since to identify the worst preference value the user needs to check all of them (that is, those involved in a partial or complete assignment). DPI.WORST.BRANCH requires the user to look at 60% of the missing preferences at most, even when incompleteness is 100%.

Fig. 6(b) shows the user's effort as density varies. Also in this case, as expected, the effort grows slightly with the density level. In this case DPI.WORST.BRANCH requires the user to look at most 40% of the missing preferences, even when the density is 80%.

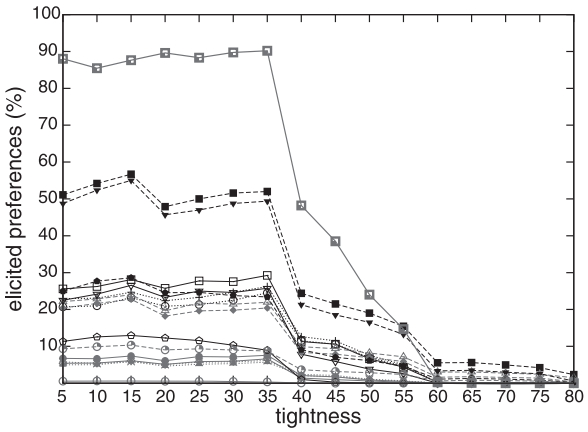
All these algorithms have a useful anytime property, since they can be stopped even before their end obtaining a possibly optimal solution with preference value higher than the solutions considered up to that moment. Fig. 7 shows how fast the various algorithms reach optimality. The y axis represents the solution quality during execution, normalized to allow for comparison among different problems. The algorithms that perform best in terms of elicited preferences, such as



(a) d=50%, t=10%



(b) t=35%, i=30%



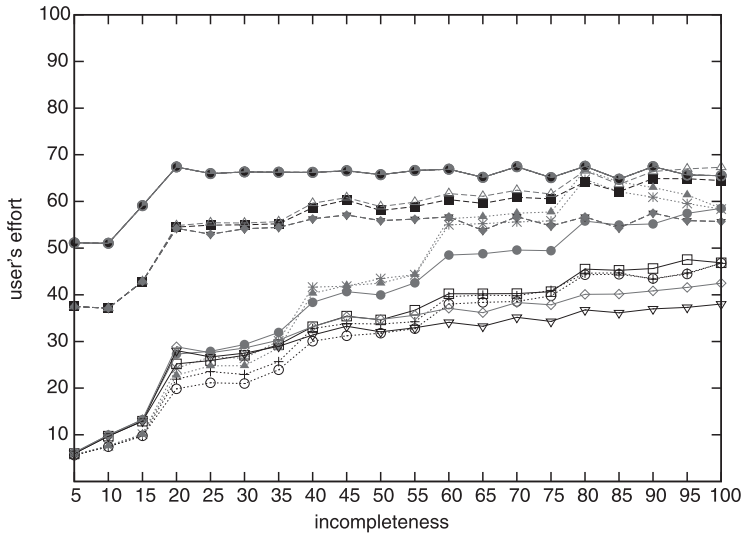
(c) d=50%, i=30%

Fig. 5. Percentage of elicited preferences in incomplete fuzzy CSPs.

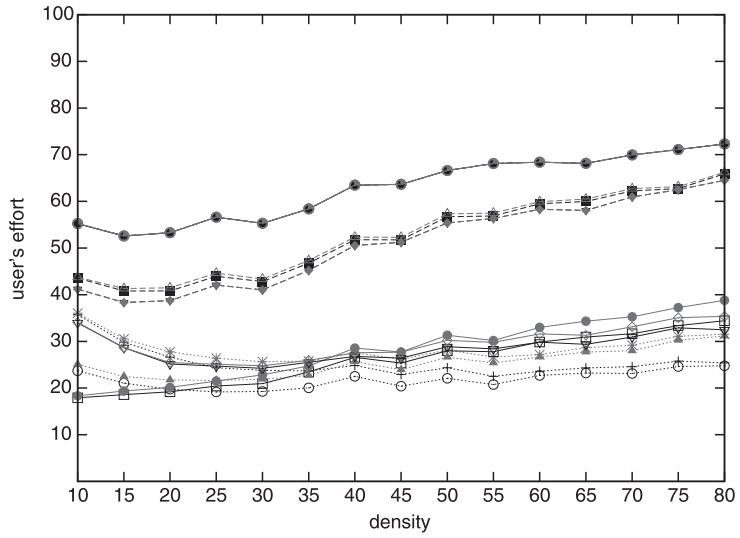
DPI.WORST.BRANCH, are also those that approach optimality fastest. We can therefore stop such algorithms early and still obtain a solution of good quality in all completions.

Fig. 8(a) shows the percentage of elicited preferences (white part of the bar) over all the preferences (white + grey part), as well as the user's effort (black part) for DPI.WORST.BRANCH. Even with high levels of incompleteness, this algorithm elicits only a very small fraction of the preferences, while asking the user to consider at most half of the missing preferences. For example, with incompleteness at 60%, the user effort is at less than 30% and the elicited preferences are at less than 10%.

Fig. 8(b) shows results for LU.WORST.BRANCH, where the user is involved in the choice of the value for the next variable. Compared to DPI.WORST.BRANCH, this algorithm is better both in terms of elicited preferences and user's effort (while SU.WORST.BRANCH is better only for the elicited preferences). We conjecture that the help the user gives in choosing



(a) d=50%, t=10%



(b) t=10%, i=30%

Fig. 6. Incomplete fuzzy CSPs: user's effort.

