Finding the next solution in constraint- and preference-based knowledge representation formalisms

Ronen Brafman,[1] Francesca Rossi,[2] Domenico Salvagnin,[2] K. Brent Venable[2] and Toby Walsh[3]

Abstract. In constraint or preference reasoning, a typical task is to compute a solution, or an optimal solution. However, when one has already a solution, it may be important to produce the next solution following the given one in a linearization of the solution ordering, where more preferred solutions are ordered first. In this paper, we study the computational complexity of finding the next solution in some common preference-based representation formalisms. We show that this problem is hard in general CSPs, but it can be easy in tree-shaped CSPs and tree-shaped fuzzy CSPs. However, it is difficult in weighted CSPs, even if we restrict the shape of the constraint graph. We also consider CP-nets, showing that the problem is easy in acyclic CP-nets, as well as in constrained acyclic CP-nets where the (soft) constraints are tree-shaped and topologically compatible with the CP-net.

1 Introduction and motivation

In combinatorial satisfaction and optimization problems, the main task is finding a satisfying or optimal solution. There have been many efforts to develop efficient algorithms to perform such tasks, to study the computational complexity of this problem in general, and to find islands of tractability [7]. Another important task is to be able to compare two solutions and to say if one dominates another [2]. In this paper, we address another task that is crucial in many scenarios. When one has already a solution, it can be useful to be able to produce the next solution following the given one in the solution ordering where more preferred solutions are ordered first. If the solution ordering has ties or incomparability, the next solution could be any solution which is tied or incomparable to the given one. In general, however, the next solution is the solution following the given one in a linearization of the solution ordering. The problem of finding the next solution is related to the problem of enumerating all the solutions of a model [5] and to the ranking problem [8], although it is quite different form the latter because we assume only a reference solution in input, and not all previous ones.

In this paper we study the computational complexity of the problem of returning the next solution in some constraint and preference-based formalisms. We show that this is a hard problem in constraint satisfaction problems (CSPs), but it can be easy in tree-shaped CSPs [6] and tree-shaped fuzzy CSPs [10]. However, it is difficult in weighted CSPs, even if we restrict the shape of the constraint graph. Nevertheless, this hardness is only weak, since we give a pseudo-polynomial algorithm to find the next solution in weighted CSPs. Moreover, we also show that this problem is easy in acyclic CP-nets [2], as well as in constrained acyclic CP-nets [3] where the (soft) constraints are tree-shaped and topologically compatible with the CP-net graph.

We intend to look for other tractable cases, and to investigate scenarios where the sufficient conditions for tractability, considered in this paper, naturally hold. We also plan to test experimentally how difficult it is in practice to find the next solution.

Parts of this paper appeared already in [4]. Due to lack of space, some formal proofs are omitted and others are only sketched.

[1] Department of Computer Science Ben Gurion University Beer-Sheva, Israel Email: brafman@cs.bgu.ac.il
[2] Dipartimento di Matematica Pura ed Applicata, University of Padova, Italy, Email: {frossi,salvagni,kvenable}@math.unipd.it
[3] NICTA and UNSW Sydney, Australia Email: Toby.Walsh@nicta.com.au

2 Background

Hard and soft constraints. A soft constraint [10, 1] is a constraint [7] where each instantiation of its variables has an associated value from a (totally or partially ordered) set coming from a c-semiring. A c-semiring is defined by $\langle A, +, \times, 0, 1\rangle$ where A is this set of values, $+$ is a commutative, associative, and idempotent operator, $\times$ is used to combine preference values and is associative, commutative, and distributes over $+$, 0 is the worst element, and 1 is the best element. The c-semiring induces a partial or total order $\leq$ over preference values where $a \leq b$ iff $a + b = b$.

A classical CSP [7] is just a soft CSP where the chosen c-semiring is $S_{CSP} = \langle\{false, true\}, \vee, \wedge, false, true\rangle$. Fuzzy CSPs [10] are instead modeled with $S_{FCSP} = \langle[0, 1], max, min, 0, 1\rangle$. That is, we maximize the minimum preference. For weighted CSPs, the c-semiring is $S_{WCSP} = \langle\mathbb{R}^+, min, +, +\infty, 0\rangle$: preferences are interpreted as costs from 0 to $+\infty$, and we minimize the sum of costs.

Given an assignment s to all the variables of an SCSP P, its preference, written $pref(P, s)$, is obtained by combining the preferences associated by each constraint to the subtuples of s referring to the variables of the constraint. 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. An optimal solution of an SCSP P is then a complete assignment s such that there is no other complete assignment s' with $pref(P, s) <_S pref(P, s')$.

Constraint propagation in classical CSPs reduces variable domains, and thus improves search performance. For some classes of constraints, constraint propagation is enough to solve the problem [6]. This is the case for tree-shaped CSPs, where directional arc-consistency, applied bottom-up on the tree shape of the problem, is enough to make the search for a solution backtrack-free. Given a variable ordering o, a CSP is directional arc-consistent (DAC) if, for any two variables x and y linked by a constraint c_{xy}, such that x precedes y in the ordering o, we have that, for every value a in the domain of

x there is a value b in the domain of y such that (a, b) satisfies c_{xy}.

