Abstracting Soft Constraints: Some Experimental Results on Fuzzy CSPs

Stefano Bistarelli[1,2], Francesca Rossi[3], and Isabella Pilan[3]

[1] Istituto di Informatica e Telematica, CNR, Pisa, Italy
Stefano.Bistarelli@iit.cnr.it
[2] Dipartimento di Scienze
Universitá degli Studi "G. D'annunzio" di Chieti-Pescara, Italy
bista@sci.unich.it

[3] Dipartimento di Matematica Pura ed Applicata
Università di Padova, Italy
{frossi,ipilan}@math.unipd.it

Abstract. Soft constraints are very flexible and expressive. However, they may also be very complex to handle. For this reason, it may be convenient in several cases to pass to an abstract version of a given soft problem, and then bring some useful information from the abstract problem to the concrete one. This will hopefully make the search for a solution, or for an optimal solution, of the concrete problem, faster.
In this paper we review the main concepts and properties of our abstraction framework for soft constraints, and we show some experimental results of its application to the solution of fuzzy constraints.

1 Introduction

Soft constraints allow to model faithfully many real-life problems, especially those which possess features like preferences, uncertainties, costs, levels of importance, and absence of solutions. Formally, a soft constraint problem (SCSP) is just like a classical constraint problem (CSP), except that each assignment of values to variables in the constraints is associated to an element taken from a set (usually ordered). These elements will then directly represent the desired features, since they can be interpreted, for example, as levels of preference, or costs, or levels of certainty.

SCSPs are more expressive than classical CSPs, but they are also more difficult to process and to solve, mainly because they are optimization rather than satisfaction problems. For this reason, it may be convenient to work on a simplified version of the given problem, trying however to not loose too much information. In [3, 1, 2], an abstraction framework for soft constraints has been proposed, where, from a given SCSP, a new simpler SCSP is generated representing an "abstraction" of the given one. Then, the abstracted version is processed (that is, solved or checked for consistency), and some information gathered during the solving process is brought back to the original problem, in order to transform it into a new (equivalent) problem easier to solve.

All this process has the main aim of finding an optimal solution, or an approximation of it, for the original SCSP, with less time or space w.r.t. a solution process that does not involve the abstraction phase. It is also useful when we don't have any solver for the class of soft problems we are interested in, but we do have it for another class, to which we can abstract to. In this way, we rely on existing solvers to automatically build new solvers.

Many properties of the abstraction framework have been proven in [3]. The most interesting one, which will be used in this paper, is that, given any optimal solution of the abstract problem, we can find upper and lower bounds for an optimal solution for the concrete problem. If we are satisfied with these bounds, we could just take the optimal solution of the abstract problem as a reasonable approximation of an optimal solution for the concrete problem.

In this paper we extend this and other results described in [3] to build a new algorithm to solve the concrete problem by using abstraction steps. More in detail, we prove that using the value of the optimal tuple in the abstract problem as a bound for the abstraction mapping, leads to building each time new abstract problem with better and better corresponding optimal concrete solution. When no more solutions are found, we can be sure that the last found optimum in the abstract problem is indeed the optimal solution of the concrete one.

Besides devising a new solving algorithm based on such results and on the abstraction framework, we also describe the results of several experiments which show the behavior of three variants of such algorithm over fuzzy constraint problems [7, 13, 14]. In these experiments, fuzzy problems are abstracted into classical CSPs, and solved via iterative abstraction of the given problem in different classical CSPs. The behavior of the three versions of our algorithm is then compared to that of Conflex, a solver for fuzzy CSPs. This paper is an extended and improved version of [12].

2 Soft constraints

A soft constraint [5] is just a classical constraint where each instantiation of its variables has an associated value from a partially ordered set. Combining constraints will then have to take into account such additional values, and thus the formalism has also to provide suitable operations for combination ($\times$) and comparison (+) of tuples of values and constraints. This is why this formalization is based on the concept of semiring, which is just a set plus two operations satisfying certain properties: $\langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$.

If we consider the relation $\leq_S$ over A defined as $a \leq_S b$ iff $a + b = b$, then we have that:

- $\leq_S$ is a partial order;
- $+$ and $\times$ are monotone on $\leq_S$;
- $\mathbf{0}$ is its minimum and $\mathbf{1}$ its maximum;
- $\langle A, \leq_S \rangle$ is a complete lattice and $+$ is its lub.

Moreover, if $\times$ is idempotent, then $\langle A, \leq_S \rangle$ is a complete distributive lattice and $\times$ is its glb. Informally, the relation $\leq_S$ gives us a way to compare (some of the) values in the set A. In fact, when we have $a \leq_S b$, we will say that *b is better than a*. Extending the partial order $\leq_S$ among tuples of values, also a partial order among constraints is induced.

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 $con \subseteq V$ and $def : D^{|con|} \rightarrow A$. 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. This element can then be interpreted in several ways: as a level of preference, or as a cost, or as a probability, etc. The correct way to interpret such elements depends on the choice of the semiring operations.

Constraints can be compared by looking at the semiring values associated to the same tuples: Consider two constraints $c_1 = \langle def_1, con\rangle$ and $c_2 = \langle def_2, con\rangle$, with $|con| = k$. Then $c_1 \sqsubseteq_S c_2$ if for all k-tuples t, $def_1(t) \leq_S def_2(t)$. The relation $\sqsubseteq_S$ induces a partial order. In the following we will also use the obvious extension of this relation to sets of constraints, and also to problems (seen as sets of constraints).

Note that a classical CSP is a SCSP where the chosen c-semiring is:

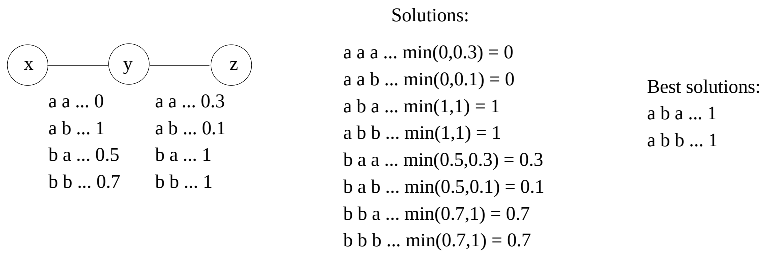
$$S_{CSP} = \langle\{false, true\}, \vee, \wedge, false, true\rangle.$$

Fuzzy CSPs [7, 13, 14], which will be the main subject of this paper, can instead be modeled in the SCSP framework by choosing the c-semiring: $S_{FCSP} = \langle[0,1], max, min, 0, 1\rangle$.

