Robust Solutions in Unstable Optimization Problems

Maria Silvia Pini[1], Francesca Rossi[1], Kristen Brent Venable[1], and Rina Dechter[2]

[1] Dipartimento di Matematica Pura ed Applicata, Università di Padova, Italy
{mpini,frossi,kvenable}@math.unipd.it
[2] School of Information and Computer Science, University of California, Irvine, CA, USA
dechter@ics.uci.edu

Abstract. We consider constraint optimization problems where costs (or preferences) are all given, but some are tagged as possibly unstable, and provided with a range of alternative values. We also allow for some uncontrollable variables, whose value cannot be decided by the agent in charge of taking the decisions, but will be decided by Nature or by some other agent. These two forms of uncertainty are often found in many scheduling and planning scenarios. For such problems, we define several notions of desirable solutions. Such notions take into account not only the optimality of the solutions, but also their degree of robustness (of the optimality status, or of the cost) w.r.t. the uncertainty present in the problem. We provide an algorithm to find solutions accordingly to the considered notions of optimality, and we study the properties of these algorithms. For the uncontrollable variables, we propose to adopt a variant of classical variable elimination, where we act pessimistically rather than optimistically.

1 Introduction

Constraint programming [2,11] is successfully applied to many application domains. Constraint satisfaction problems are defined by decision variables, domains, and constraints that have to be satisfied. Optimization problems have an objective function, or they associate costs, or preferences, with partial variable instantiations, in order to discriminate among the possibly many solutions of the problem.

Soft constraints [1] are a general formal modelling framework for constraint optimization problems, where it is possible to express several different optimization criteria. For example, both fuzzy constraints and weighted constraints, as well as MaxCSPs, can naturally be modelled in this formalism.

The specification of a complex constraint optimization problem is a difficult modelling task, that tries to capture the current knowledge about the constraints and the costs of the problem. Even when the specification is complete, only some parts of the problem's parameters may be certain. Others may be viewed as unstable due to possible future changes.

A. Oddi, F. Fages, and F. Rossi (Eds.): CSCLP 2008, LNAI 5655, pp. 116–131, 2009.

Unstable costs are present in many real-life problems. A typical example is the budget estimate for next year in a company. Typically, such an estimate is based on data which is not known or not certain, and most of the times such uncertainty is represented by using last year's value (which can be seen as the default value), plus some range of possible other values around the default value. For example, one may not have the cost of fuel for next year, but he may know that last year it was 2 dollars, and usually the new value is never more than 30% higher. As another example, one may have to base some calculation on the number of pieces that will be produced in the year: a reasonable estimate could be last year's number assuming that the new number is within 5% from the old value. In the first example, the default value is at the lower end of the range, while in the second example it is in the middle of the range.

Other types of problems where unstable values may occur are when we want to numerically represent linguistic concepts, such as "more or less", "around", "at least", or "at most". In all these cases, the natural formulation is to have a value and a range around (or above, or below) such a value.

In all these settings, it is often possible to express the instability by a bounding range of values. As another example, we may have a default cost of 10 for each piece of a material, with a range from 5 to 15 containing all possible foreseen alternatives costs.

Even though costs are unstable, we would still like to reason and perform inference on the given "default" optimization problem. This is possible in some cases, as we will see in this paper.

Given constraint optimization problems where some of the costs are tagged as unstable, we define several notions of desirable solutions, that take into account not only cost-optimality, but also a form of robustness (of the optimality status, or of the cost) with respect to the uncertainty present in the problem. For example, we could desire solutions that are cost-optimal and that remain optimal even if the unstable costs change. In other scenarios, it could instead be important to find solutions that are cost-optimal and whose cost does not increase if the unstable costs change.

Some of the considered notions will yield sets of solutions that can possibly be empty, while others (usually the least attractive) will always have at least a single element. For each of the notions of optimality, we provide an algorithm to find solutions according to that criterion, and we study their properties.

In addition to the notion of instability, we also accomodate the dichotomy of having some uncontrollable variables, whose value cannot be decided by the agent, but will be decided by Nature or by some other agent. This yields an orthogonal form of uncertainty often found in scheduling or temporal problems [12], where the occurrence of certain events can be decided only by others. For example, in scheduling the activities of a satellite taking pictures of Earth, we may have to schedule in advance the best times for taking some pictures of an area without knowing the local weather conditions (that heavily impacts on the quality of the pictures), which is decided by Nature.

To handle the uncontrollable variables, we adopt a variant of classical variable elimination, where we act pessimistically. This allows processing the uncontrollable variables first, and then working on the controllable part ensuring that the resulting inferences are safe with respect to the uncontrollable part.