Constraint propagation can be applied also to soft CSPs, and it maintains the usual properties, as in classical CSPs, if the soft constraint class is based on an idempotent semiring (that is, one where the combination operator is idempotent). This is the case for fuzzy CSPs, for example. As for classical CSPs, DAC is enough to find the optimal solution to a fuzzy CSP when the problem has a tree shape [10].

Fuzzy CSPs can also be solved via a cut-based approach. Given a fuzzy CSP P, an α-cut of P, where α is between 0 and 1, is a classical CSP with the same variables, domains, and constraint topology as the given fuzzy CSP, and where each constraint allows only the tuples that have preference above α in the fuzzy CSP. We will denote such a problem by $cut(P, \alpha)$. The set of solutions of P with preference greater than or equal to α coincides with the set of solutions of $cut(P, \alpha)$.

CP-nets. CP-nets [2] are a graphical model for compactly representing conditional and qualitative preference relations. CP-nets are sets of *ceteris paribus (cp)* preference statements. For instance, the statement *I prefer red wine to white wine if meat is served."* asserts that, given two meals that differ *only* in the kind of wine served *and* both containing meat, the meal with red wine is preferable to the meal with white wine. A CP-net has a set of features $F = \{x_1, \ldots, x_n\}$ with finite domains $\mathcal{D}(x_1), \ldots, \mathcal{D}(x_n)$. For each feature x_i, we are given a set of *parent* features $Pa(x_i)$ that can affect the preferences over the values of x_i. This defines a *dependency graph* in which each node x_i has $Pa(x_i)$ as its immediate predecessors. Given this structural information, the agent explicitly specifies her preference over the values of x_i for *each complete assignment* on $Pa(x_i)$. This preference is assumed to take the form of total or partial order over $\mathcal{D}(x_i)$. An *acyclic* CP-net is one in which the dependency graph is acyclic.

Consider a CP-net whose features are A, B, C, and D, with binary domains containing f and $\overline{f}$ if F is the name of the feature, and with the preference statements as follows: $a \succ \overline{a}$, $b \succ \overline{b}$, $(a \wedge b) \vee (\overline{a} \wedge \overline{b}) : c \succ \overline{c}$, $(a \wedge \overline{b}) \vee (\overline{a} \wedge b) : \overline{c} \succ c$, $c : d \succ \overline{d}$, $\overline{c} : \overline{d} \succ d$. Here, statement $a \succ \overline{a}$ represents the unconditional preference for A=a over A=$\overline{a}$, while statement $c : d \succ \overline{d}$ states that D=d is preferred to D=$\overline{d}$, given that C=c.

The semantics of CP-nets depends on the notion of a worsening flip. A *worsening flip* is a change in the value of a variable to a less preferred value according to the cp statement for that variable. For example, in the CP-net above, passing from $abcd$ to $ab\overline{c}d$ is a worsening flip since c is better than $\overline{c}$ given a and b. One outcome α is *better* than another outcome β (written $\alpha \succ \beta$) iff there is a chain of worsening flips from α to β. This definition induces a preorder over the outcomes, which is a partial order if the CP-net is acyclic.

In general, finding the optimal outcome of a CP-net is NP-hard. However, in acyclic CP-nets, there is only one optimal outcome and this can be found in linear time by sweeping through the CP-net, assigning the most preferred values in the preference tables. For instance, in the CP-net above, we would choose A=a and B=b, then C=c, and then D=d.

3 Solution orderings and linearizations

Each of the constraint or preference-based formalisms recalled in the previous section generate a *solution ordering* over the variable assignments, where solutions dominate non-solutions, and more preferred solutions dominate less preferred ones. This solution ordering can be a total order, a total order with ties, or even a partial order with ties. However, the problem of finding the next solution needs a strict linear order over the variable assignments, thus we will need to consider a linearization of the solution ordering.

CSPs generate a solution ordering which is total order with ties: all the solutions are in a tie (that is, they are equally preferred), and dominate in the ordering all the non-solutions, which again are in a tie. In soft constraints, the solution ordering is in general a partial order with ties: some assignments are equally preferred, others are incomparable, and others dominate each other. If we consider fuzzy or weighted CSPs, there can be no incomparability (since the set of preference values is totally ordered), so again we have a total order with ties, and a solution dominates another one if its preference value is higher. In this context, linearizing the solution ordering just means giving an order over the elements in each tie.

In acyclic CP-nets, the solution ordering is a partial order. In this scenario, any linearization of the solution ordering has to order every pair of incomparable assignments.

In the following, given a problem P and a linearization l of its solution ordering, we will denote with Next(P,s,l) the problem of finding the solution just after s in the linearization l. Note that, while there is only one solution ordering for a problem P, there may be several linearizations of such a solution ordering.