Given two constraints $c_1 = \langle def_1, con_1\rangle$ and $c_2 = \langle def_2, con_2\rangle$, their *combination* $c_1 \otimes c_2$ is the constraint $\langle def, con\rangle$ defined by $con = con_1 \cup con_2$ and $def(t) = def_1(t \downarrow^{con}_{con_1}) \times def_2(t \downarrow^{con}_{con_2})$. In words, combining two constraints means building a new constraint involving all the variables of the original ones, and which associates to each tuple of domain values for such variables a semiring element which is obtained by multiplying the elements associated by the original constraints to the appropriate sub-tuples.

Given a constraint $c = \langle def, con\rangle$ and a subset I of V, the *projection* of c over I, written $c \Downarrow_I$, is the constraint $\langle def', con'\rangle$ where $con' = con \cap I$ and $def'(t') = \sum_{t/t\downarrow^{con}_{I\cap con}=t'} def(t)$. Informally, projecting means eliminating some variables. This is done by associating to each tuple over the remaining variables a semiring element which is the sum of the elements associated by the original constraint to all the extensions of this tuple over the eliminated variables.

The *solution* of a SCSP problem $P = \langle C, con\rangle$ is the constraint $Sol(P) = (\bigotimes C) \Downarrow_{con}$: we combine all constraints, and then project over the variables in *con*. In this way we get the constraint over *con* which is "induced" by the entire SCSP. Optimal solutions are those solutions which have the best semiring element among those associated to solutions. The set of optimal solutions of an SCSP P will be written as $Opt(P)$. In the following, we will sometimes call "a solution" one tuple of domain values for all the problem's variables (over *con*), plus its associated semiring element. Figure 1 shows an example of fuzzy CSP and its solutions.



a a ... 0	a a ... 0.3
a b ... 1	a b ... 0.1
b a ... 0.5	b a ... 1
b b ... 0.7	b b ... 1

Solutions:

a a a ... min(0,0.3) = 0
a a b ... min(0,0.1) = 0
a b a ... min(1,1) = 1
a b b ... min(1,1) = 1
b a a ... min(0.5,0.3) = 0.3
b a b ... min(0.5,0.1) = 0.1
b b a ... min(0.7,1) = 0.7
b b b ... min(0.7,1) = 0.7

Best solutions:
a b a ... 1
a b b ... 1

Fig. 1: A fuzzy CSP and its solutions.

Consider two problems P_1 and P_2. Then $P_1 \sqsubseteq_P P_2$ if $Sol(P_1) \sqsubseteq_S Sol(P_2)$. If $P_1 \sqsubseteq_P P_2$ and $P_2 \sqsubseteq_P P_1$, then they have the same solution, thus we say that they are equivalent and we write $P_1 \equiv P_2$.

SCSP problems can be solved by extending and adapting the technique usually used for classical CSPs. For example, to find the best solution we could employ a branch-and-bound search algorithm (instead of the classical backtracking), and also the successfully used propagation techniques, like arc-consistency [11], can be generalized to be used for SCSPs. The detailed formal definition of propagation algorithms (sometimes called also *local consistency* algorithms) for SCSPs can be found in [5]. For the purpose of this paper, what is important to say is that a *propagation rule* is a function which, taken an SCSP, solves a subproblem of it. It is possible to show that propagation rules are idempotent, monotone, and intensive functions (over the partial order of problems) which do not change the solution set. Given a set of propagation rules, a local consistency algorithm consists of applying them in any order until stability. It is possible to prove that local consistency algorithms defined in this way have the following properties if the multiplicative operation of the semiring is idempotent: equivalence, termination, and uniqueness of the result.

Thus we can notice that the generalization of local consistency from classical CSPs to SCSPs concerns the fact that, instead of deleting values or tuples, obtaining local consistency in SCSPs means changing the semiring values associated to some tuples or domain elements. The change always brings these values towards the worst value of the semiring, that is, the **0**. Thus, it is obvious that, given an SCSP problem P and the problem P' obtained by applying some local consistency algorithm to P, we have $P' \sqsubseteq_S P$.

3 Abstraction

The main idea [6] is to relate the concrete and the abstract scenarios by a pair of functions, the *abstraction* function α and the *concretization* function γ, which form a Galois connection.

Let $(\mathcal{C}, \sqsubseteq)$ and $(\mathcal{A}, \leq)$ be two posets (the concrete and the abstract domain). A Galois connection $\langle \alpha, \gamma \rangle : (C, \sqsubseteq) \rightleftharpoons (A, \leq)$ is a pair of maps $\alpha : \mathcal{C} \to \mathcal{A}$ and $\gamma : \mathcal{A} \to \mathcal{C}$ such that 1. α and γ are monotonic, 2. for each $x \in \mathcal{C}, x \sqsubseteq \gamma(\alpha(x))$ and 3. for each $y \in \mathcal{A}, \alpha(\gamma(y)) \leq y$. Moreover, a Galois insertion (of $\mathcal{A}$ in $\mathcal{C}$) $\langle \alpha, \gamma \rangle : (\mathcal{C}, \sqsubseteq) \rightleftharpoons (\mathcal{A}, \leq)$ is a Galois connection where $\alpha \cdot \gamma$ is the identity over $\mathcal{A}$, that is, $Id_{\mathcal{A}}$.

An example of a Galois insertion can be seen in Figure 2. Here, the concrete lattice is $\langle [0,1], \leq \rangle$, and the abstract one is $\langle \{0,1\}, \leq \rangle$. Function α maps all real numbers in $[0, 0.5]$ into 0, and all other integers (in $(0.5, 1]$) into 1. Function γ maps 0 into 0.5 and 1 into 1.

