Finding the Next Solution in Constraint- and Preference-based Knowledge Representation Formalisms

R. Brafman F. Rossi and D. Salvagnin and K. B. Venable T. Walsh
Department of Computer Science, Ben Gurion University, Negev, Beer-Sheva, Israel
Email: brafman@cs.bgu.ac.il
Dipartimento di Matematica Pura ed Applicata, University of Padova, Italy
Email: {frossi,kvenable,salvagni}@math.unipd.it
NICTA and UNSW Sydney, Australia
Email: Toby.Walsh@nicta.com.au

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.

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 (Dechter 2003). Another important task is to be able to compare two solutions and to say if one dominates another (Boutilier et al. 2004a). 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.

In this paper we study the computational complexity of the problem of computing the next solution in some constraint and preference-based formalisms. We show that this is a hard problem in constraint satisfaction problems (CSPs) (Rossi, Beek, and Walsh 2006), but it can be easy in tree-shaped CSPs (Dechter 2003) and tree-shaped fuzzy CSPs (Bistarelli, Montanari, and Rossi 1997). However, it is difficult in weighted CSPs, even if we restrict the shape of the constraint graph. Moreover, we also show that it is easy in acyclic CP-nets (Boutilier et al. 2004a), as well as in constrained acyclic CP-nets (Boutilier et al. 2004b) where the (soft) constraints are tree-shaped and topologically compatible with the CP-net graph.

We came across the problem of computing the next solution when studying the stable marriage problem (Gusfield and Irving 1989). The stable marriage problem is a well-known problem of matching men to women so that no man and woman who are not married to each other both prefer each other. Practical applications range from matching resident doctors to hospitals, to matching students to schools, to matching applicants to job offers, to any two-sided market. Stable marriage problems are usually solved with the Gale-Shapley algorithm. One of the main operations in this algorithm is computing the next solution in a preference ordering. However, the ability to compute the next solution is useful in many other scenarios. For instance, when we want to determine the k best solutions in an auction winner determination problem (Kelly and Byde 2006), or also when we look for the top k solutions in a web search, we want to find the optimal solution and the next $k-1$ solutions in the ordering. As a second example, suppose we are configuring a product, and the user doesn't like the first configuration we compute as we only know their preferences partially. We might choose to compute the next most preferred solution according to the preferences that we do know.

Formal background

Hard and soft constraints

A soft constraint (Bistarelli, Montanari, and Rossi 1997) is a constraint (Dechter 2003) 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 (Dechter 2003) is just a soft CSP where the chosen c-semiring is $S_{CSP} = \langle\{false, true\}, \vee, \wedge, false, true\rangle$. Fuzzy CSPs (Bistarelli, Montanari, and

Rossi 1997) 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 (soft CSP) 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 $pref(P,s) <_S pref(P,s')$.

Classical CSPs are usually solved via a backtracking search interleaved with constraint propagation (Dechter 2003). Soft constraints need instead to find an optimal solution, so they employ usually branch and bound techniques, where the bound computation exploits properties of the considered constraint class (Bistarelli, Montanari, and Rossi 1997).

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 (Dechter 2003). 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 (Bistarelli, Montanari, and Rossi 1997).

Fuzzy CSPs can also be solved via the well known 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 (Boutilier et al. 2004a) 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.

Determining if one outcome is better than another (a dominance query) is NP-hard even for acyclic CP-nets. Whilst tractable special cases exist, there are also acyclic CP-nets in which there are exponentially long chains of worsening flips between two outcomes. In the CP-net of the example, $\overline{a}b\overline{c}\overline{d}$ is worse than $abcd$.

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 of a solution ordering which is a total order with ties (no matter how many levels there are), 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 respect the existing dominance, while it can give an order between assignments that are incomparable.

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 explicitely, since it has an exponential length and it would mean knowing all the solutions and their relative order. For these reasons, we will assumpe 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.

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 preceds 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 $iin[1, n]$ 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 linearization given by $lex(O)$ is used, the problem of finding the next solution is NP-hard.