It is not tractable to compute l explicitly, since it has an exponential length and it would mean knowing all the solutions and their relative order. For these reasons, we will assume the linearization is implicitly given to the Next procedure. For example, a lexicographic order on the variable assignments induces a linearization of the solution ordering of a problem, yet it is polynomially describable.

4 Finding the next solution in CSPs

Let P be a CSP with n variables, and let us consider any variable ordering $o = (x_1, \ldots, x_n)$ and any value orderings $o_1, \ldots, o_n$, where o_i is an ordering over the values in the domain of variable x_i. We will denote with O the set of orderings $\{o, o_1, \ldots, o_n\}$. These orderings naturally induce a lexicographical linearization of the solution ordering, that we call $lex(O)$, where, given two variable assignments, say s and s', we write $s \prec_{lex(O)} s'$ (that is, s precedes s') if either s is a solution and s' is not, or s precedes s' in the lexicographic order induced by O (that is, $s = (s_1, \ldots, s_n)$, $s' = (s'_1, \ldots, s'_n)$, and there exists $i \in [1, n]$ such that $s_i \prec_{o_i} s'_i$ and $s_j = s'_j$ for all $j < i$). We will now show that, if the take the linearization given by $lex(O)$, the problem of finding the next solution is NP-hard.

Theorem 1 *Computing Next(P,s,lex(O)), where P is a CSP, s is one of its solutions and O is a set of orderings, is NP-hard.*

The proof is based on a reduction from SAT.

This result can be extended to a wider class of orderings, as the following theorem states.

Theorem 2 *For each polynomially describable total order ω over complete variable assignments such that its top element does not depend on the constraints of the CSP and is polynomially computable, let us consider the linearization of the solution ordering induced by ω, say $l(\omega)$. Then there exists a solution s such that computing Next(p,s,$l(\omega)$), where p is a CSP, is NP-hard.*

5 Next on tree-shaped CSPs

We know that finding an optimal solution becomes easy if we restrict the constraint graph of the problem to have the shape of a tree. It is therefore natural to consider this class to see whether also the Next problem becomes easy under this condition. We will see that this is indeed so: if the CSP is tree-shaped, it can be easy to find the next solution.

In this section we focus on tree-shaped CSPs. However, the same results hold for bounded tree-width. For a tree-shaped CSP with variable set $X = \{x_1, \cdots, x_n\}$, let us consider the linearization $tlex(O)$, which is the same as $lex(O)$ defined in the previous section, with the restriction that the variable ordering o respects the tree shape: each nodes comes before its children. For example, let us consider the tree-shaped CSP shown in Figure 1, and assume that $o = (x_1, x_2, x_3, x_4, x_5)$ and that in all domains $a \prec_{o_i} b \prec_{o_i} c$. The solutions of the CSP are then ordered by $tlex(O)$ as follows: $(a,b,a,b,b) \prec (a,b,a,c,b) \prec (b,a,b,a,a) \prec (b,a,b,a,b) \prec (b,a,b,c,a) \prec (b,a,b,c,b) \prec (b,b,b,b,b) \prec (b,b,b,c,b)$.

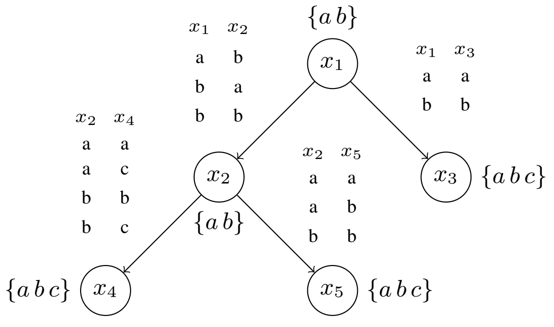

Figure 1. A tree-shaped CSP.

We will now describe an algorithm that, given as input a directionally arc consistent tree-shaped CSP P and a solution s for P, it either returns the consistent assignment following s according to $tlex(O)$, or it detects that s is the last consistent assignment in such an ordering. The algorithm works bottom-up in the tree, looking for new

Algorithm 1: CSP-Next

```
Input: tree-shaped and DAC CSP $P$, orderings $o, o_1, \ldots, o_n$,
assignment $s$
Output: an assignment $s'$, or "no more solutions"
for $i$=$n$ to 1 do
    Search $D(x_i)$ for the next value w.r.t. $o_i$ which is consistent
    with $s_{f(i)}$, say $v'$;
    if $v'$ exists then
        $s_i \leftarrow v'$
        Reset-succ($s$,$i$)
        return $s$
return "no more solutions"
```

variable values that are consistent with the value assigned to their father (denoted by $f(i)$ in Algorithm 1) and successive to the ones assigned in s in the domain orderings. As soon as it finds a variable for which such a value exists, it resets all the following variables (according to the variable ordering o) to their smallest compatible values w.r.t. the domain orderings (via procedure Reset-succ).