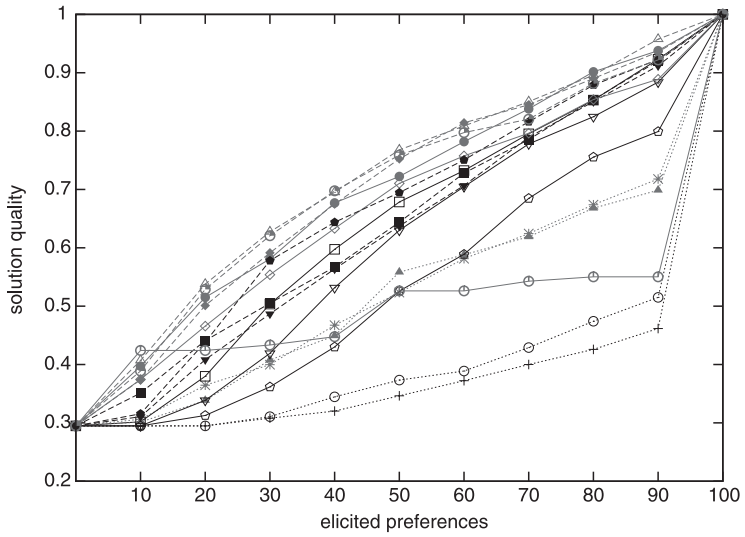
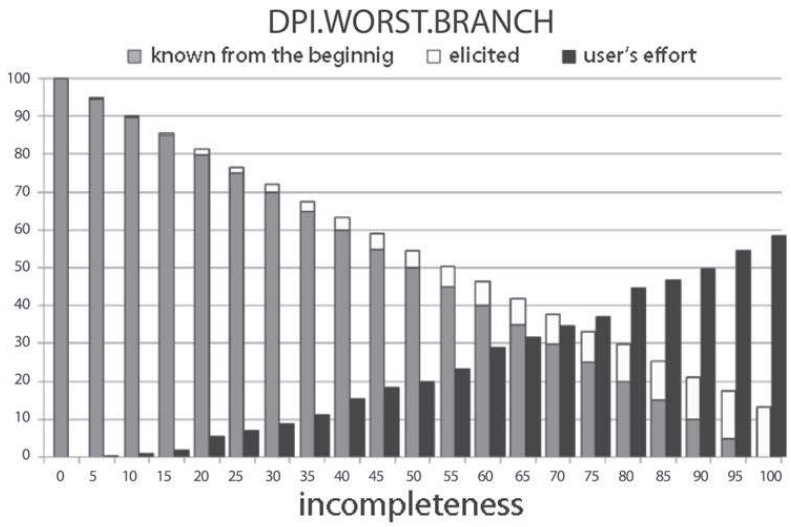
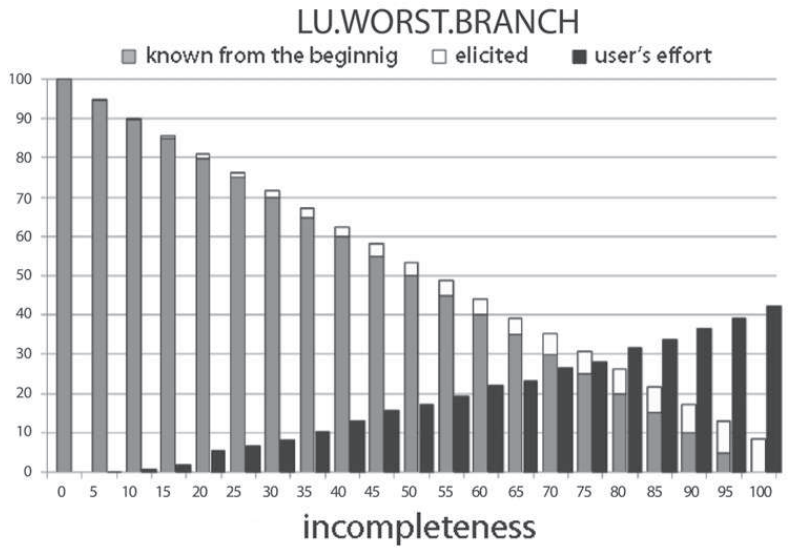


Fig. 7. Incomplete fuzzy CSPs: solution quality.



(a) d=50%, t=10%



(b) d=50%, t=10%

Fig. 8. Incomplete fuzzy CSPs: best algorithms.

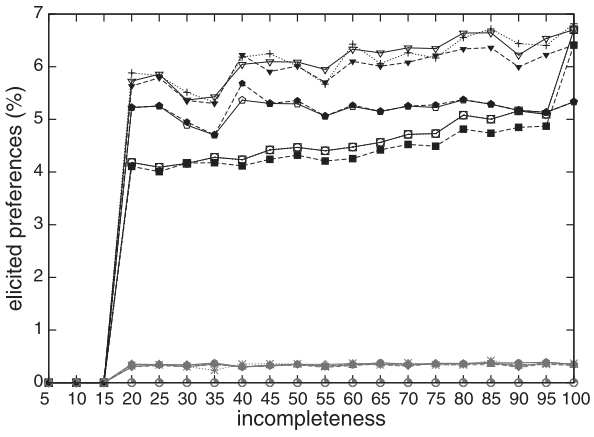
the next value guides the search towards better solutions, thus resulting in an overall decrease of the number of elicited preferences.

7.2. Incomplete CSPs

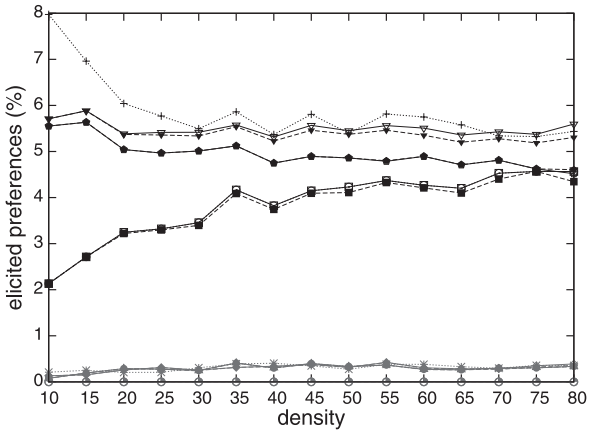
We also tested these algorithms on incomplete hard CSPs. In this case, preferences are only 0 and 1, and necessarily optimal solutions are complete assignments which are feasible in all completions. The problem generator is adapted accordingly. The parameter *what* now has a specific meaning: $what = worst$ means asking if there is a 0 among the considered missing preferences. If there is no 0, we can infer that all the considered missing preferences are 1s.

Fig. 9 shows the percentage of elicited preferences in terms of amount of incompleteness, density, and tightness. Notice that the scale on the y axis varies to include the highest values. The best algorithms are those with $what = worst$, where the inference explained above about missing preferences can be performed. It is easy to see a phase transition at tightness about 35%, which is when problems pass from being solvable to having no solutions. However, the percentage of elicited preferences is below 20% for all algorithms even at the peak.