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

Proof We give a reduction from SAT. We shall compute Next on a Boolean CSP, which is a special case of CSPs with only two values in each variable domain (0 and 1). Let us assume $0 \prec_{o_i} 1$ for all i. Consider a set of SAT clauses ϕ. To build the corresponding CSP, we add the two new literals X and $\bar{Y}$ to each clause, and we position them as the first and second variables in the order o. These new clauses are then modelled as constraints for the CSP. We then ask for the next solution to the one in which X = Y = 0 and every other variable is set to 1. Notice that this is a solution. If ϕ is satisfiable, the next best solution of the CSP has X = 0, Y = 1 (and the first lexicographical solution of ϕ for the other variables). If ϕ is unsatisfiable, the next solution of the CSP has X = 1, Y = 0, and 0 for all other variables, since Y = 0 will satisfy all the constraints of the CSP. Hence a single call to Next determines the satisfiability of ϕ. □

Thus, given a CSP, there is at least a linearization of its solution ordering, and a solution s, such that finding the Next solution in l after s is difficult. The result of the previous theorem can be extended to a wider class of orderings, as the following theorem states.

Theorem 2 *For each total order ω such that*

- *ω defines an ordering on the complete variable assignments;*
- *ω is polynomially describable, that is, given any CSP, it can be specified in time polynomial with respect to the size of the CSP;*
- *computing the top element of ω is polynomial;*
- *the top element of ω does not depend on the constraints of the CSP;*

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(ω)), where p is a CSP, is NP-hard.

Proof We give a reduction from SAT. Consider a formula φ. Let s be the first assignment in the total order ω (by hyphotesis, s can be computed in polynomial time). Let ψ be a formula asserting the truth of s (e.g., if s is $x_1 = true$ and $x_2 = false$, then ψ could be $x_1 \wedge \neg x_2$). Let $p = \varphi \vee \psi$ and $t = Next(p, s, l(\omega))$. If s or t satisfies φ, then φ is satisfiable, otherwise it is not. Hence a single call to $Next$ determines the satisfiability of φ. □

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, that is, its underlying constraint graph has no cycles, it can be easy to find the next solution.

In this section we focus on tree-shaped binary CSPs. However, the same results hold for (binary or non-binary) CSPs with a 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: $(a, b, a, b, b) \prec (a, b, a, c, b) \prec (b, a, b, a, a) \prec (b, a, b, a, b) \prec (b, a, b, c, a) \prec (b, a, b, c, b) \prec (b, b, b, b, b) \prec (b, b, b, c, b)$.

We will now describe an algorithm that, given as input a DAC tree-shaped CSP P and a solution s for P, it either returns the consistent assignment following s according to

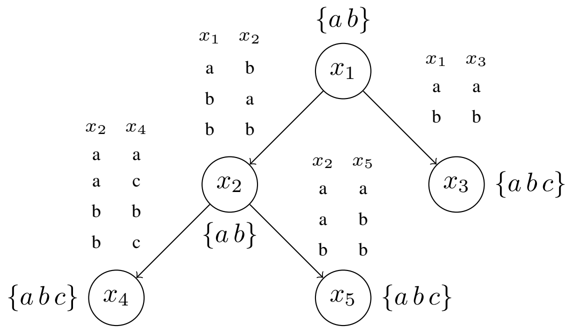

Figure 1: A tree-shaped CSP.