For example, if we run CSP-Next giving in input the CSP of Figure 1 and solution s=(b,a,b,a,b), the algorithm first tries to find a value for x_5 consistent with $x_2 = a$ and following b in the domain ordering of x_5. Since no such value exists, it moves to x_4 and performs a similar search, that yields $x_4 = c$. Procedure Reset-succ then sets $x_5 = a$, the first value in the ordering for x_5 consistent with $x_2 = a$.

Theorem 3 *Consider a tree-shaped and DAC CSP P and the ordering $tlex(O)$ defined above. If s is not the last solution in ordering $tlex(O)$, the output of CSP-next(P,s) is the successor of s according to $tlex(O)$; otherwise, the output of CSP-next(P,s) is "no more solutions".*

If $|D|$ is the cardinality of the largest domain, it is easy to see that the worst case complexity of CSP-next is $O(n|D|)$, since both looking for consistent assignments and resetting to the earliest consistent assignment takes $O(|D|)$, and such operations are done $O(n)$ times.

From Theorem 3 we can thus conclude that Next(P,s,$tlex(O)$) is polynomial, since it can be computed by applying DAC to P and then CSP-Next to P and s, both of which are polynomial-time algorithms.

Note also that the choice of the linearization is crucial for the complexity of the algorithm. Indeed, a different choice for l may turn Next(P,s,l) into an NP-hard problem, even on tree-shaped CSP, as proved in the following theorem.

Theorem 4 *Computing Next(P,s,l), where P is a tree-shaped CSP, s is one of its solutions, and l is an arbitrary linearization, is NP-hard.*

The proof is based on a reduction from the subset sum problem.

6 Next on weighted CSPs

Theorem 5 *Computing Next(P,s,l), where P is a weighted CSP and s is one of its solutions, is NP-hard, for* any *linearization l.*

The proof of Theorem 4 can be easily adapted for the purpose of this statement.

Note that theorems 4 and 5, while very similar in proof, have quite a different implication. Indeed, while for tree-shaped CSPs computing Next is NP-hard only for some choices of the linearization l, for weighted CSPs computing Next is *always* NP-hard, irrespective of the linearization.

However, we will now show that, if we consider the lexicographical ordering, then this NP-hardness is weak, since we can provide a pseudo-polynomial algorithm to find the next solution in weighted CSPs. Thus, it is only the possibility to use large utilities that makes the problem intractable in general.

In the context of a weighted CSP, finding the next solution means that, given a solution, we want to return the next assignment in lexicographical order with the same utility or, if there is no such assignment, the first assignment in lexicographical order with the next smaller utility.

Theorem 6 *It is weakly NP-hard to compute the next solution in a weighted CSP.*

To show that finding the next solution is only weakly NP-hard, we give a pseudo-polynomial algorithm which is a simple generalization of the dynamic programming algorithm for deciding subset sum [9, 11]. Given a weighted CSP with no constraints, and an assignment $a = (a_1, \ldots, a_n)$ to its variables x_i with $1 \leq i \leq n$, with utility $U(a) = s$, we define $Q_a(j,t)$ as the lexicographically smaller assignment $b = (b_1, \ldots, b_j)$ with utility t, that involves only the first j variables, and such that $b \succ_{lex} a$. If no such assignment exists, then $Q_a(j,t) = nil$. A simple recursion can be used to compute $Q_a(j,t)$, for any j from 1 to n, and any utility level t from 0 to s. In particular, we can initialize $Q_a(1,t)$ as $Q_a(1,t) = \text{lexmin}_{b_1 \in D_1}\{(b_1) : u_1(b_1) = t \ \wedge \ b_1 \succ_{lex} a_1\}$ and then compute recursively $Q_a(j,t) = \text{lexmin}_{b_j \in D_j}\Big\{\{(Q_a(j-1, t-v_j), b_j) :$

$(b_1,\dots,b_j) \succ_{lex} (a_1,\dots,a_j)\}\cup\{(a_1,\dots,a_{j-1},b_j) : u_j(b_j) = u_j(a_j) \wedge b_j \succ_{lex} a_j\}\Big\}$ where $v_j = u_j(b_j)$. To compute the next solution given an assignment a with utility s, we just need to check whether $Q_a(n,s)$ is not *nil*. If it is, $Q_a(n,s)$ contains the lexicographically next assignment with utility s. Otherwise, we use a very similar dynamic program to compute the lexicographically smallest assignment with the next smallest utility. The running time of both dynamic programs is $O(n\cdot s\cdot|D|)$, where $|D|$ is the size of largest domain. Thus we can compute the next solution in pseudo-polynomial time.

This algorithm works also when the weighted CSP problem has some constraints, provided that there is a polynomial number of such constraints and each has a bounded scope. In such weighted CSPs, we compute $Q_a(j,t)$ from $Q_a(j-1,t-u_i)$ where u_i is the additional utility added by assigning $x_j = b_j$. This is polynomial to compute given our assumption that there is a polynomial number of constraints, each involving a polynomial number of variables. Hence, there is a pseudo-polynomial time algorithm to compute the next solution also for any such weighted CSP.