Consider a Galois insertion from $(\mathcal{C}, \sqsubseteq)$ to $(\mathcal{A}, \leq)$. Then, if $\sqsubseteq$ is a total order, so is $\leq$.

Most of the times it is useful, and required, that the abstract operators show a certain relationship with the corresponding concrete ones. This relationship is called *local correctness*. Let $f : \mathcal{C}^n \to \mathcal{C}$ be an operator over the concrete lattice, and assume that $\tilde{f}$ is its abstract counterpart. Then $\tilde{f}$ is locally correct w.r.t. f if $\forall x_1, \ldots, x_n \in \mathcal{C}, f(x_1, \ldots, x_n) \sqsubseteq \gamma(\tilde{f}(\alpha(x_1), \ldots, \alpha(x_n)))$.

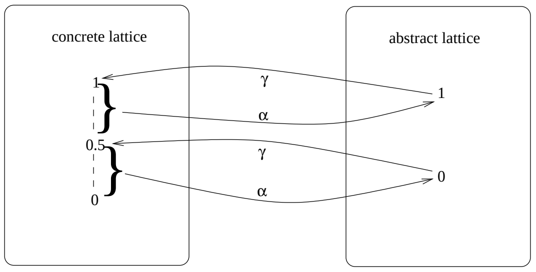


Fig. 2: A Galois insertion.

3.1 Abstracting soft CSPs

The main idea of the abstraction framework presented in [3] is very simple: we just want to pass, via the abstraction, from an SCSP P over a certain semiring S to another SCSP $\tilde{P}$ over the semiring $\tilde{S}$, where the lattices associated to $\tilde{S}$ and S are related by a Galois insertion as shown above.

Consider the *concrete* SCSP problem $P = \langle C, con \rangle$ over semiring S, where

– $S = \langle A, +, \times, 0, 1 \rangle$ and
– $C = \{c_0, \ldots, c_n\}$ with $c_i = \langle con_i, def_i \rangle$ and $def_i : D^{|con_i|} \to A$;

we define an *abstract* SCSP problem $\tilde{P} = \langle \tilde{C}, con \rangle$ over the semiring $\tilde{S}$, where

– $\tilde{S} = \langle \tilde{A}, \tilde{+}, \tilde{\times}, \tilde{0}, \tilde{1} \rangle$;
– $\tilde{C} = \{\tilde{c_0}, \ldots, \tilde{c_n}\}$ with $\tilde{c_i} = \langle con_i, \tilde{def_i} \rangle$ and $\tilde{def_i} : D^{|con_i|} \to \tilde{A}$;
– if $L = \langle A, \leq \rangle$ is the lattice associated to S and $\tilde{L} = \langle \tilde{A}, \tilde{\leq} \rangle$ the lattice associated to $\tilde{S}$, then there is a Galois insertion $\langle \alpha, \gamma \rangle$ such that $\alpha : L \to \tilde{L}$;
– $\tilde{\times}$ is locally correct with respect to $\times$.

Notice that the kind of abstraction we consider in this paper does not change the structure of the SCSP problem. The only thing that is changed is the semiring.

Notice also that, given two problems over two different semirings, there may exist zero, one, or also many abstractions (that is, a Galois insertion between the two semirings) between them. This means that given a concrete problem over S and an abstract semiring $\bar{S}$, there may be several ways to abstract such a problem over $\bar{S}$.

Example 1. As an example, consider any SCSP over the semiring for optimization $\langle \mathcal{R}^- \cup \{-\infty\}, \max, +, -\infty, 0 \rangle$ and suppose we want to abstract it onto the semiring for fuzzy reasoning $\langle [0,1], max, min, 0, 1 \rangle$. In other words, instead of computing the maximum of the sum of all costs (which are negative reals), we just want to compute the maximum of their minimum value, and we want to normalize the costs over $[0,1]$. Notice that the abstract problem has an idempotent $\times$ operator (which is the min). This means that in the abstract framework we can perform local consistency over the problem in order to find inconsistencies.

Example 2. Another example is the abstraction from the fuzzy semiring to the classical one, which will be widely used in the rest of this paper:

$$S_{CSP} = \langle \{0,1\}, \vee, \wedge, 0, 1 \rangle.$$

Here function α maps each element of $[0,1]$ into either 0 or 1. For example, one could map all the elements in $[0,x]$ onto 0, and all those in $(x,1]$ onto 1, for some fixed x. Figure 2 represents this example with $x = 0.5$.

3.2 Properties of the abstraction

We will now summarize the main results about the relationship between a concrete problem and an abstraction of it.

Let us consider the scheme depicted in Figure 3. Here and in the following pictures, the left box contains the lattice of concrete problems, and the right one the lattice of abstract problems. The partial order in each of these lattices is shown via dashed lines. Connections between the two lattices, via the abstraction and concretization functions, is shown via directed arrows. In the following, we will call $S = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ the concrete semiring and $\tilde{S} = \langle \tilde{A}, \tilde{+}, \tilde{\times}, \tilde{\mathbf{0}}, \tilde{\mathbf{1}} \rangle$ the abstract one. Thus we will always consider a Galois insertion $\langle \alpha, \gamma \rangle : \langle A, \leq_S \rangle \rightleftharpoons \langle \tilde{A}, \leq_{\tilde{S}} \rangle$.

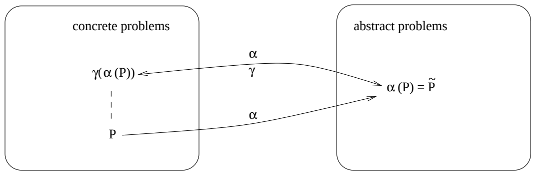


Fig. 3: The concrete and the abstract problem.

In Figure 3, P is the starting SCSP problem. Then with the mapping α we get $\tilde{P} = \alpha(P)$, which is an abstraction of P. By applying the mapping γ to $\tilde{P}$, we get the problem $\gamma(\alpha(P))$. Let us first notice that these two problems (P and $\gamma(\alpha(P))$) are related by a precise property:

$$P \sqsubseteq_S \gamma(\alpha(P)).$$

Notice that this implies that, if a tuple in $\gamma(\alpha(P))$ has semiring value $\mathbf{0}$, then it must have value $\mathbf{0}$ also in P. This holds also for the solutions, whose semiring value is obtained by combining the semiring values of several tuples. Therefore, by passing from P to $\gamma(\alpha(P))$, no new inconsistencies are introduced. However, it is possible that some inconsistencies are forgotten.