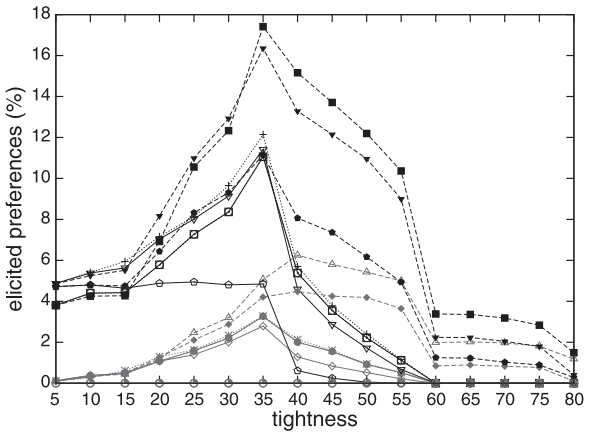
Fig. 10 shows the user's effort in terms of amount of incompleteness and in terms of density. Overall, the best algorithm is again DPI.WORST.BRANCH, whose percentage of elicited preferences and users effort are shown in Fig. 11 in detail. In this figure we also show the percentage of 1s that are inferred by the system (light grey area). It is possible to note that also with the 100% of missing preferences the user's effort is below 22%.



(a) d=50%, t=10%



(b) t=10%, i=30%



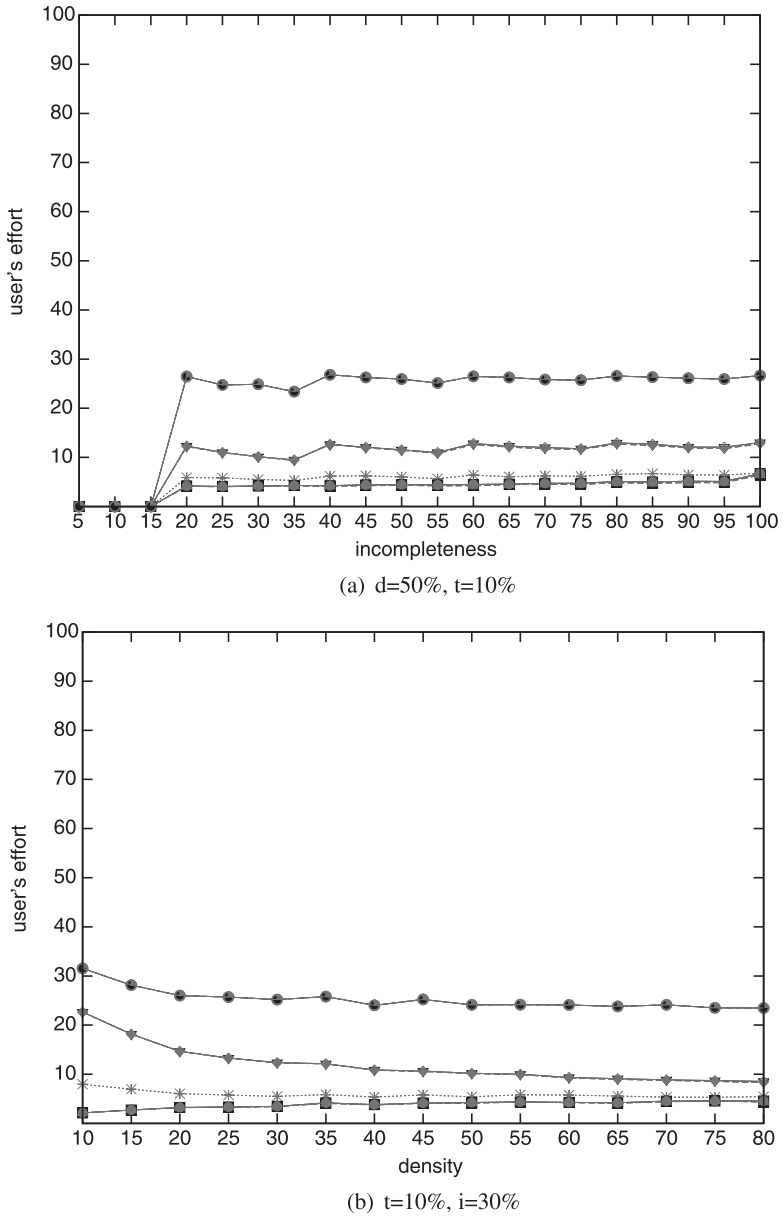
(c) d=50%, i=30%

Fig. 9. Percentage of elicited preferences in incomplete CSPs.

7.3. Incomplete temporal fuzzy CSPs

We also performed some experiments on fuzzy simple temporal problems [15]. In such problems variables represent instantaneous events and constraints model time intervals for durations and distances of such events. Moreover, it is possible to associate a fuzzy preference to each possible duration or distance. Thus, a fuzzy temporal constraint on variables X and Y has the form $\langle [a, b], f \rangle$ where $[a, b]$ is an interval such that $a \leqslant Y - X \leqslant b$ and f is a preference function associating a preference in $[0, 1]$ to each value in $[a, b]$.

Fast consistency-based solvers have been developed for a tractable sub-class of such problems, where all the preference functions are semi-convex [15]. Such solvers are however unable to deal with missing preferences since they make the problems intractable in general. We have thus decided to experiment on this class of problems our branch-and-bound-



(a) d=50%, t=10%

(b) t=10%, i=30%

Fig. 10. Incomplete CSPs: user's effort.

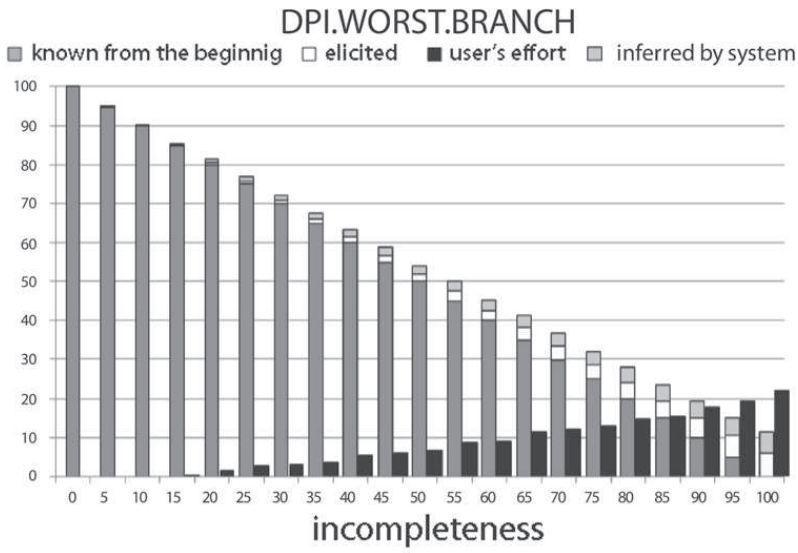
based techniques. In fact, in addition to the value of testing on problems with such a specific structure, the large amount of information required by the specification of such problems makes missing preferences very likely to appear in practice.