Example: To make it simple, we consider a weighted CSP with no constraints and five Boolean variables x_1 to x_5 where $u_i(0) = 0$ and $u_i(1) = i$. Suppose we are given the solution $a = (0,1,1,0,0)$ of utility $s = 5$ and we want to compute the lexicographically next solution. The dynamic programming algorithm will build from the bottom-left corner the following table:

t	x_1	x_2	x_3	x_4	x_5
5	*nil*	*nil*	*nil*	(1,0,0,1)	(1,0,0,1,0)
4	*nil*	*nil*	(1,0,1)	(1,0,1,0)	(1,0,1,0,0)
3	*nil*	(1,1)	(1,1,0)	(1,1,0,0)	(1,1,0,0,0)
2	*nil*	*nil*	*nil*	*nil*	*nil*
1	(1)	(1,0)	(1,0,0)	(1,0,0,0)	(1,0,0,0,0)
0	*nil*	*nil*	*nil*	*nil*	*nil*

For example, $Q_a(j=3,t=4) = (1,0,1)$. This means that $(1,0,1)$ is the lexicographically smaller assignment with utility 4, that involves only the first 3 variables, and such that it follows $(0,1,1)$ (that is, the restriction of a to the first 3 variables) lexicographically. Since $Q_a(j=5,t=5)$ is not *nil*, then the lexicographically next assignment with utility 5 is $(1,0,0,1,0)$.

7 Next on tree-shaped fuzzy CSPs

Turning our attention to fuzzy CSPs, we will show that Next on tree-like fuzzy CSPs can be easy. Let P be a fuzzy tree-shaped CSP with variable set $X = \{x_1,\dots,x_n\}$ and set of constraints C, and let us consider a variable ordering $o = \{x_1,\dots,x_n\}$ which respects the tree shape. Moreover, let o_i be a total order over the values in the domain of x_i, for $i = 1,\dots,n$.

We will consider set $T = \{t = (x_i = v_i, x_j = v_j) | i < j, \exists c \in C, t \in c, pref_c(t) > 0\}$, where $pref_c(t)$ denotes the preference assigned to t by constraint c that is, the set of all pairs of variables assignments appearing in P with preference greater than 0. The preferences assigned to tuples by the constraints in C and the orderings $o, o_i, \cdots, o_n$ induce the following ordering o_T over T: $(x_i = v, x_j = w) \prec_{o_T} (x_h = z, x_k = u)$ if

- the preference associated to tuple $(x_i = v, x_j = w)$ by its constraint is higher than the preference associated to tuple $(x_h = z, x_k = u)$ by its constraint, or
- they have the same preference, and the variable pair (x_i, x_j) lexicographically precedes the variable pair (x_h, x_k) according to o, or
- they have the same preference, $i = h$, $j = k$ and the value pair (v,w) lexicographically precedes the value pair (z,u) according to domain orderings o_i and o_j.

We will now use this strict total order over the set of tuples of P to define a strict total order over the set of solutions of P. Given two complete assignments to X, say s and s', let $t_s = min_{o_T}\{t$ tuple of s with preference $pref(s)\}$ and $t'_s = min_{o_T}\{t$ tuple of s' with preference $pref(s')\}$. Let $opt(P)$ denote the optimal preference of a fuzzy CSP P. We write $s \prec_f s'$ (that is, s precedes s' in ordering $\prec_f$), if

- $pref(s) > pref(s')$, or
- $pref(s) = pref(s') = opt(P)$ and s precedes s' in the lexicographic order induced by o and the domain orderings $o_1,\dots,o_n$, or
- $pref(s) = pref(s') < opt(P)$ and $t_s \prec_{o_T} t'_s$,
- $pref(s) = pref(s') < opt$, $t_s = t'_s$ and s precedes s' in the lexicographic order induced by o and the domain orderings $o_1,\dots,o_n$.

It is possible to show that $\prec_f$ is a linearization of the solution ordering.

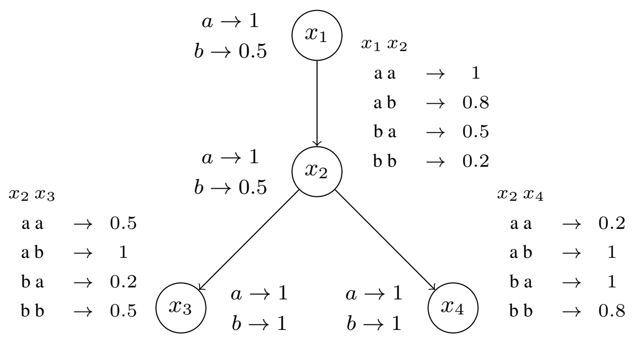

Figure 2. A tree-shaped DAC fuzzy CSP.