Example 3. Consider the abstraction from the fuzzy to the classical semiring, as described in Figure 2. Then, if we call P the fuzzy problem in Figure 1, Figure 4 shows the concrete problem P, the abstract problem $\alpha(P)$, and its concretization $\gamma(\alpha(P))$. It is easy too see that, for each tuple in each constraint, the associated semiring value in P is lower than or equal to that in $\gamma(\alpha(P))$.

If the abstraction preserves the semiring ordering (that is, applying the abstraction function and then combining gives elements which are in the same ordering as the elements obtained by combining only), then the abstraction is called *order-preserving*,

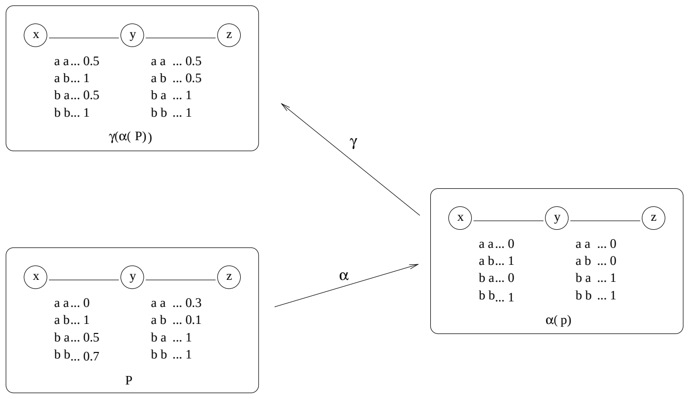


Fig. 4: An example of the abstraction fuzzy-classical.

and in this case there is also an interesting relationship between the set of optimal solutions of P and that of $\alpha(P)$. In fact, if a certain tuple is optimal in P, then this same tuple is also optimal in $\alpha(P)$.

Example 4. Consider again the previous example. The optimal solutions in P are the tuples $\langle a,b,a\rangle$ and $\langle a,b,b\rangle$. It is easy to see that these tuples are also optimal in $\alpha(P)$. In fact, this is a classical constraint problem where the solutions are tuples $\langle a,b,a\rangle$, $\langle a,b,b\rangle$, $\langle b,b,a\rangle$, and $\langle b,b,b\rangle$.

Thus, if we want to find an optimal solution of the concrete problem, we could find all the optimal solutions of the abstract problem, and then use them on the concrete side to find an optimal solution for the concrete problem. Assuming that working on the abstract side is easier than on the concrete side, this method could help us find an optimal solution of the concrete problem by looking at just a subset of tuples in the concrete problem.

Another important property, which holds for any abstraction, concerns computing bounds that approximate an optimal solution of a concrete problem. In fact, any optimal solution, say t, of the abstract problem, say with value $\tilde{v}$, can be used to obtain both an upper and a lower bound of an optimum in P. In fact, we can prove that there is an optimal solution in P with value between $\gamma(\tilde{v})$ and the value of t in P [3, Theorem 29].

Thus, if we think that approximating the optimal value with a value within these two bounds is satisfactory, we can take t as an approximation of an optimal solution of P. Notice that this theorem does not need the order-preserving property in the abstraction, thus any abstraction can exploit this result.

Example 5. Consider again the previous example. Now take any optimal solution of $\alpha(P)$, for example tuple $\langle b,b,b\rangle$. Then the above result states that there exists an optimal solution of P with semiring value v between the value of this tuple in P, which is 0.7, and $\gamma(1) = 1$. In fact, there are optimal solutions with value 1 in P.

However, a better lower bound can be computed in the special case of an abstraction where the semirings are totally ordered and have idempotent multiplicative operations. In this case, any abstraction is order-preserving. In fact, consider an abstraction between totally ordered semirings with idempotent multiplicative operations. Given an SCSP problem P over S, consider an optimal solution of $\alpha(P)$, say t, with semiring value $\tilde{v}$ in $\alpha(P)$. Consider also the set $V = \{v_i \mid \alpha(v_i) = \tilde{v}\}$. Then there exists an optimal solution $\bar{t}$ of P, say with value $\bar{v}$, such that $min(V) \leq \bar{v} \leq max(V)$.

3.3 New Properties

When dealing with the mapping from fuzzy to classical CSPs we can also prove other important results. Consider an abstraction that maps all the semiring values better than the fuzzy value α into 1 (that is, the boolean *true*) and all the fuzzy values worse than or equal than α to 0 (that is, the boolean value $false$). Let us also call P the fuzzy CSP and $\alpha(P)$ the corresponding abstracted CSP. Then we can prove that:

Given an SCSP problem P over the fuzzy semiring, and the corresponding abstract problem $\alpha(P)$ over the boolean semiring, obtained by mapping all values lower than or equal than α to $false$ and all the values bigger than α to $true$.

- if $\alpha(P)$ has no solution, problem P has an optimal solution with an associated semiring fuzzy value worse than or equal than α;
- if P has a solution tuple t with associated semiring level α, and $\alpha(P)$ has no solution, tuple t is an optimal solution for P.

These properties will be very useful in devising the three versions of the abstraction-based algorithm we will define in the next section.

4 Solving by iterative abstraction

The results of the previous section can be the basis for a constraint solving method, or more precisely a family of methods, where abstraction will be used to compute or to approximate the solution of a concrete problem.

Here we will focus on the version of this solving method which applies to fuzzy CSPs, because our experimental results will focus on this class of soft CSPs. The general version of the algorithm is given in [3].

A method to solve a fuzzy CSP is to reduce the fuzzy problem to a sequence of classical (boolean) CSPs to be solved by a classical solver. This method has been for instance recently implemented in the JFSolver [10].