We have generated classes of such problems following the approach in [15], adapted to consider incompleteness. Fig. 12 shows that even in this domain it is possible to find a necessarily optimal solution by asking about 10% of the missing preferences, for example via algorithm DPI.WORST.BRANCH.

7.4. Incomplete weighted CSPs

We considered incompleteness also on weighted CSPs (WCSPs). WCSPs model optimization problems where the goal is to minimize the total cost (time, space, number of resources, etc.) of the proposed solution. To handle these problems, we instantiated our general framework using the $\langle A, min, +, +\infty, 0\rangle$ c-semiring. In a similar way as in IFCSPs, in Incomplete Weighted CSPs (IWCSPs) some costs, associated with each tuple, are missing. In this case the multiplicative operator is not idempotent (as in the fuzzy setting) and thus, we have to know all the missing costs associated with a given assignment to obtain its global cost.

To adapt our algorithms to deal with IWCSPs, it is necessary to develop new elicitation strategies optimized for this new environment. We defined three different ways to ask the user for the missing costs. The resulting strategies, as already defined in Section 5, are identified with the following values for the *what* parameter: WW, BB, and BW. Notice that, with WW, we elicit the worst missing cost (that is, the highest) until either all the costs are elicited or the current global cost of



(a) d=50%, t=10%

Fig. 11. Incomplete CSPs: best algorithm.

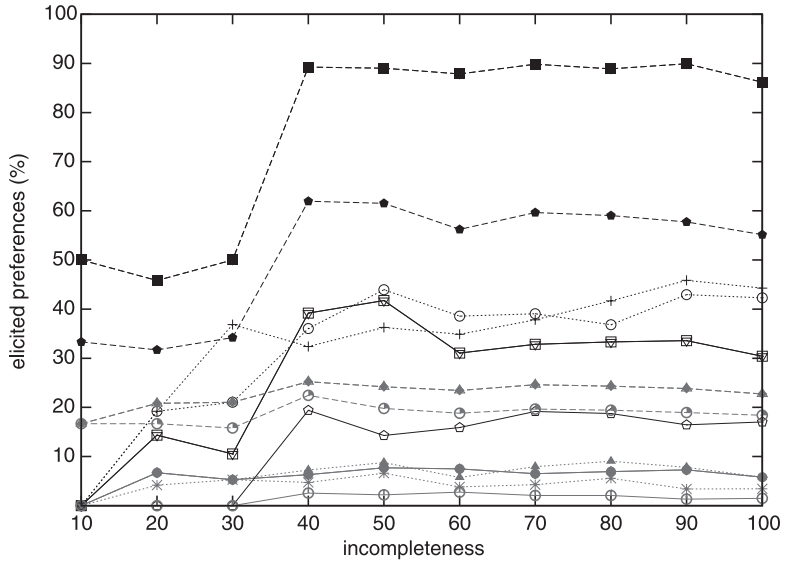


Fig. 12. Percentage of elicited preferences in incomplete fuzzy temporal CSPs.

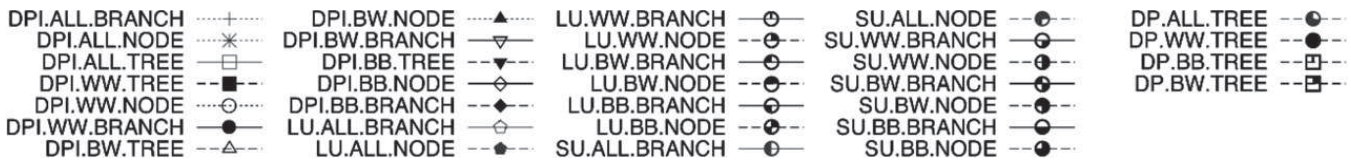
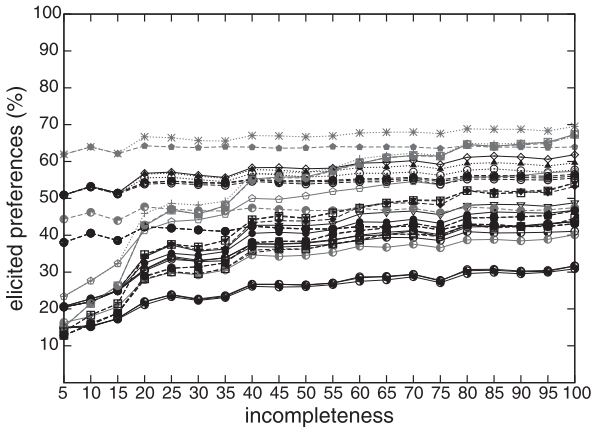


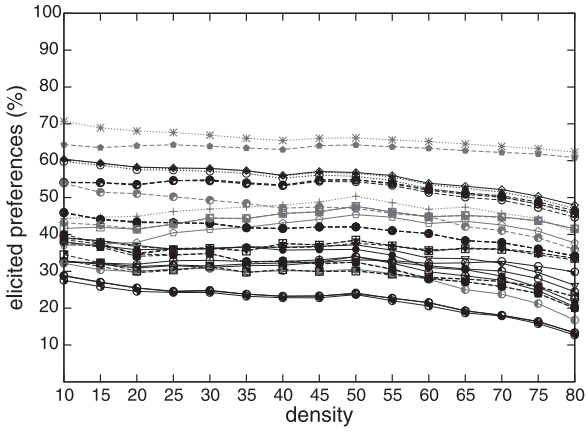
Fig. 13. Algorithms for IWCSPs.

the (possibly partial) assignment is higher than the optimum found so far. By doing this, we know if the global cost exceeds the optimum as early as possible. With BB, we elicit the best (i.e. the minimum) cost until either all the costs are elicited or the current global cost of the (possibly partial) assignment is higher than the optimum found so far. Knowing the best cost of a given assignment allows the system to infer that all the other missing costs are at least as high as the last one elicited. This inference allows us to update the **1**-completion during the search by lowering the upper bound of the missing costs. In this way, we overestimate the real value of the unknown costs. Let P_1^* be the **1**-completion updated with the inferred costs. Given an assignment s, $pref(P_1, s) \geqslant_S pref(P_1^*, s) \geqslant_S pref(P', s)$ where P' is P in which all the incomplete tuples of s are elicited. With BW, we elicit in turn the best and the worst cost. In this case we want to test empirically if the combination of the previous two strategies is better in practice.