For example, let us consider the tree-shaped DAC Fuzzy CSP shown in Figure 2. Assume that $o = (x_1,x_2,x_3,x_4)$ and that $a \prec_{o_i} b$ for $i = 1,\dots,4$. Then, if we consider solutions $s = (b,a,a,b)$ and $s' = (a,b,b,b)$, we have that $s \prec_f s'$ since $pref(s) = pref(s') = 0.5 < 1 = opt(P)$, $t_s = (x_1 = b, x_2 = a)$, $t_{s'} = (x_2 = b, x_3 = b)$, and thus $t_s \prec_{o_T} t_{s'}$; If instead we consider solutions $s = (b,b,a,a)$ and $s' = (b,b,a,b)$, we have again that $s \prec_f s'$, since $pref(s) = pref(s') = 0.2 < 1 = opt(P)$, $t_s = (x_1 = b, x_2 = b) = t'_s$, and s precedes s' lexicographically.

As with CSPs, we provide a polynomial time algorithm that solves the Next problem for tree-shaped fuzzy CSPs. The main idea that we exploit is that, in a fuzzy CSP, a solution can have preference p only if it includes a tuple that has preference p. In Algorithm 2:

- procedure *fix(P,t)* takes in input a fuzzy CSP P and one of its tuples, $t = (x_i = v, x_j = w)$ and returns the fuzzy CSP obtained from P by removing from the domains of variables x_i and x_j all values except v and w;
- procedure *cut(P,p)* takes in input a fuzzy CSP P and a preference p and returns the CSP corresponding to the $p-cut$ of P as defined in Section 2;
- procedure *Solve(P)* takes in input a CSP P and returns the first solution in a lexicographic order given the variable and the domain orderings.

Algorithm 2: FuzzyCSP-Next

```
Input: tree-shaped and DAC Fuzzy CSP $P$, orderings
$o, o_1, \ldots, o_n, o_T$, assignment $s$ with preference $p$
Output: an assignment $s'$, or "no more solutions"
if $p = opt(P)$ then
    $P' \leftarrow cut(P, p)$
    if CSP-next(P',s) $\neq$ "no more solutions" then
        return CSP-next(P',s)
if $p \neq opt(P)$ then
    compute tuple $t_s$
    $t^* = t_s$
else
    let $t^*$ be the first tuple s.t. $pref(t^*) = next(p)$
$p^* = pref(t^*)$
$P' \leftarrow cut((fix(P, t^*)), p^*)$
if CSP-next(P',s) $\neq$ "no more solutions" then
    return CSP-next(P',s)
$pref(t) \leftarrow 0, \forall t \in T$ such that $pref(t) = p^*$ and $t \leq_{o_T} t^*$
$cpref \leftarrow p^*$
for each tuple $t >_{o_T} t^*$ following order $o_T$ until $pref(t) > 0$
do
    if $pref(t) < cpref$ then
        reset all preferences, previously set to 0, to their original
        values
    if pref( Solve ( cut(fix(P,t),pref(t)))) = pref(t) then
        return Solve( cut(fix(P,t),pref(t)))
    $cpref \leftarrow pref(t)$
    $pref(t) \leftarrow 0$
return "no more solutions"
```

Intuitively, when solution s with preference p is given in input, if s is optimal, we look for the next solution in the CSP obtained from P by performing a cut at level p and running CSP-next. If no solution is returned, then s must have been the last solution with optimal preference in the ordering and its successor must be sought for at lower preference levels.

If s is not optimal, we consider its tuples and we identify the smallest tuple of s, say t_s, according to ordering o_T, that has preference p in the corresponding constraint. We fix such a tuple, via $fix(P, t_s)$, and we cut the obtained fuzzy CSP at level p. We then look for the solution lexicographically following s in such a CSP by calling CSP-next. If no such solution exists, s must be the last solution with preference p among those that get their preference from t_s.

The next solution may have preference p or lower. However, if it does have preference p, such a preference must come from a tuple with preference p which follows t_s in the ordering o_T. In order to avoid finding solutions with preference equal to p that come from tuples with preference p preceding t_s according to o_T, we set the preference of all such tuples to 0. If none of the tuples with preference p following t_s generate solutions with preference p, we move down one preference level, restoring all modified preference values to their original values. This search continues until a solution is found or all tuples with preference greater than 0 have been considered.

Theorem 7 *Given a tree-shaped DAC fuzzy CSP P and a solution s, algorithm FuzzyCSP-Next computes the successor of s according to $\prec_f$ if s is not the last solution with preference greater than 0, and outputs "no more solutions" otherwise. The worst case time complexity of algorithm FuzzyCSP-Next is $O(|T||D|n)$, where $|T|$ is the number of tuples of P and $|D|$ the cardinality of the largest domain.*

The correctness of FuzzyCSP-Next follows directly from the description of the algorithm. For the complexity, we notice that the complexity of FuzzyCSP-Next is bounded by that of running $|T|$ times the CSP-next algorithm. $\square$

Given a tree-shaped fuzzy CSP P, one of its solutions s, and the solution ordering $\prec_f$, $Next(P, s, \prec_f)$ can be therefore computed in polynomial time: we just need to achieve DAC (which is polynomial) and then run algorithm FuzzyCSP-Next. Again, it is not difficult to prove that the choice of the order is crucial for the complexity of the algorithm, and that Next(P,s,l) is in general NP-hard even on tree-shaped fuzzy CSPs. Indeed, since tree-shaped fuzzy CSPs admit tree-shaped CSPs as a special case, the result is a direct consequence of Theorem 4.