Let us formalize this algorithm within our abstraction framework. We want to abstract a fuzzy CSP $P = \langle C, con \rangle$ into the boolean semiring. Let us consider the abstraction α which maps the values in [0,0.5] to 0 and the values in]0.5,1] to 1, which is depicted in Figure 2. Let us now consider the abstract problem $\tilde{P} = \alpha(P) = \langle \tilde{C}, con \rangle$. There are two possibilities, depending whether $\alpha(P)$ has a solution or not.

1. If $\alpha(P)$ has a solution, then (by the previous results) P has an optimal solution $\bar{t}$ with value $\bar{v}$, such that $0.5 \leq \bar{v} \leq 1$. We can now further cut this interval in two parts, e.g. [0.5,0.75] and]0.75, 1], and consider now the abstraction α' which maps the values in [0,0.75] to 0 and the values in]0.75,1] to 1, which is depicted in Figure 5. If $\alpha'(P)$ has a solution, then P has a corresponding optimal solution with fuzzy value between 0.75 and 1, otherwise the optimal solution has fuzzy value between 0.5

and 0.75, because we know from the previous iteration that the solution is above 0.5. If tighter bounds are needed, one could further iterate this process until the desired precision is reached.

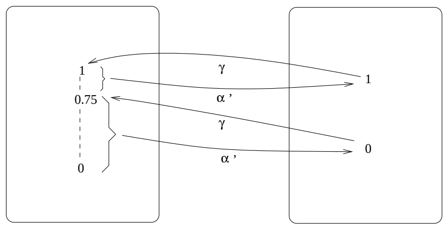


Fig. 5: An abstraction from the fuzzy semiring to the boolean one, cut level = 0.75.

2. If $\alpha(P)$ has no solution, then (by the previous results) P has an optimal solution $\bar{t}$ with value $\bar{v}$, such that $0 \leq \bar{v} \leq 0.5$. We can now further cut this interval in two parts, e.g. [0,0.25] and]0.25,0.5[, and consider now the abstraction α'' which maps the values in [0,0.25] to 0 and the values in]0.25,1] to 1, which is depicted in Figure 6. If $\alpha''(P)$ has a solution, then P has a corresponding optimal solution with fuzzy value between 0.25 and 0.5, otherwise the optimal solution has fuzzy value between 0 and 0.25. And so on and so forth.

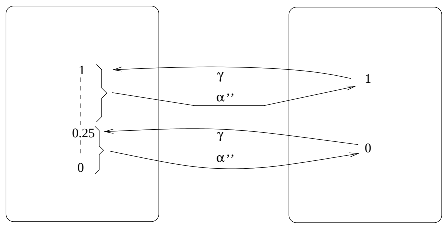


Fig. 6: An abstraction from the fuzzy semiring to the boolean one, cut level = 0.5.

Example 6. For example, consider again the fuzzy constraint problem and the initial abstraction of Figure 4. This abstraction uses a cut level of 0.5 and generates the classical constraint problem in the right part of Figure 4. This problem has solutions (for example, $x = a$, $y = b$, and $z = a$), thus the algorithm above sets the new cut level at 0.75. The new abstracted problem is still soluble: for example, the assignment above is still a solution. Thus the set the new cut level to 0.875. Again, the abstracted problem has solutions (it is actually the same problem as before), so we set the new cut level to 0.9375. The abstracted problem has again solutions. If we have reached the desired precision (assume that we are happy with a tolerance of 0.1, we can conclude the algorithm by reporting the existence of a solution for the initial fuzzy CSPs with value higher than, or equal to 0.9375. More precisely, the iterative algorithm reports that there is a solution in the interval [0.9375,1].

Observe that the dichotomy method used in this example is not the only one that be can used: we can cut each interval not necessarily in the middle but at any point at will (e.g., if some heuristic method allows us to guess in which part of the semiring the optimal solution is). The method will work as well, although the convergence rate, and thus the performance, could be different. In particular we can use the results of the previous section and cut each time at level α corresponding to the value of the tuple t that is optimal in the abstract problem (let's call this heuristic *current best*). We can then continue this procedure until we find no solutions. At this point we are sure that the tuple t found at the previous step is an optimal solution for the concrete problem P.

Example 7. In the example above, this version of the algorithm would work as follows. First, the fuzzy problem is cut at level 0.5, and its abstracted version is the one at the right in Figure 4. This problem has solutions, so we take any solution, for example $x = b$, $y = b$, and $z = b$, and we compute its value in the fuzzy problem. In this case the value is 0.7. By setting the new cut level to 0.7, the new abstracted problem has still solutions (although tuple ¡b,b¿ between x and y has now value 0), so we take any of such solutions, say $x = a$, $y = b$, and $z = b$, and we compute its value in the fuzzy problem, which is 1. Now the new abstracted problem has no solution, so we stop reporting $x = a$, $y = b$, and $z = b$ as an optimal solution for the initial fuzzy problem.

5 Experimental setting and results

Our experimental setting involves a generator of fuzzy SCSPs and the implementation of three versions of the iterative algorithm defined in the previous section. More precisely, the three algorithms we will consider are the following ones:

A1 : Algorithm A1 sets the cut level α to cut the current interval in two equal parts at each abstraction step, as defined in the previous section. It stops when a precision of 1/10 is reached (that is, the size of the considered interval is smaller than or equal to 0.1).

A2 : Algorithm A2 sets the cut level α to the semiring level of the current best solution found. It stops when a precision of 1/10 is reached.

A3 : Algorithm A3 behaves as algorithm A2, except that it stops when no solution in the abstract problem is found. At this point, the last solution found is an optimal solution of the concrete problem.

The generator generates binary fuzzy CSPs with a certain number of variables (n), number of values per variable (m), density (d, which is the percentage of constraints over the maximum possible number) and tightness (that is, percentage of tuples with preference 0, denoted by t). For each set of parameters, we generate three instance problems, and we show the mean result on them.

In all our experiments, which have been performed on a Pentium 3 processor at 850 MHz, we solve concrete fuzzy CSPs via Conflex [7] and, within the abstraction-based algorithms, we solve their abstracted boolean versions via Conflex as well. Since

Conflex is especially designed to solve fuzzy CSPs, it has some overhead when solving classical CSPs. Thus we may imagine that by using a solver for boolean CSPs we may get a better performance for the proposed algorithms.