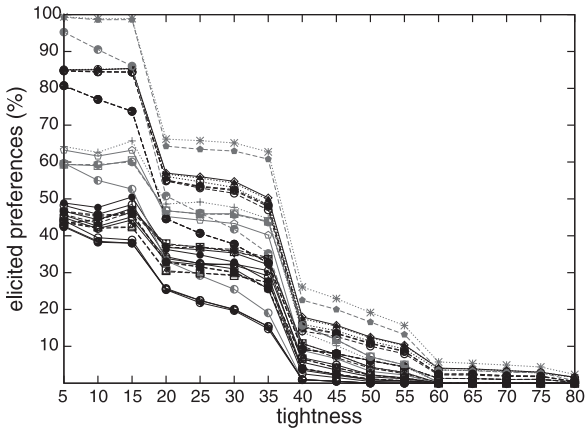
In all the experimental results, the association between an algorithm name and a line symbol is shown in Fig. 13.



(a) d=50%, t=25%



(b) t=25%, i=30%



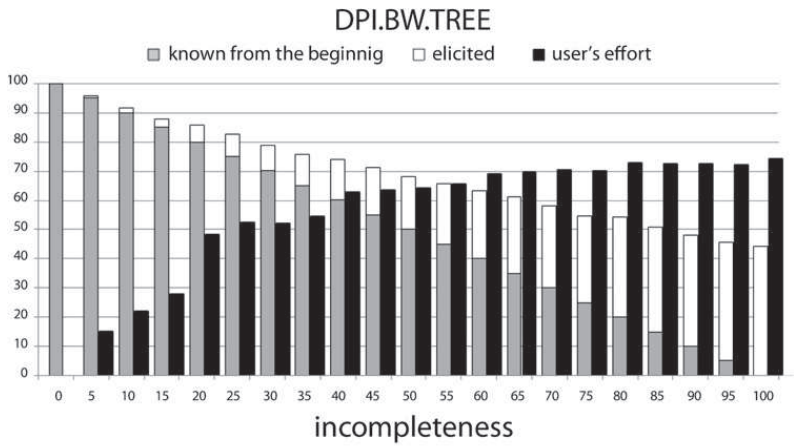
(c) d=50%, i=30%

Fig. 14. Percentage of elicited preferences in incomplete weighted CSPs.

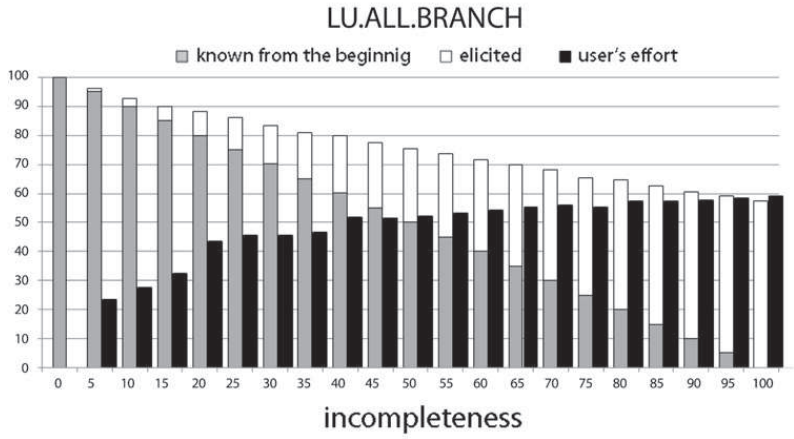
We tested our algorithms on randomly generated IWCSPs, and we used the *what = all* method as a baseline because it is the most general elicitation strategy that can be applied in every possible setting: hard CSPs, fuzzy ISCSPs, etc.

The randomly generated IWCSPs have the same default parameter values as in the IFCSPs experiments, except for the tightness that has a default value of 25%. We choose this value because it is near to the middle of the interval from 0 to 40%, where we will see a phase transition in the percentage of elicited tuples (Fig. 14(c)). We recall that each cost has a value in $[0, 10] \cup \{+\infty\}$.

Fig. 14 shows the percentage of elicited preferences as we vary density, incompleteness and tightness. As expected, the number of elicited tuples increases with the incompleteness of the problem (see Fig. 14(a)). As we vary the tightness (Fig. 14(c)), we can observe a phase transition at $t = 40\%$. At that point, most of the problems have an optimal solution with infinite cost, thus the algorithms do not need more information to find that extreme solution.



(a) t=25% and d=50%



(b) t=25% and d=50%

Fig. 15. Best algorithms for incomplete weighted CSPs.

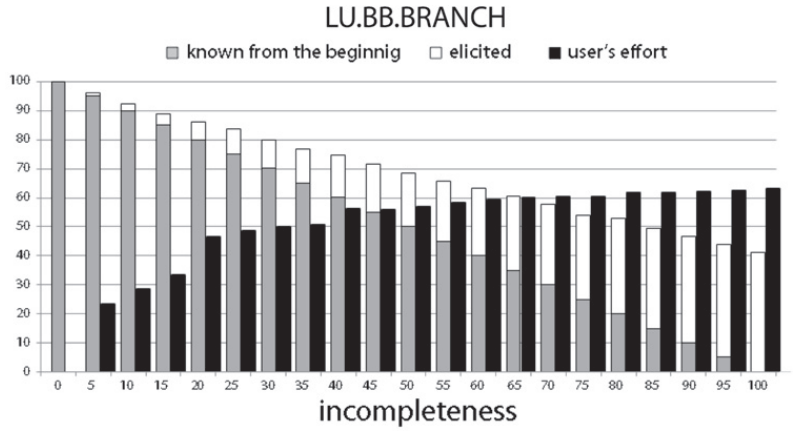
When the density increases (Fig. 14(b)), the percentage of elicited costs tends to decrease slightly. This may be surprising, since, increasing the number of constraints, the number of incomplete tuples increases, thus the percentage of elicited costs should increase. However, the number of infinite costs increases together with the incompleteness. In this particular case, using $t = 25\%$ and $i = 30\%$, the number of infinite cost values increases enough to make the problem easier to solve, thus it requires less elicitation. We performed other experiments decreasing the default tightness value to $t = 10\%$. In this case the percentage of elicited values increases slightly, because there are not enough infinite costs to make the problem easier to solve. On the other hand, the number of incomplete tuples increases with density, making the problem harder to solve (thus requiring more elicitation). Summarizing, increasing density, both incompleteness and tightness increase. When $t = 25\%$ the contribution of the tightness is more important than incompleteness and the problems become easier to solve. On the contrary, when $t = 10\%$, the contribution of the incompleteness is greater and thus the problems become harder.