8 Next on acyclic CP-nets

We now consider the complexity of the Next operation in acyclic CP-nets. It turns out that Next is easy on such CP-nets, if we consider a certain linearization of the solution ordering. We first define the concept of *contextual lexicographical linearization* of the solution ordering. Let us consider any ordering of the variables where, for any variable, its parents are preceding it in the ordering. Let us also consider an arbitrary total ordering of the elements in the variable domains. For sake of simplicity, let us consider Boolean domains. Given an acyclic CP-net with n variables, we can associate a Boolean vector of length n to each complete assignment, where element in position i corresponds to variable i (in the variable ordering), and it is a 0 if this variable has its most preferred value, given the values of the parents, and 1 otherwise. Therefore, for example, the optimal solution will correspond to a vector of n zeros.

To compute such a vector from a complete assignment, we just need to read the variable values in the variable ordering, and for each variable we need to check if its value is the most preferred or not, considering the assignment of its parents. This is polynomial if the number of parents of all variables is bounded. Given a vector, it is also easy to compute the corresponding assignment.

Let us now consider a linearization of the ordering of the solutions where incomparability is linearized by a lexicographical ordering over the vectors associated to the assignments. We will call such a linearization a *contextual lexicographical linearization*. Note that there is at least one of such linearizations for every acyclic CP-net.

Theorem 8 *Computing Next(N,s,l), where N is an acyclic CP-net, s is one of its solutions, and l is any contextual lexicographical linearization of its solution ordering, is in P.*

Given any solution s and its associated vector, as defined above, the vector of the next solution in l can be easily obtained by a standard Boolean vector increment operation. Therefore, given any solution s, it is possible to obtain the next solution by 1) computing the vector associated to s, 2) incrementing it, and 3) computing the solution associated to the new vector. Since each of these steps is polynomial, the overall process is polynomial. The same proof can be easily extended to non-Boolean domains. $\square$

Figure 3 shows an acyclic CP-net, with features A, B, and C, and its solution ordering. It is assumed that the variables have each two values: for A we have a and $\bar{a}$, and similarly for B and C. Also, the variable ordering is $A \prec B \prec C$. Given solution abc (that is, A=a, B=b, C=c), the associated Boolean vector (as described above) is 000, since a is the most preferred value for A, b is the most preferred value for B given A=a, and c is the most preferred value for C. Instead, the vector associated to solution $\bar{a}\bar{b}c$ is 100, and the vector associated to $\bar{a}b\bar{c}$ is 111. Given vector 101, the associated solution is $\bar{a}\bar{b}\bar{c}$. In Figure 3 it is possible to see the CP-net, the solution ordering, and the vector for each solution. Also, if we order the solutions according to a standard lexicographical order over their vectors, we get a linearization of the partial solution ordering.

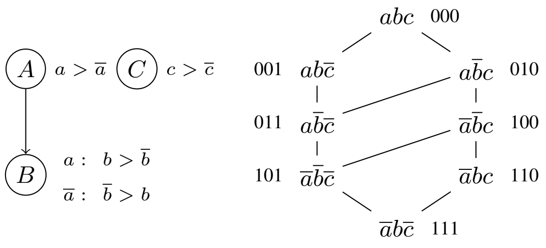

Figure 3. An acyclic CP-net and its solution ordering.

9 Next on constrained CP-nets

Some statements are better expressed via constraints, other via preferences. Moreover, some preferences are better modeled via soft CSPs, other via CP-nets. However, usually in a real-life problem we may have statements of all these kinds, thus requiring to use all the above considered formalisms in the same problem. It is therefore useful to consider problems where CP-nets and CSPs, or soft CSPs, coexist [3].

We thus consider here the notion of a *constrained CP-net*, which is just a CP-net plus some (soft) constraints [3]. Given a CP-net N and a constraint problem P, we will write (N,P) to denote the constrained CP-net given by N and P. For sake of simplicity, in the following we will assume that the CP-net and the CSP involve the same variables. Nevertheless, our results hold also for the more general setting.

Given a constrained CP-net (N,P), its solution ordering, written $\prec_{np}$, is that given by the (soft) constraints, where ties can be broken by the CP-net preferences. More precisely, solution s dominates solution s' (that is, $s \prec_{np} s'$) if s dominates s' according to the constraints in P, or s and s' are equally preferred according to the constraints in P, but s dominates s' according to the CP-net N.