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"
```

$tlex(O)$, 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 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".*

Proof If s is not the last solution, then it has successor s' which is also a solution. Assume $s = (s_1, \cdots, s_n)$ and $s' = (s'_1, \cdots, s'_n)$. Since s and s' are both solutions, it must be that s' is lexicographically greater than s: there must be a variable x_j such that, $\forall i < j$ $s_i = s'_i$ and $s_j \prec_{o_j} s'_j$. This means that CSP-next will not terminate returning "no more solutions", since there is at least an iteration of the for-loop (with $i = j$), where the **if** condition will be satisfied and an assignment will be returned. Moreover, it easy to see that this assignment must be a solution, since it coincides with s up to variable x_{i-1}, it assigns a value to x_i which is consistent with previous values, and all variables following x_i are set to their smallest compatible values (w.r.t. the domain orderings). Such values must exist due to the fact that P is DAC and tree-shaped.

Let s'' be the solution returned by CSP-next. From the reasoning above, we have that CSP-next must have terminated when $i = j$ or before. If it has terminated when $i = k > j$, we would have that s and s'' coincide up to the value for the k-th variable and $s_k \prec_{o_k} s''_k$. Since $k > j$, s'' would precede s' lexicographically. This is not possible since s' is the successor of s. We can thus conclude that CSP-next must have stopped with $i = j$. Thus it must be that s' and s'' coincide up to variable j. Let, x_k, with $k > j$ be the first variable on which they differ. By the definition of function Reset-succ it must be that $s''_k \prec_{o_k} s'_k$. But in such a case we would have $s \prec_{tlex(O)} s'' \prec_{tlex(O)} s'$ which contradicts that s' is the successor of s. This allows us to conclude that $s' = s''$.

If s is the last solution in the ordering, then, for every variable v, there is no larger value in the domain ordering than the one assigned in s that is consistent with the values assigned by s to v's father. This means that the **if** condition is never satisfied. Thus, CSP-Next returns "no more solutions". $\square$

If $|D|$ is the cardinality of the largest domain, it 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 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 any linearization of its solution ordering, is NP-hard.*

Proof We give a reduction from the subset sum problem. Given a set C of integer elements $\{t_1, \ldots, t_n\}$ and an integer t, the subset sum problem consists in finding a subset C' of elements of C such that their sum equals t, or show that no such subset exists. Given a subset sum problem $\{t_1, \ldots, t_n, t\}$, let P be a CSP with $n + 1$ variables $\{x_0, x_1, \ldots, x_n\}$, whose domains are follows: $x_0 \in \{t - 1/2, 0\}, x_i \in \{t_i, 0\}$ $\forall i \in \{1, \ldots, n\}$, and an empty constraint set. Then, we consider the total order l that ranks the solutions by increasing sum of their values, using a lexicographic order to break ties. Consider the solution $s = \{t - 1/2, 0, \ldots, 0\}$ and take $s' = Next(P, s, l)$. If the sum of the values of s' is t, we have found a feasible solution for our subset sum problem (note that in such a solution $x_0 = 0$, otherwise the sum would be fractional), otherwise we have proven that there is none. $\Box$

A problem similar to that of finding the next solution in a CSP has been considered in (Bulatov et al. 2009), where the computational complexity of the problem of enumerating all solutions of a CSP (including the first one) has been studied, and some classes of CSPs (including tree-like CSPs) are shown to have a polynomial delay algorithm to solve this problem. For such classes, it is therefore tractable to find the next solution. However, they do not provide any concrete polynomial algorithm to find the next solution to a given one, in a given ordering.

Next on weighted CSPs

We will show that Next on weighted CSPs is always a difficult problem.

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.*

Proof The proof of Theorem 4 can be easily adapted for the purpose. Let P be a weighted CSP, with $n+1$ binary variables $\{x_0, x_1, \ldots, x_n\}$ and unary constraints c_i on the variable domains as follows: $c_0(1) = t - 1/2, c_i(1) = t_i \ \forall i \in \{1, \ldots, n\}, c_i(0) = 0 \ \forall i \in \{0, \ldots, n\}$. Consider the solution $s = \{1, 0, \ldots, 0\}$ with cost $t - 1/2$. For any linearization l, if $s' = Next(P, s, l)$ has cost t, we have found a feasible solution for our subset sum problem, otherwise we have proven that there is none. $\Box$

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.

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 $X = \{x_1, \ldots, x_n\}$ and set of constraints C, and let us consider a variable ordering $o = \{x_1, \ldots, x_n\}$ which respects the tree shape. Moreover, let o_i be a total order over the values in the domain of x_i, for $i = 1, \ldots, n$.

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 preceeds 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 preceeds 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 $pref(P, s)\}$ and $t'_s = min_{o_T}\{t$ tuple of s' with preference $pref(P, s')\}$. We write $s \prec_f s'$ (that is, s preceeds s' in ordering $\prec_f$), if

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

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

Lemma 6 *Ordering $\prec_f$ it a strict total order over the set of solutions, which linearizes the solution ordering.*

Proof The fact that $\prec_f$ is a strict total order derives directly from the fact that $o, o_1, \ldots, o_n$ and o_T are strict total orders by definition and, thus, so are the lexicographic orders they induce. This implies that all solutions having the same preference are strictly ordered according to $\prec_f$. Moreover, by definition of $\prec_f$, if two solutions have a different preference, the one with higher preference preceeds the other one according to $\prec_f$. $\Box$

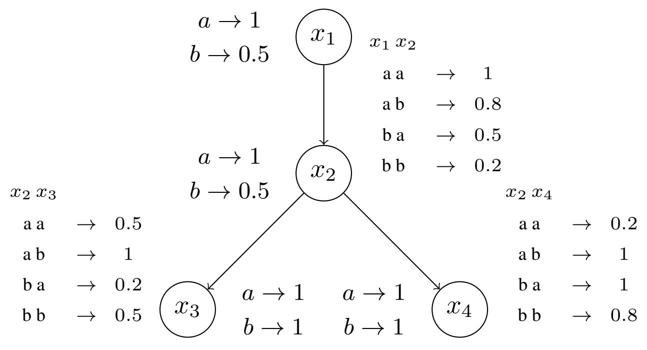

Figure 2: A tree-shaped DAC fuzzy CSP.

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 $i = 1, \ldots, 4$. Then, if we consider solutions $s = (b, a, a, b)$ and $s' = (a, b, b, b)$, we have that $s \prec_f s'$ since $pref(P, s) = pref(P, s') = 0.5 < 1 =$

$opt(P)$, $t_s = (x_1 = b, x_2 = a)$, $t_{s'} = (x_2 = b, x_3 = b)$, 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 $pref(P, s) = pref(P, s') = 0.2 < 1 = opt(P)$, $t_s = (x_1 = b, x_2 = b) = t'_s$, 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.

Algorithm 2: FuzzyCSP-Next