It easy to see that the best algorithms are those with $when = branch$ and $who = su$. These algorithms ask for only around 30% of the costs needed to solve a totally incomplete problem. The parameter $who = su$ forces the user to select the value to instantiate, which implies an additional effort by the user. This behavior is depicted in Fig. 16(b) where the algorithm elicits a very small number of tuples but the user has to check almost all the incomplete tuples every time.

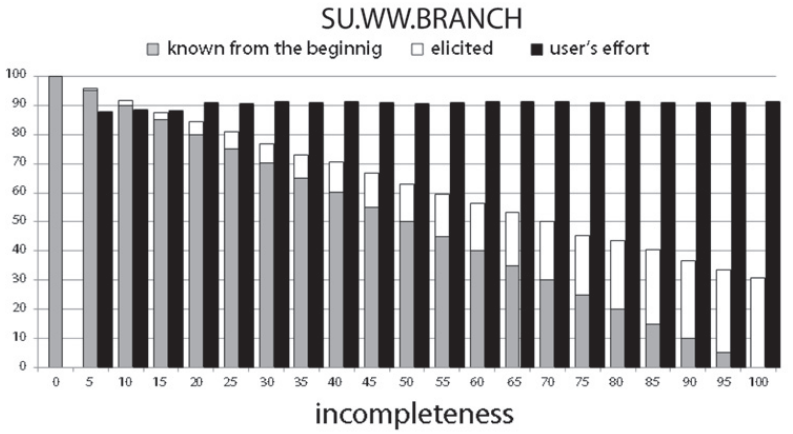
Among the algorithms where the user does not help the system in the value instantiation, the algorithm that elicits less values is DPI.BW.TREE (see Fig. 15(a)).

DPI.BW.TREE elicits less that half of the unknown costs on 100% of incompleteness, requiring the user to look at 70% of incomplete tuples to answer the system's query. All our experiments shown that iteratively asking the user for the best and the worst preference of a given assignment is the best compromise when the value instantiation is totally done by the system. On the other hand, it forces the user to consider more than 70% of the incomplete tuples.

If we want to minimize the user effort, the best choice is the LU.ALL.BRANCH algorithm (see Fig. 15(b)). As shown in Fig. 17(a), the user does less work with LU.ALL.BRANCH than with the other algorithms when the incompleteness is varying. We obtain the same result when the density or the tightness is varying (see Figs. 17(b), 17(c)). If we want a balance between the percentage of elicited costs and the user effort, the best compromise is LU.BB.BRANCH. Fig. 16(a) shows that, on IWCSPs with no initial costs, it elicits 40% of incomplete tuples with a user effort of about 60%.



(a) t=25% and d=50%



(b) t=25% and d=50%

Fig. 16. Best algorithms for incomplete weighted CSPs.

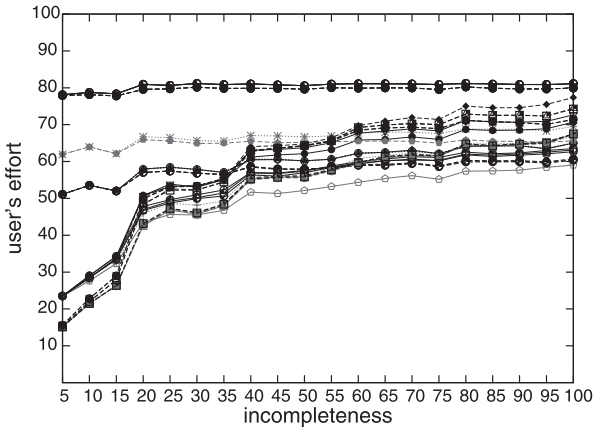
Summarizing, SU.WW.BRANCH (Fig. 16(b)) is the algorithm which elicits less tuples but, if we want the user to just answer the elicitation queries, the best algorithm is DPI.BW.TREE. The algorithm that minimizes the user effort is LU.ALL.BRANCH, whereas the best compromise between user effort and elicitation is LU.BB.BRANCH.

8. Related work

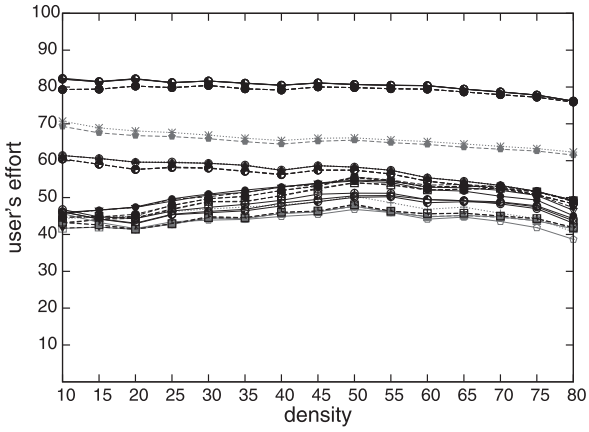
Recently, some lines of work have addressed issues similar to those considered in this paper by allowing for open settings in CSPs: both open CSPs [7,9] and interactive CSPs [16] work with domains that can be partially specified, and in dynamic CSPs [6] variables, domains, and constraints may change over time. It has been shown that these approaches are closely related. In fact, interactive CSPs can be seen as a special case of both dynamic and open CSPs [18].

While the main goal of the work on interactive CSPs is to minimize the run time of the solver, we emphasize the minimization of the number of queries to the user and/or of the user effort. The work closest to ours is the one on open CSPs. An open CSP is a possibly unbounded, partially ordered set $\{CSP(0), CSP(1), \ldots\}$ of constraint satisfaction problems, each of which contains at least one more domain values than its predecessor. Thus, in [9] the approach is to solve larger and larger problems until a solution is found, while minimizing the number of variable values asked to the user. To compare it to our setting, we assume all the variable values are known from the beginning, while some of the preferences may be missing, and our algorithms work on different completions of the given problem. Also, open CSPs exploit a *Monotonicity Assumption* that each agent provides variable values in strictly non-decreasing order of preference. Even when there are no preferences, each agent gives only variable values that are feasible. Working under this assumption means that the agent that provides new values/costs for a variable must know the bounds on the remaining possible costs, since they are provided best value first. If the bound computation is expensive or time consuming, then this is not desirable. This is not needed in our setting, where single preferences are elicited.