Conflex solves fuzzy constraint problems by using a branch and bound approach, combined with constraint propagation. Moreover, it allows users to set a threshold (between 0 and 1) which is used to disregard those solutions with values below the threshold (and thus also to perform pruning in the search tree).

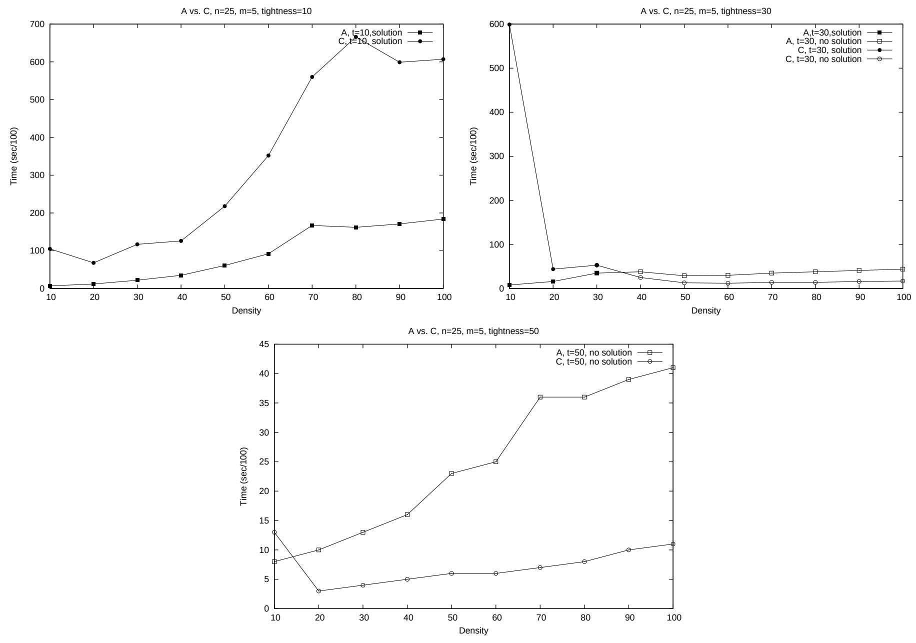


Fig. 7: Time for algorithms A and C, tightness 10%, 30% and 50%.

We start our tests by comparing algorithm A1 to Conflex. Figure 7 shows the time needed to find an optimal solution, or to discover that no solution exists, for both A1 (denoted by A in this pictures since it is the only abstraction-based algorithm) and Conflex (denoted by *C*), with a varying density over the x axis, and varying tightness in the three figures (t=10,30 and 50). The number of variables is fixed to 25, while the domain size is 5. For these experiments, we set the initial threshold 0.01 for algorithm *C* (thus not much pruning is possible initially because of the threshold), and the initial cut level to 0.5. The filled points denote problems with solutions, while the empty points denote problems with no solution.

The graphs show very clearly that, in the presence of solutions, method A1 is better, while Conflex is more convenient when there is no solution. This is predictable: in fact, the iterative algorithm A1, in presence of no solution, would nevertheless continue shrinking the interval until the desired precision.

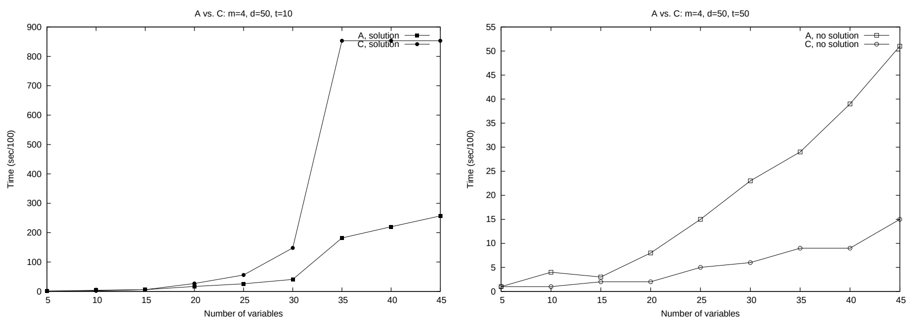


Fig. 8: Time for A and C, varying number of variables, tightness 10% and 50%.

Let us now see what happens when the number of variables varies. The results are shown in Figure 8, where the x axis shows the number of variables. Density is set to 50%, while tightness varies (10% and 50 %). Again, the threshold in *C* is 0.01 and the initial cut level is 0.5. The graphs show again that the abstraction-based method is convenient in solving problems with solutions, while the classical method (that is, Conflex in our experiments) is better when there is no solution.

One could argue that a threshold of 0.01 given to Conflex is a very bad situation for this algorithm, since it cannot perform almost any pruning because of such a low threshold. However, it has to be noticed that, if we give a threshold which is higher than the value of the optimal solution, then Conflex would not find the optimal solutions. Nevertheless, we run some experiments with different thresholds.

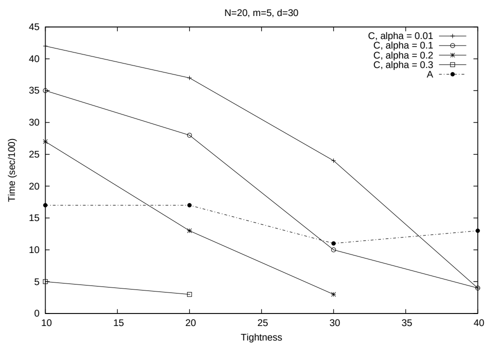


Fig. 9: Time for A and C (with varying threshold), density 30%.

Figure 9 shows the time needed by Conflex with different thresholds (from 0.1 to the first threshold which generates no solution) in different density/tightness scenarios. We can see that algorithm A1 has a balanced performance when the tightness varies, while Conflex, as predictable, behaves better when the threshold is close to the value of the optimal solution (which can be deduced by the first threshold which generates no solution). Therefore, if we can guess a threshold which is close to the optimal solution value, then using *C* is better. However, in absence of this information, we should rather use the abstraction-based method, which gives a good performance in all situations.