```
Input: tree-shaped and DAC Fuzzy CSP $P$, orderings
$o, o_1, \dots, o_n, o_T$, 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 $o_T$ with
$pref(t) > 0$ 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”
```

In Algorithm 2:

- $opt(P)$ denotes the optimal preference of a fuzzy CSP P;
- $next(p)$ is the preference value, among those appearing in P, following p in decreasing order;
- 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 the background section;
- 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.

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, $|D|$ is the cardinality of the largest domain, and n is the number of variables.*

Proof (Sketch) 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. □

Therefore, the next solution can be computed in polynomial time.

Theorem 8 *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 computed in polynomial time.*

Proof Follows directly from Theorem 7 and from the fact that applying DAC is polynomial. □

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.

Next on acyclic CP-nets

As noticed above, the solution ordering in an acyclic CP-net is a partial order with one top element. The acyclic nature of the CP-net makes it easy to find the unique optimal solution (by sweeping forward in the CP-net DAG). 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 (this condition is necessary to obtain a linearization of the solution 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: for each variable in the given ordering, the values of the vector corresponding to its parents tell us the row of the CP-table to consider for that variable, and the value of the vector corresponding to the variable tells us which value to choose.

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 *contexual lexicographical linearization*. Note that there is at least one such linearizations for every acyclic CP-net.

Theorem 9 *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.*

Proof 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. □

Figure 3 shows an acyclic CP-net, with features A, B, and C, and its solution ordering. It is assumed that the variables each have 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.

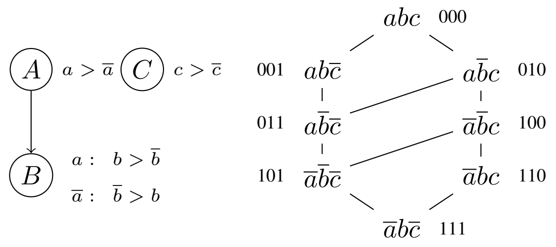

Figure 3: An acyclic CP-net and its solution ordering.

Next on constrained CP-nets

Some statements are better expressed via constraints, others via preferences. Moreover, some preferences are better modelled via soft CSPs, others 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 (Boutilier et al. 2004b).

We thus consider here the notion of a *constrained CP-net*, which is just a CP-net plus some (soft) constraints (Boutilier et al. 2004b). 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 the one given by the (soft) constraints, where ties can be broken by the CP-net preferences. More precisely, in the solution ordering of a constrained CP-net, 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.

If, given two solutions, no one dominates the other one according to this definition, they are considered in a tie. Notice that, in such an ordering, solutions that are in a tie are incomparable for the CP-net N. Notice also that, when we consider classical CSPs, the notion of dominance between s and s' means that s is a solution and s' is not.

We now 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 10 *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.*

Proof The statement can be proven by reducing Next on a CSP to Next on a constrained CP-net. Given a CSP P, we can consider a constrained CP-net where the CP-net has just one variable, say x, and the CSP is obtained from P by adding the unconstrained variable x. In such a constrained CP-net, take a solution s, and assume it is not the last one in the ordering $lex(o, \prec_{np})$. Then, let s' be the next solution according to $lex(o, \prec_{np})$. Now, if s and s' have the same variable assignment for x, then $t' = Next(P, t, lex(o))$, where t and t' are obtained from s and s' by deleting the assignment to x. If s is the last solution or s and s' have a different value for x, then t (as defined above) is the last one in the ordering $lex(o)$. Thus, computing Next in the constrained CP-net would also compute Next on the CSP. Since Next on generic CSPs is difficult, as shown in in Theorem 1, also Next on constrained CP-nets is so. □

