The Next Best Solution

R. Brafman Ben Gurion Univ., Beer-Sheva, Israel
E. Pilotto, F. Rossi, D. Salvagnin, K. B. Venable Univ. of Padova, Italy
T. Walsh NICTA and UNSW Sydney, Australia

Abstract

We study the computational complexity of finding the next most preferred solution in some common formalisms for representing constraints and preferences. The problem is computationally intractable for CSPs, but is polynomial for tree-shaped CSPs and tree-shaped fuzzy CSPs. On the other hand, it is intractable for weighted CSPs, even under restrictions on the constraint graph. For CP-nets, the problem is polynomial when the CP-net is acyclic. This remains so if we add (soft) constraints that are tree-shaped and topologically compatible with the CP-net.

Introduction

In combinatorial satisfaction and optimization, we often want to find a satisfying or optimal solution, as well as to decide if one solution is better than another. However, there are other useful reasoning tasks. One such task is finding the *next* solution – the solution that comes next according to an ordering where more preferred solutions are ordered first. We have therefore started a systematic study of the computational complexity of computing the next solution in some common constraint and preference-based formalisms. Our results cover general CSPs, fuzzy CSPs, weighted CSPs and CP-nets, as well as their acyclic versions.

We came across this problem when computing stable marriages (Gusfield and Irving 1989a). This problem has many practical applications (from matching resident doctors to hospitals, to matching students to schools, to matching applicants to job offers, to any two-sided market). For large stable marriage problems, we have proposed using formalisms like CP-nets which can compactly represent preferences. Computing a stable marriage in an algorithm like Gale-Shapley's then requires being able to find the next most preferred marriage partner in such a preference representation. However, computing the next solution is useful in many other scenarios. For instance, it is useful when we ask for the top k solutions in a web search. As a third example, suppose we are configuring a product, and the user doesn't like the first configuration computed 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. Finally, an efficient next operation can be used as a tool for producing diverse solutions, a feature which is increasingly interesting in practical applications. This paper summarizes results appearing in (Brafman et al. 2010) and (Pilotto et al. 2009).

Background

Hard and soft constraints. A constraint satisfaction problem (CSP) (Dechter 2003) is a set of variables, each with a domain of possible values, and a set of constraints. Each constraint involves some of the variables and is satisfied only by some combination of values of such variables. A solution of a CSP is an assignment of all variables which satisfies all constraints.

A soft CSP is a CSP where all constraints are soft. A soft constraint (Meseguer, Rossi, and Schiex 2005; Bistarelli, Montanari, and Rossi 1997) assignes a preference value, taken from a given set, to each instantiation of its variables. The set of preference values is ordered: a better prerference is higher in the ordering. Soft CSPs come equipped also with a combination operator that tells us how to combine the preferences of two or more constraints. In soft CSPs, the preference of each solution is given by the combination of all the preferences given by the constraints to the solution. A solution is optimal if there is no other solution with a better preference.

Fuzzy CSPs (Meseguer, Rossi, and Schiex 2005) are obtained by considering preference values in [0,1], where a higher value denotes a better preference, and by combining them via the *min* operator: optimal solutions maximize the minimum preference. Finally, the so-called weighted CSPs use preferences in $\mathbb{R}^+$ that are interpreted as costs, thus a higher value denotes a worse cost, and costs are combined via *sum*: optimal solutions minimize the sum of the costs.

Constraint propagation in classical CSPs reduces variable domains, improving search performance. For tree-shaped CSPs, directional arc-consistency (DAC) applied bottom-up finds a solution without backtracking. Constraint propagation can be applied also to soft CSPs. DAC is enough to find the optimal solution to a fuzzy CSP when the problem has a tree shape (Meseguer, Rossi, and Schiex 2005).

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. 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.

The semantics of CP-nets depends on the notion of a *worsening flip*. This is the change in the value of a variable to a less preferred value according to the cp statement for that variable. 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 induces a preorder over the outcomes, and a partial order if the CP-net is acyclic. In general, finding the optimal outcome of a CP-net, as well as comparing two outcomes, is NP-hard (Boutilier et al. 2004a). However, in acyclic CP-nets, there is only one optimal outcome which can be found in linear time by sweeping through the dependency graph, assigning the most preferred values to each variable.

Solution orderings

The constraint and preference-based formalisms introduced 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 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 requires a strict linear order. We may therefore have to linearize 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. If we consider fuzzy or weighted CSPs, there can be no incomparability (since the set of preference values is totally ordered), so we have a total order with ties, and a solution dominates another one if its preference value is higher. In acyclic CP-nets, the solution ordering is a partial order.