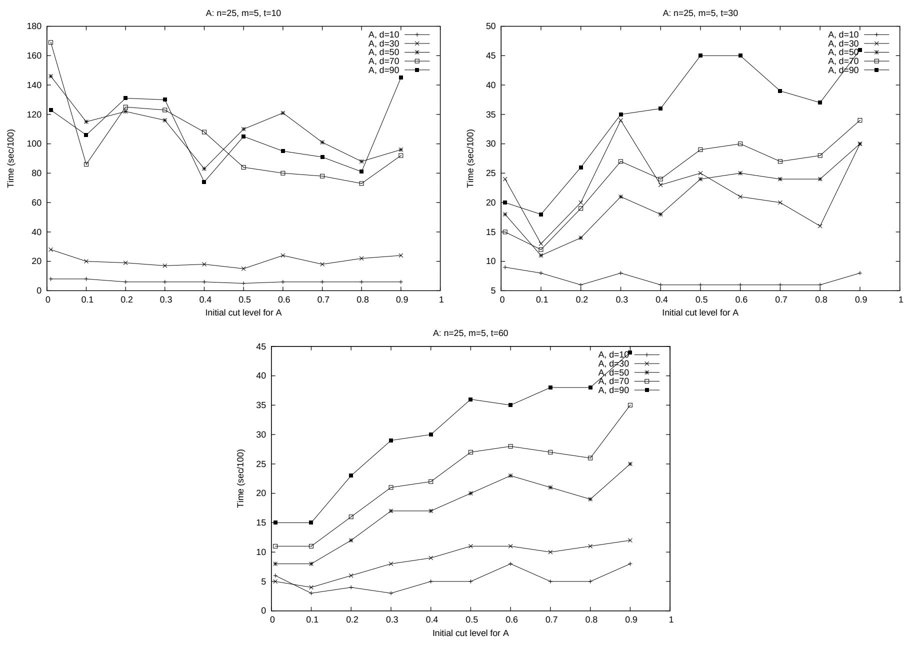


Fig. 10: Time for A, different initial cut levels, tightness 10%, 30%, and 60%.

We may also wonder about the influence of the initial cut level, which up to now has always been 0.5, over the behaviour of the iterative abstraction method. Figures 10 shows the time needed for method A1 when tightness, density, and initial cut level vary. It is easy to see that, with high tightness and density, it is better to set a low initial cut level. This is probably because a high tightness and density usually imply low values for the optimal solutions, thus starting with a high cut level would generate more iterations to get down to the interval containing the optimal solution, if any.

We now pass to consider the other two variants of the original abstraction algorithm: A2 and A3. Figure 11 shows the time needed to find an optimal solution, or to discover that no solution exists, for both the iterative algorithms A1, A2, and A3, and C, with a varying density over the x axis, an tightness t = 10%. The number of variables is fixed to

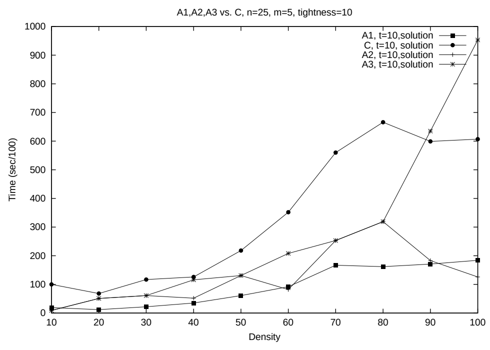


Fig. 11: Time for algorithms A1, A2, A3, and C, tightness 10%.

25 and the domain size is 5. For these experiments, we set as before an initial threshold of 0.01 in C and an initial cut level of 0.5. The graphs show how algorithm A1, A2, and A3 have similar performance, and all of them are better than C when the tightness is not too high. With high tightness, algorithm A3 is worse than C. In fact, the iterative algorithm A3 would spend more time in shrinking the intervals until a precise solution is found.

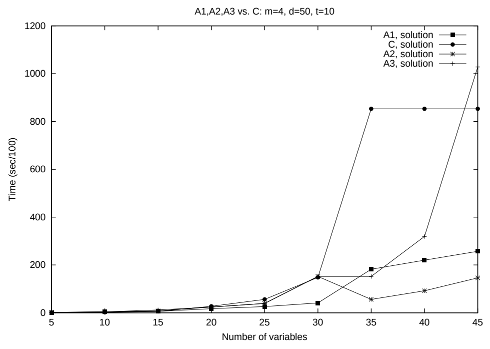


Fig. 12: Time for A1, A2, A3, and C, varying number of variables, tightness 10%.

Let us now compare C to the iterative abstraction algorithms when the number of variables varies (over the x axis). Density is set to 50%, tightness to 10%, and threshold to 0.01. The results are shown in Figure 12. Again, the graphs show that the abstraction-based methods are convenient in solving problems with solutions. We also notice how algorithm A2 is better than A1 when the number of variables increase.

The next experiment shows the comparison of A1, A2, and A3 with C over combinations of densities and tightnesses which generate mostly problems with solutions.

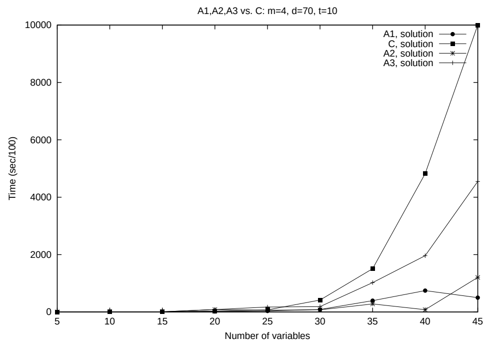


Fig. 13: Time for A1, A2, A3, and C, density 70%, tightness 10%.

Figure 13 shows the results for problems with density 70%. As before, algorithms A1 and A2 are better than C. We recall that algorithm A3 always finds an exact solution, whilst method A1 only stops when the desired precision is reached.

Summarizing, we can learn the following lessons from these last experiments:

- With a small number of variables (25):
  - A2 is more expensive than A1; therefore, it is not worthed to use the value of the abstract solutions to decide the next cut level;
  - A3 is more expensive than A1, but it obtains an optimal solution, not an interval where optimal solutions are contained.
- As the number of variables increases:
  - A2 is less expensive than A1;
  - A3 is more expensive but still convenient w.r.t. C when tightness is not very high.