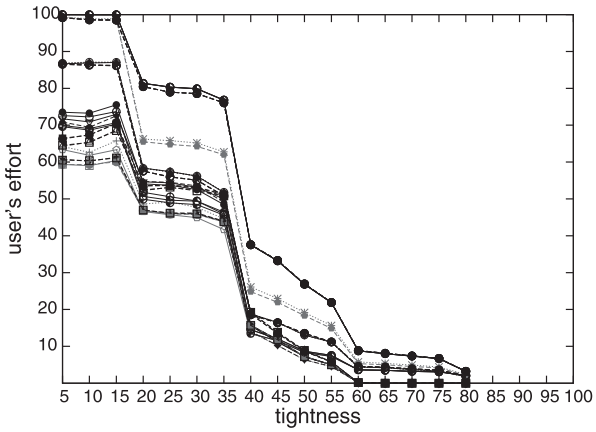
In [7,9,8] the authors develop specific algorithms for open CSPs, and then generalize them to fuzzy and weighted open CSPs. Our approach, on the other hand, went from the general to the specific level: we defined a general framework for incomplete soft constraint problems, and then we instantiated it to specific classes, such as classical CSPs, fuzzy CSPs, and



(a) d=50%, t=25%



(b) t=25%, i=30%



(c) d=50%, i=30%

Fig. 17. User effort in incomplete weighted CSPs.

weighted CSPs. Thus, the general framework is maintained in all these classes, and it can be augmented by the definition of specific optimized elicitation strategies, which may exploit the properties of the c-semiring used.

In addition to the theoretical differences between our approach and the one in [9], we performed an experimental comparison on the same classes of problems considered in [9]. We randomly generated coloring problems with 5–14 variables and 3–11 values per variable, and inequality constraints between randomly chosen variable pairs so that the constraint graph is at least connected and at most complete. We used algorithm LU.WORST.BRANCH on a test set generated in the following way: we randomly generated 1000 coloring problems each with 5–14 variables with 11 values per variable. Then we set the cost of 0 to 8 values per variable to $+\infty$ (in order to have 3–11 feasible values). We also ensured that the constraint graph was at least connected and at most complete. To simulate the totally unknown domains in [9] at the beginning of the search process, we generated variables with no known costs. Finally, we grouped these problems in classes with the same average number of values per domain. In [9] the authors measure the average number of queries per variable. To compare

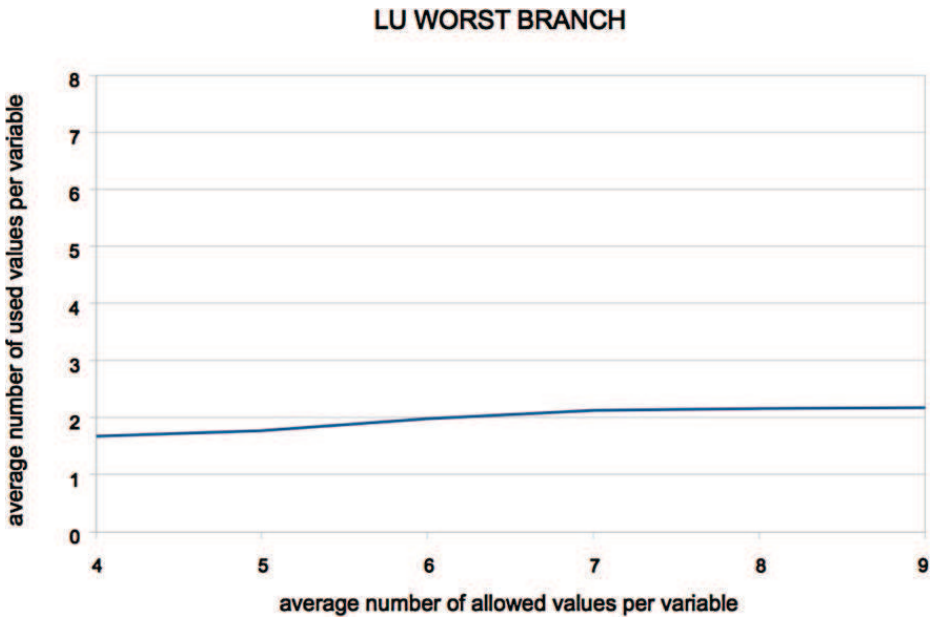


Fig. 18. Average number of values used in incomplete coloring problems with 11 values per variable.

their algorithm with LU.WORST.BRANCH, we measured the average number of different values per variable our algorithm needs to solve the problem (counting 1 every time a value is instantiated for the first time). In Fig. 18 we can see that our algorithm needs about 2 values per variable to solve problems with 4 to 9 values per domain. This is the same result as the best algorithm in [9], which is pleasing as our algorithm is general purpose and not specifically designed for this purpose.

An example of another approach for elicitation in incomplete CSPs is the one presented in [4], where the user provides a classical CSP and a partially unknown utility function over its solutions. The system then performs elicitation queries to select a specific utility function by a regret-based technique, where the elicitation is used to ease the computation of the minimax regret function. In particular, the elicitation concerns bounds on the parameters of the utility function. Moreover, quasi-optimal decisions may be obtained, since often they require much less effort than finding optimal ones. This approach is very interesting but rather far from ours. In particular, our approach does not work with bounds, since we ask for specific preference values. Moreover, our preference functions are based on the preferences in the constraints, and not on the variable values. Also, we want to obtain an optimal solution, although we experimentally analyzed all our algorithms to see whether a good solution is obtained early (see Fig. 7).

9. Conclusion and future work

We have considered incomplete soft constraint problems where some preferences are missing. We have defined a formalism, that extends the soft constraint one, to model such an incompleteness. Two new notions of optimal solutions have been considered: the possibly optimal solutions, that are optimal in at least one way of revealing missing preferences, and the necessarily optimal solutions, that are optimal in all ways of revealing missing preferences, and we have given characterizations of these two sets of optimal solutions. To find necessarily optimal solutions of these problems, we have defined a general solver schema, that interleaves branch and bound (B&B) search with elicitation steps. This solver schema can be instantiated in several ways, that depend on when to elicit, what to elicit, and who elicits. We have tested and compared 16 instances over incomplete fuzzy CSPs and 32 instances over incomplete weighted CSPs, by measuring the percentage of the elicited preferences and the user's effort, that is, the missing preferences that the user has to consider in order to respond to elicitation queries. For fuzzy problems, the percentage of elicited preferences for the best algorithms is below 10%. As expected, weighted problems are more difficult to handle, due to their additive nature: the best solvers need to elicit at most 30% of the missing preferences. In addition, we have considered incomplete CSPs where the soft constraints have been replaced by hard constraints. Experimental results have shown a trend which is similar to the one registered for incomplete fuzzy CSPs. We have also considered problems with a precise structure, such as fuzzy simple temporal problems. In this context, to find a necessarily optimal solution with the best algorithm, it is sufficient to ask about 10% of the missing preferences.