The same proof applies also to constrained CP-nets where the CP-net is acyclic.

Acyclicity and compatibility

In the previous section we have see that finding the next solution in a constrained CP-net, even an acyclic one, is difficult. However, in this section we show that 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 directed acyclic graph (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.

Theorem 11 *Consider an acyclic CP-net* N *and a tree-shaped CSP* P*, and assume that* N *and* P *are S-compatible,*

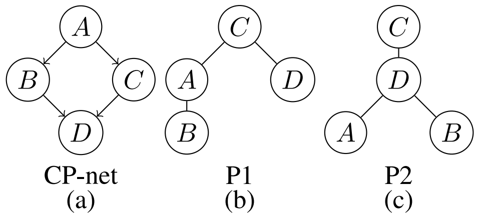

Figure 4: A CP-net dependency graph and two trees.

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.*

Proof 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. In this way, when we choose the next value for a variable which is compatible with the father variable (in order to find the next solution), we take the next most preferred one according to the CP-net preference statements. Thus, we find a new solution (if it exists) and, among the solutions not considered so far, we take the most preferred according to the CP-net. This models exactly the solution ordering $lex(o, \prec_{np})$ defined for constrained CP-nets in the previous section: if the algorithm returns a new solution s', it means that s is not the last solution and s' is the next solution to s in the ordering. □

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.

Theorem 12 *Given an acyclic CP-net* N*, a tree-shaped fuzzy CSP* P*, and a solution* s *for* (N, P)*,* $Next((N, P), s, lex(o, \prec_{np}))$ *is tractable if* N *and* P *are compatible.*

Proof As in the proof of the previous theorem, we can adapt in a similar way the algorithm to compute Next on fuzzy CSPs in order to compute Next on constrained CP-nets where the constraints are fuzzy. Since FuzzyCSP-Next uses CSP-Next, this can be simply done by instead using the modified version of CSP-Next as described in the proof of the previous theorem. □

The above compatibility condition between the CP-net and the (soft) constraints is naturally satisfied in some scenarios. Consider for example the situation in which we first identify the topology of the problem to be modelled (that is, variables and links among them) and then we decide if such links are constraints or conditional dependencies a la CP-net. If the topology is acyclic, then the constraint graph and the dependency graph are both acyclic and are compatible.

Conclusions and future work

Finding the next solution in a solution ordering of a constraint or preference problem can be very useful in many settings. It is therefore important to understand the computational cost of this operation. We considered some well-known knowledge representation formalisms and we identified several islands of tractability for the problem of finding the next solution, such as acyclic CSPs, acyclic fuzzy CSPs, acyclic CP-nets, and acyclic constrained CP-nets where the constraints and the CP-net are topological compatible. We also showed that the problem is NP-hard for CSPs and linearizations with certain features, as well as for weighted CSPs (no matter which linearization is chosen). In the future, we intend to look for other tractable cases, and to investigate scenarios where the compatibility conditions considered in this paper naturally hold. We also plan to test experimentally how difficult it is in practice to find the next solution.

References

Bistarelli; Montanari; and Rossi. 1997. Semiring-based constraint satisfaction and optimization. *Journal of the ACM* 44:201–236.

Boutilier, C.; Brafman, R. I.; Domshlak, C.; Hoos, H. H.; and Poole, D. 2004a. CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research* 21:135–191.

Boutilier, C.; Brafman, R. I.; Hoos, H. H.; and Poole, D. 2004b. Preference-based constrained optimization with CP-nets. *Computational Intelligence* 20(2):137–157.

Bulatov, A. A.; Dalmau, V.; Grohe, M.; and Marx, D. 2009. Enumerating homomorphisms. In *Proc. STACS 2009*.

Dechter, R. 2003. *Constraint Processing*. Morgan Kaufmann.

Gusfield, D., and Irving, R. W. 1989. *The Stable Marriage Problem: Structure and Algorithms*. MIT Press.

Kelly, T., and Byde, A. 2006. Generating k-best solutions to auction winner determination problems. *SIGecom Exch.* 6(1):23–34.

Rossi, F.; Beek, P. V.; and Walsh, T. 2006. *Handbook of Constraint Programming*. Elsevier.