6 Related Work

Besides Conflex, there exist other systems which allow to solve soft constraints in general and thus also fuzzy constraints.

One is the CLP(FD,S) [9] language, which is a constraint logic programming language where the underlying constraint solver can handle soft constraints. In particular, CLP(FD,S) is a language for modeling and solving semiring-based constraints. The programmer can specify the semiring to be used, thus creating a specific instance of the language, which can handle soft constraints over that semiring. When the semiring to be used has an idempotent combination operator, like for instance for fuzzy CSPs, local propagation techniques are used in the solution process, which is based on a branch and bound approach.

Another one is the Constraint Handling Rules (CHR) [8] system. CHR is a high level formalism to specify constraint solvers and propagation algorithms. It has been originally designed to model and solve classical constraints, but recently [4] it has been extended to handle also semiring-based soft constraints. This extension solves soft constraints by performing local propagation (like node and arc consistency), embedded in one of the two available complete solvers, based on dynamic programming and branch and bound, respectively.

In the experimental work reported in this paper, we did not implement any specific propagation technique or solver: we just solved fuzzy constraint problems by using several times the Conflex solver on classical constraint problems, and we compared this method to using Conflex directly on the given fuzzy problem. For this reason, our results are not directly comparable with [9] or [4].

However, we can try to give an indirect comparison of our results with the CLP(FD,S) system. In fact, in [9] the Conflex system is compared with CLP(FD,S) when solving fuzzy CSPs. The results shown in [9] show that CLP(FD,S) is 3 to 6 times faster than the Conflex system. Our implementation, which is very naive, performs 3 times better than Conflex in average. Thus, we are comparable to CLP(FD,S), which is an optimized system for soft constraints. We plan to implement local propagation techniques during the abstraction steps. We believe that this will burst the performance of our technique, and make it more convenient than CLP(FD,S). We notice however that CLP(FD,S) is unfortunately not maintained any longer, thus it will be difficult to make a fair and complete comparison.

7 Conclusions and future work

We have run several experiments to study the behavior of three versions of an iterative abstraction-based algorithm to solve fuzzy CSPs. The main lesson learnt from these experiments is that, when we work with problems for which we can guess the existence of some solutions, the abstraction methods are more convenient. This holds also when we don't have any information on the value of the optimal solutions. Among the three versions of the algorithm, the first two (A1 and A2) are the best ones. However, since A2 always finds a solution and not an approximation of it, it is to be chosen.

The iterative abstraction methodology we have tested looks promising and suitable to solve fuzzy CSPs. We recall that our experiments used Conflex for solving both the concrete fuzzy problem instances and also the abstract boolean ones. So we may guess that by using a classical boolean CSP solver to solve the abstract version, the abstraction method would result even more convenient.

Our results do not say anything about the convenience of our methodology on other classes of soft constraints. We plan to study the applicability of the abstraction methodology to solve also other classes of soft CSPs. It would be interesting also to study the interaction between the described solving methodology based on abstraction and the notion of global constraints.

References

1. S. Bistarelli, P. Codognet, Y. Georget, and F. Rossi. Abstracting soft constraints. In K. Apt, E. Monfroy, T. Kakas, and F. Rossi, editors, *Proc. 1999 ERCIM/Compulog Net workshop on Constraints*, Springer LNAI 1865, 2000.
2. S. Bistarelli, P. Codognet, and F. Rossi. An Abstraction Framework for Soft Constraints, and Its Relationship with Constraint Propagation. In B. Y. Chouery and T. Walsh, Eds., *Proc. SARA 2000 (Symposium on Abstraction, Reformulation, and Approximation)*, Springer LNAI 1864, 2000.
3. S. Bistarelli, P. Codognet, and F. Rossi. Abstracting Soft Constraints: Framework, Properties, Examples. *AI journal*, 139, 2002.

4. Stefano Bistarelli, Thomas Fruhwirth, Michal Marte and Francesca Rossi. Soft Constraint Propagation and Solving in Constraint Handling Rules. In *Proc. ACM Symposium on Applied Computing, 2002*.
5. S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based Constraint Solving and Optimization. *Journal of the ACM*, 44(2):201–236, March 1997.
6. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Fourth ACM Symp. Principles of Programming Languages*, pages 238–252, 1977.
7. D. Dubois, H. Fargier, and H. Prade. The calculus of fuzzy restrictions as a basis for flexible constraint satisfaction. In *Proc. IEEE International Conference on Fuzzy Systems*, pages 1131–1136. IEEE, 1993.
8. T. Frühwirth, Theory and Practice of Constraint Handling Rules, Special Issue on Constraint Logic Programming (P. J. Stuckey and K. Marriot, Eds.), *Journal of Logic Programming*, Vol 37(1-3):95-138 Oct-Dec 1998.
9. Yan Georget and Philippe Codognet. Compiling Semiring-based Constraints with clp(FD,S). In *Proc. 4th International Conference on Principles and Practice of Constraint Programming (CP98)*, pages 205–219, Springer-Verlag, LNCS 1520, 1998.
10. Ryszard Kowalczyk and Van Anh Bui. JFSolver: A Tool for Solving Fuzzy Constraint Satisfaction. In Hong Yan, Editor, *Proc. FUZZ-IEEE 2001 (10th IEEE International Conference on Fuzzy Systems)*, Melbourne, Australia, IEEE Press 2001.
11. A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.
12. I. Pilan, F. Rossi. Abstracting soft constraints: some experimental results. Proc. ERCIM/Colognet workshop on CLP and constraint solving, Budapest, June 2003.
13. Zs. Ruttkay. Fuzzy constraint satisfaction. In *Proc. 3rd IEEE International Conference on Fuzzy Systems*, pages 1263–1268, 1994.
14. T. Schiex. Possibilistic constraint satisfaction problems, or “how to handle soft constraints?”. In *Proc. 8th Conf. of Uncertainty in AI*, pages 269–275, 1992.