In the problems considered in this paper, we have no information about the missing preferences. We are currently considering settings in which each missing preference is associated with a range of possible values, that may be smaller than the whole range of preference values. For such problems, we intend to define several notions of optimality, among which necessarily and possibly optimal solutions are just two examples, and to develop specific elicitation strategies for each of them. We are also studying soft constraint problems where no preference is missing, but some preferences are unstable, and are associated with a perturbation range of possible alternative values. Moreover, we will consider other

approaches to preference elicitation and measures of the information elicited from the user, such as those in [3,4,19]. We also intend to build solvers based on local search or variable elimination methods. Finally, we want to add elicitation costs and to use them also to guide the search, as done in [25] for hard CSPs.

Acknowledgements

This work has been partially supported by Italian MIUR PRIN project "Constraints and Preferences" (No. 2005015491). The last author is funded by the Department of Broadband, Communications and the Digital Economy, and the Australian Research Council.

References

[1] S. Bistarelli, U. Montanari, F. Rossi, Semiring-based constraint solving and optimization, J. ACM 44 (2) (1997) 201–236.
[2] S. Bistarelli, U. Montanari, F. Rossi, T. Schiex, G. Verfaillie, H. Fargier, Semiring-based CSPs and valued CSPs: Frameworks, properties, and comparison, Constraints 4 (3) (1999) 199–240.
[3] C. Boutilier, A POMDP formulation of preference elicitation problems, in: Proc. AAAI 2002, 2002, pp. 239–246.
[4] C. Boutilier, R. Patrascu, P. Prosser, P. Poupart, D. Schuurmans, Constraint-based optimization and utility elicitation using the minimax decision criterion, Artificial Intelligence 170 (8) (2006) 686–713.
[5] R. Dechter, Constraint Processing, Morgan Kaufmann, 2003.
[6] R. Dechter, A. Dechter, Belief maintenance in dynamic constraint networks, in: Proc. AAAI 1998, 1998, pp. 37–42.
[7] B. Faltings, S. Macho-Gonzalez, Open constraint satisfaction, in: Proc. CP 2002, in: Lecture Notes in Comput. Sci., vol. 2470, Springer, 2002, pp. 356–370.
[8] B. Faltings, S. Macho-Gonzalez, Open constraint optimization, in: Proc. CP 2003, in: Lecture Notes in Comput. Sci., vol. 2833, Springer, 2003, pp. 303–317.
[9] B. Faltings, S. Macho-Gonzalez, Open constraint programming, Artificial Intelligence 161 (1–2) (2005) 181–208.
[10] H. Fargier, J. Lang, Uncertainty in constraint satisfaction problems: A probabilistic approach, in: Proc. ECSQARU 1993, in: Lecture Notes in Comput. Sci., vol. 747, Springer, 1993, pp. 97–104.
[11] H. Fargier, T. Schiex, G. Verfaille, Valued constraint satisfaction problems: Hard and easy problems, in: Proc. IJCAI 1995, Morgan Kaufmann, 1995, pp. 631–637.
[12] M. Gelain, M.S. Pini, F. Rossi, K.B. Venable, Dealing with incomplete preferences in soft constraint problems, in: Proc. CP 2007, in: Lecture Notes in Comput. Sci., vol. 4741, Springer, 2007, pp. 286–300.
[13] M. Gelain, M.S. Pini, F. Rossi, K.B. Venable, Toby Walsh, Elicitation strategies for fuzzy constraint problems with missing preferences: Properties, algorithms and experimental studies, in: Proc. CP 2008, in: Lecture Notes in Comput. Sci., vol. 5202, Springer, 2008, pp. 402–417.
[14] I.P. Gent, E. Macintyre, P. Prosser, B.M. Smith, T. Walsh, Random constraint satisfaction: Flaws and structure, Constraints 6 (4) (2001) 345–372.
[15] L. Khatib, P. Morris, R. Morris, F. Rossi, A. Sperduti, K. Brent Venable, Solving and learning a tractable class of soft temporal problems: Theoretical and experimental results, AI Commun. 20 (3) (2007) 181–209.
[16] E. Lamma, P. Mello, M. Milano, R. Cucchiara, M. Gavanelli, M. Piccardi, Constraint propagation and value acquisition: Why we should do it interactively, in: Proc. IJCAI 1999, 1999, pp. 468–477.
[17] J. Lang, M.S. Pini, F. Rossi, K.B. Venable, T. Walsh, Winner determination in sequential majority voting, in: Proc. IJCAI 2007, 2007, pp. 1372–1377.
[18] S. Macho González, C. Ansótegui, P. Meseguer, On the relation among open, interactive and dynamic CSP, in: Proc. Fifth Workshop on Modelling and Solving Problems with Constraints (IJCAI), 2005.
[19] R.T. Maheswaran, J.P. Pearce, E. Bowring, P. Varakantham, M. Tambe, Privacy loss in distributed constraint reasoning: A quantitative framework for analysis and its applications, Journal of Autonomous Agents and Multiagent Systems (JAAMAS) (2006).
[20] M.S. Pini, F. Rossi, K.B. Venable, T. Walsh, Incompleteness and incomparability in preference aggregation, in: Proc. IJCAI 2007, 2007, pp. 1464–1469.
[21] M.S. Pini, F. Rossi, K.B. Venable, T. Walsh, Computing possible and necessary winners from incomplete partially-ordered preferences, in: Proc. ECAI 2006, 2006, pp. 767–768.
[22] Z. Ruttkay, Fuzzy constraint satisfaction, in: Proc. 1st IEEE Conference on Evolutionary Computing, Orlando, 1994, pp. 542–547.
[23] Toby Walsh, Uncertainty in preference elicitation and aggregation, in: Proc. AAAI 2007, 2007, pp. 3–8.
[24] T. Walsh, Complexity of terminating preference elicitation, in: Proc. AAMAS 2008, 2008.
[25] N. Wilson, D. Grimes, E.C. Freuder, A cost-based model and algorithms for interleaving solving and elicitation of csps, in: Proc. CP 2007, in: Lecture Notes in Comput. Sci., vol. 4741, Springer, 2007, pp. 666–680.