In the following, given a problem P and a linearization l of its solution ordering (given implicitely), Next(P,s,l) is 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 this solution ordering.

Finding the next solution in CSPs

Let P be a CSP with n variables. 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 denote by O the set of orderings $\{o, o_1, \ldots, o_n\}$. These orderings naturally induce a lexicographical linearization of the solution ordering, $lex(O)$, where given two variable assignments, 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 $j < i$). For the linearization $lex(O)$, the problem of finding the next solution is computationally intractable.

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

This result can be extended to a wider class of orderings.

Theorem 2. *Consider any polynomially describable and computable total order ω over variable assignments whose top element does not depend on the constraints of the CSP, and the linearization $l(\omega)$ of the solution ordering induced by ω. Then there exists a CSP P and a solution s such that computing Next(P,s,l(ω)) is NP-hard.*

The proof of both theorems are based on a polynomial reduction from SAT.

We know that finding an optimal solution becomes polynomial if the constraint graph is a tree. It is therefore natural to consider this class to see whether also the Next problem becomes polynomial. 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 parent comes before its children.

We have defined an algorithm (called CSP-Next) that, given as input a directionally arc consistent tree-shaped CSP P and a solution s for P, either returns the satisfying assignment following s according to $tlex(O)$, or detects that s is the last satisfying assignment in this ordering. The algorithm works bottom-up in the tree, looking for new values for children that are consistent with the value assigned to their parent 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.

Theorem 3. *Computing $Next(P, s, tlex(O))$, where P is a tree-shaped and DAC CSP, is polynomial.*

In fact, 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 smallest consistent assignment takes $O(|D|)$, and such operations are done $O(n)$ times. The following results shows that the choice of the linearization is crucial for tractability.

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 polynomial reduction from the subset sum problem.

Next on weighted CSPs

With weighted CSP, finding the next solution means that, given a solution, we return the next assignment in lexicographical order with the same cost or, if there is no such assignment, the first assignment in lexicographical order with the next smallest cost. Unfortunately, the following result holds:

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 is based on a polynomial reduction from the subset sum problem. Note that theorems 4 and 5, whilst having similar proofs, have quite different implications. Indeed, for tree-shaped CSPs computing Next is NP-hard only for some choices of the linearization l, while for weighted CSPs computing Next is *always* NP-hard, irrespective of the linearization, because the "native" solution ordering is already sufficient for NP-hardness. However, if we consider just lexicographical orderings, and weighted CSPs with unary constraints only, then Next is not strongly NP-hard since a pseudo-polynomial algorithm exists.

Theorem 6. *Given a weighted CSP P with unary constraints only, a solution a, and a linearization l induced by a lexicographic ordering, computing Next(P,a,l) is weakly NP-hard.*

The proof uses a simple generalization of the dynamic programming algorithm for deciding subset sum. Thus, the complexity here only comes when the weights are large.

Next on tree-shaped fuzzy CSPs

With fuzzy CSPs, Next appears more tractable. In particular, Next on tree-like fuzzy CSPs is polynomial. The algorithm proposed in (Brafman et al. 2010) exploits the fact that, in a fuzzy CSP, a solution can have preference p only if it includes a tuple that has preference p. The worst case time complexity of the algorithm is $O(|T||D|n)$, where $|T|$ is the number of tuples of P and $|D|$ the cardinality of the largest domain.

Theorem 7. *Given a tree-shaped DAC fuzzy CSP P and a solution s, computing Next(P,s,lex(O)) is polynomial.*

Again, the choice of the order is crucial for the complexity of the algorithm. For instance, Theorem 4 implies that Next(P,s,l) is NP-hard on tree-shaped fuzzy CSPs in general.

Next on acyclic CP-nets

On acyclic CP-nets, Next is polynomial to compute 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 simplicity, we consider Boolean domains. Given an acyclic CP-net with n variables, we 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 its value is 0 if the variable has its most preferred value, given the values of the parents, and 1 otherwise. The optimal solution thus gives 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 polynomial to compute the corresponding assignment.

The *contextual lexicographical linearization* of the ordering of the solutions linearizes incomparability via 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 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 polynomial.*

Next on constrained CP-nets

It is often 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. Its solution ordering, $\prec_{NP}$, is the ordering given by the (soft) constraints, where ties are broken by the CP-net preferences. Unfortunately, computing the next solution is intractable if we take the lexicographical linearization (given o, which is an ordering over the variables) of $\prec_{np}$, denoted by $lex(o, \prec_{NP})$.

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 proof uses a polynomial reduction from the Next problem on CSPs.

Next becomes polynomial 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 constraint graph and the dependency graph of the CP-net. Informally, compatibility means that it is possible to take a tree of the constraints where the top-down parent-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.