We now ask consider the complexity of computing the next solution in a linearization of this ordering. The first results says that the problem is difficult if we take the lexicographical linearization (given o, which is an ordering over the variables) of $\prec_{np}$, denoted with $lex(o, \prec_{np})$.

Theorem 9 *Computing* $Next((N,P), s, lex(o, \prec_{np}))$*, where* (N,P) *is a constrained CP-net and* s *is one of its solutions, is NP-hard.*

The statement can be proven by reducing Next on a CSP to Next on a constrained CP-net. The same proof applies also to constrained CP-nets where the CP-net is acyclic.

Next becomes easy if we consider acyclic CP-nets, tree-shaped CSPs, and we add a compatibility condition between the acyclic CP-net and the constraints. This compatibility condition is related to the topology of the CP-net dependency graph and of the constraint graph.

Consider two variables in an acyclic CP-net, say x and y. We say that x *depends on* y if there is a dependency path from y to x in the acyclic DAG of the CP-net. Given an acyclic CP-net N and a tree-shaped CSP P, we say that N and P are *compatible* if there exists a variable of the CSP, say r, such that: for any two variables x and y such that x is the father of y in the r-rooted tree, we have that x does not depend on y in the CP-net. Informally, this means that it is possible to take a tree of the constraints where the top-down father-child links, together with the CP-net dependency structure, do not create cycles. If the compatibility holds for any root taken from a set S, then we will write that N and P are S-compatible.

Figure 4 shows an example of a CP-net DAG and two trees, of which the one in Fig. 4 (b) is compatible with the CP-net: if we choose A as the root, the father-child relationship is not contradicted by the CP-net dependencies. Instead, the tree in Fig. 4 (c) is incompatible with the CP-net: whatever root is chosen, some tree links are contradicted by the CP-net dependencies.

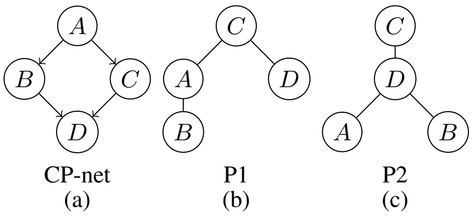

Figure 4. A CP-net dependency graph and two trees.

Theorem 10 *Consider an acyclic CP-net N and a tree-shaped CSP P, and assume that N and P are S-compatible, where S is a subset of the variables of P. Taken a solution s for* (N,P)*, and a variable ordering o which respects the tree shape of P with root an element of S, we have that* $Next((N,P), s, lex(o, \prec_{np}))$ *is in P.*

To compute $Next((N,P), s, lex(o, \prec_{np}))$, we use algorithm CSP-Next, except that we dynamically order each variable domains according the CP-net: for any variable, we order its domain according to the row of its CP table associated to the fixed assignment to the parent variables.

Under these same conditions, Next remains easy even if we consider CP-nets constrained by fuzzy CSPs rather than hard CSPs. We just need to adapt in a similar way algorithm FuzzyCSP-Next.

ACKNOWLEDGEMENTS

Research partially supported by the Italian MIUR PRIN project 20089M932N: "Innovative and multi-disciplinary approaches for constraint and preference reasoning".

REFERENCES

[1] Bistarelli, Montanari, and Rossi, 'Semiring-based constraint satisfaction and optimization', *Journal of the ACM*, **44**, 201–236, (1997).

[2] Craig Boutilier, Ronen I. Brafman, Carmel Domshlak, Holger H. Hoos, and David Poole, 'CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements', *Journal of Artificial Intelligence Research*, **21**, 135–191, (2004).

[3] Craig Boutilier, Ronen I. Brafman, Holger H. Hoos, and David Poole, 'Preference-based constrained optimization with CP-nets', *Computational Intelligence*, **20**(2), 137–157, (2004).

[4] R. Brafman, F. Rossi, D. Salvagnin, B. Venable, and T. Walsh, 'Finding the next solution in constraint- and preference-based knowledge representation formalisms', in *Proc. KR 2010*, (2010).

[5] A. A. Bulatov, V. Dalmau, M. Grohe, and D. Marx, 'Enumerating homomorphisms', in *Proc. STACS 2009*, (2009).

[6] R. Dechter, 'Tractable structures for CSPs', in *Handbook of Constraint Programming*, eds., P. Van Beek F. Rossi and T. Walsh, Elsevier, (2005).

[7] Rina Dechter, *Constraint Processing*, Morgan Kaufmann, 2003.

[8] Terence Kelly and Andrew Byde, 'Generating k-best solutions to auction winner determination problems', *SIGecom Exch.*, **6**(1), 23–34, (2006).

[9] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*, Wiley, 1990.

[10] P. Meseguera, F. Rossi, and T. Schiex, 'Soft constraints', in *Handbook of Constraint Programming*, eds., P. Van Beek F. Rossi and T. Walsh, Elsevier, (2005).

[11] D. Pisinger, 'Linear time algorithms for knapsack problems with bounded weights', *ALGORITHMS: Journal of Algorithms*, **33**, (1999).