Interestingly, no matter what notion of optimality we use, the complexity of reasoning with the unstable and uncontrollable problems considered in this paper do not increase the overall worst case complexity (for complete algorithms). In particular, if the default problem belongs to a tractable class, even its unstable and/or uncontrollable version is tractable.

Issues related to those considered in this paper have been studied also in open CSPs [6] and interactive CSPs [9]. However, in such frameworks the uncertainty is in the form of missing domain values, and not unstable costs. Also, in dynamic CSPs [3], variables, domains, and constraints may change over time. However, no preference range is given, and there are no uncontrollable variables. Preference ranges are considered in [14], however no default value is given in the various preference ranges. In [7] only uncontrollable variables are considered, but no imprecise ranges are given. In [8] some preferences are missing (thus there are no default value nor preference ranges) and the focus is on preference elicitation to obtain the so-called necessarily optimal solutions (called O-ROB in this paper). A similar setting can be found in [13] for hard CSPs.

2 Background: Soft Constraints for Optimization Problems

A soft constraint [1] is just a classical constraint [2] where each instantiation of its variables has an associated value from a (totally or partially ordered) set. This set has two operations, which makes it similar to a semiring, and is called a c-semiring.

More precisely, a c-semiring is a tuple $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ containing a set of preferences A, a combination operation $\times$, that is useful to combine preferences, and an additive operator $+$ that induces a partial order $\leq$ over A. Such an ordering gives us a way to compare (some of the) tuples of values and constraints. In fact, when we have $a \leq_S b$, we will say that b *is better than* a. Thus, $\mathbf{0}$ is the worst value and $\mathbf{1}$ is the best one. The combination operator is *intensive*, that is, $\forall a, b \in A$, $a \times b \leq_S a, b$.

A c-semiring $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ is said to be *strictly monotonic* iff the combination operator $\times$ is strictly monotonic, i.e., for every $a, b \in A$, if $a < b$ then, for every $c \in A$, $a \times c < b \times c$.

Given a set of variables V with finite domain D, and a c-semiring $\langle A,+,\times,0,1\rangle$, a *soft constraint* is a pair $\langle def, con\rangle$ where $con \subseteq V$ is the scope of the constraint and $def : D^{|con|} \longrightarrow A$ is the preference function of the constraint associating to each tuple of assignments to the variables in con either a preference value ranging between $\mathbf{0}$ and $\mathbf{1}$. A *soft constraint problem* (SCSP) is a triple $\langle C, V, D\rangle$, where C is a set of soft constraints over the variables in V with domain D.

Given two constraints $c_1 = \langle def_1, con_1 \rangle$ and $c_2 = \langle def_2, con_2 \rangle$, their *combination* $c_1 \otimes c_2$ is the constraint $\langle def, con \rangle$ defined by $con = con_1 \cup con_2$ and $def(t) = def_1(t \downarrow_{con_1}^{con}) \times def_2(t \downarrow_{con_2}^{con})$[1]. In words, combining two constraints means building a new constraint which involves all the variables of the original ones and which associates to each tuple of domain values for its variables a specific semiring element. Such an element is obtained by multiplying the elements associated by the original constraints to the appropriate subtuples.

It may be useful to eliminate some variables from a constraint, using a notion of projection. Given a subset of variables $I \subseteq V$, and a soft constraint $c = \langle def, con \rangle$, the *projection* of c over I, written $c \Downarrow_I$, is a new soft constraint $\langle def', con' \rangle$, where $con' = con \cap I$ and $def(t') = \sum_{\{t | t \downarrow_{con'} = t'\}} def(t)$. In particular, the scope, con', of the projection constraint contains the variables that con and I have in common, and thus $con' \subseteq con$. Moreover, the preference associated to each assignment to the variables in con', denoted with t', is the highest ($\sum$ is the additive operator of the c-semiring) among the preferences associated by def to any completion of t', t, to an assignment to con.

Many known classes of satisfaction or optimization problems can be cast in this formalism. For example, a classical CSP is just an SCSP where the chosen c-semiring is: $S_{CSP} = \langle \{false, true\}, \vee, \wedge, false, true \rangle$. In fact, constraints can only be either satisfied (true) or violated (false), the logical and models the fact that we want all the constraints to be satisfied, and the logical or models the fact that we prefer true to false.