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 whose root is an element of S, then $Next((N, P), s, lex(o, \prec_{NP}))$ is polynomial.*

Under these same conditions, Next remains polynomial even if we consider CP-nets constrained by fuzzy CSPs rather than hard CSPs.

Next for stable marriage problems

The stable marriage problem is a well-known matching-problem (Gusfield and Irving 1989b). Given n men and n women, where each person strictly orders all members of the opposite sex, we wish to marry the men to the women such that there is not a man and woman who would both rather be married to each other than to their current partners. If there is no such couple, the matching is called *stable*. Surprisingly, a stable marriage always exists. A well-known algorithm to solve this problem is the Gale-Shapley (GS) algorithm (Gale and Shapley 1962). The algorithm consists of a number of rounds in which each un-engaged man proposes to the most preferred woman to whom he has not yet proposed. Each woman receiving a proposal becomes "engaged", provisionally accepting the proposal from her most preferred man. The main operations that this algorithm requires are: for a man, to compute his most preferred woman and, given any woman, the next best one; for a woman, to compare two men according to her preferences.

In some applications, the number of men and women can be large. For example, the men and women might represent combinatorial structures. It may therefore be unreasonable to assume that each man and woman provides a strict ordering of the members of the other sex. Here we assume that each man and woman has an acyclic CP-net decribing his/her preferences over the members of the other sex. In this scenario, the operations needed by the GS algorithm must be computed on acyclic CP-nets. The existing literature tells us that finding the best outcome is easy. The results of the previous sections assure us that finding the next best woman is polynomial in the number of features for an appropriate linearization. In this same linearization, it is also easy to compare two outcomes (Pilotto et al. 2009).

We can either precompute the whole linear order, or we can compute just the part of the ordering that GS needs during its execution. In the first scenario, we compute $Next$ n^2 times and then GS can run in $O(n^2)$ time. In terms of space, we need to store all the linearizations, which takes $O(n^2)$ space. In the second scenario, no pre-computation is required but each step of GS requires additional time to perform a $Next$ (and possibly a $Compare$) operation. GS now runs in $O(n^2 log(n))$ time but just $O(nlog(n))$ space. We ran some experiments to see which of these two scenarios is more effective in practice with randomly generate acyclic CP-nets.

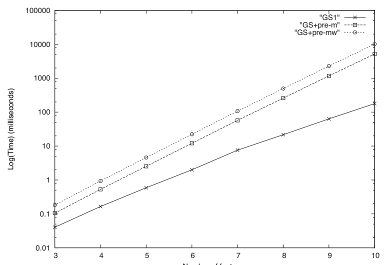

Figure 1: Execution time for three versions of GS.

Figure 1 gives the log-scale time needed to run three different version of the GS algorithm: when Next and Compare are executed on demand (GS1), when the linearization for the men is precomputed, but Compare for the women is executed on demand (GS+pre-m), and when both linearizations are precompted (GS+pre-mw). We see that is is inefficient to pre-compute the linearizations in advance, even for just the men. This is perhaps not too surprising, since computing the linearizations needs to run the Next operation exactly n^2 (or $2n^2$) times, while the GS algorithm is $O(n^2)$ time in the worst case but may in practice require only a much smaller number of proposals. Our tests show that algorithm GS takes less than 1 second even for CP-nets with 10 features. This is a practical sized setting as it models problems with about a thousand members of each sex where it would be unreasonable to ask each agent to rank all members of the other sex. By comparison, it is a simple task to ask an user to specify a CP-net over 10 features.

Acknowledgments

This research is partially supported by the Italian MIUR PRIN project 20089M932N: "Innovative and multidisciplinary approaches for constraint and preference reasoning".

References

Bistarelli, S.; Montanari, U.; and Rossi, F. 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.

Brafman, R.; Rossi, F.; Salvagnin, D.; Venable, B.; and Walsh, T. 2010. Finding the next solution in constraint- and preference-based knowledge representation formalisms. In *Proc. KR 2010*.

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

Gale, D., and Shapley, L. S. 1962. College admissions and the stability of marriage. *Amer. Math. Monthly* 69.

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

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

Meseguer, P.; Rossi, F.; and Schiex, T. 2005. Soft constraints. In F. Rossi, P. V. B., and Walsh, T., eds., *Handbook of Constraint Programming*. Elsevier.

Pilotto, E.; Rossi, F.; Venable, K. B.; and Walsh, T. 2009. Compact preference representation in stable marriage problems. In Rossi, F., and Tsoukiàs, A., eds., *ADT*, volume 5783 of *Lecture Notes in Computer Science*, 390–401. Springer.