Fuzzy CSPs can be modeled in the SCSP framework by choosing the c-semiring: $S_{FCSP} = \langle [0, 1], max, min, 0, 1 \rangle$. This means that preferences are values between 0 and 1. The max operator shows that we prefer higher values to lower ones. The min operator says that, when we combine preferences of several constraints, we take the lowest value. Thus in fuzzy CSPs we want to maximize the minimum preference. This is a pessimistic approach to preference handling, that works well in application domains where one needs to be very cautious, such as medical or space applications.

For weighted CSPs, the semiring is $S_{WCSP} = \langle \Re^+, min, +, +\infty, 0 \rangle$. Here preferences are interpreted as costs from 0 to $+\infty$, which are combined with the sum and compared with min. Thus the optimization criterion is to minimize the sum of the costs.

Given an assignment s to all the variables of an SCSP $P = \langle C, V, D \rangle$, we denote by $pref(s, P)$ the preference of s in P, defined as $pref(s, P) = \prod_{<def, con> \in C} def(s_{\downarrow con})$. In words, it is obtained by taking the combination of the preferences associated to the sub-tuples corresponding to the solution in the constraints. A complete assignment of values to all the variables is an *optimal solution* if its preference is the best one w.r.t. the ordering induced by the additive operator. Thus, if we are working with fuzzy CSPs, its preference value must be the highest one, and if we are working with weighted CSPs, its cost must be the lowest. Given an SCSP P, we denote with $Opt(P)$ the set of all the optimal solutions of P.

[1] By $t \downarrow_Y^X$ we mean the subtuple obtained by projecting the tuple t (defined over the set of variables X) over the set of variables $Y \subseteq X$.

Semiring-based soft constraints model optimization problems by using preferences in the constraints and combining them via the semiring combination operator. This induces an ordering over the solutions of the problem, which can be seen as an objective function. Thus soft constraints can model all objective functions that are decomposable over the topology of the problem.

Techniques used to find optimal solutions of constraint optimization problems can be divided into search-based schemes and inference-based schemes [2]. The most common search-based algorithm for constraint optimization is *Branch and Bound* (BB) [2,11]. On the other hand, a very general inference-based algorithm is *Bucket elimination* (BE) [5], which may be seen as an extension of adaptive consistency [2] to optimization problems. Given a linear order over the variables, in the bucket processing phase, each variable is considered, in one direction of the order, and it is removed by projecting the combination of the constraints involving it over all the other variables in such constraints. After all the variables (but one) have been eliminated, in the forward phase, the variables can be assigned, following the other direction of the order, and an optimal solution can be found in polynomial time.

Both techniques have an exponential worst time case complexity. BE, in contrast with BB, also needs possibly exponential space, but it can exploit the graph structure of the problem. For some structures, such as problems with tree-shaped constraint graphs, optimal solutions can be found in polynomial time [2]. Recent extension of Branch and Bound strategies that explore the AND/OR search space of a graphical model were shown to allow similar complexity bounds to inference-based schemes [4,10].

3 Unstable Optimization Problems

We define unstable SCSPs as SCSPs where there may be some unstable preferences. Such preferences are specified by a default value d plus an an interval $[l, u]$, that contains all possible values that can replace the default value d.

Definition 1 (unstable soft constraint). *Given a set of variables V with finite domain D, and a c-semiring $S = \langle A, +, \times, 0, 1\rangle$, an unstable soft constraint is a pair $\langle f, con\rangle$ where $con \subseteq V$ is the scope of the constraint and the preference function of the constraint is $f : D^{|con|} \longrightarrow A \times I$, s.t. $t \mapsto (d, [l, u])$, where I is a set of all the intervals of values in A and $l \leq_S d \leq_S u$. All tuples mapped into $(d, [l, u])$ where $l <_S u$ (resp., $l = u$) are called unstable (resp., stable) tuples and their preference d is called an unstable (resp., stable) preference.*

In what follows, when there is a stable preference, instead of writing $(d, [d, d])$ we will simply write d. Also, when it is clear from the context, we will omit the semiring name and we will write $\leq$ instead of $\leq_S$. Notice that $\leq_S$ is not always the usual $\leq$ over naturals or reals. In fact, if we are dealing with costs, where higher means worst in the semiring, we have that $c_1 \leq_S c_2$ when $c_2 \leq c_1$.

As an example of an unstable constraint using the fuzzy c-semiring $\langle [0, 1], max, min, 0, 1\rangle$, consider $V = \{X, Y\}$, $D = \{a, b\}$, $con = V$, and preference function

$f(X = a, Y = a) = 0.1$, $f(X = a, Y = b) = (0.5, [0.3, 0.6])$, $f(X = b, Y = a) = 0.6$, $f(X = b, Y = b) = (0.7, [0.5, 1])$.

Instead, as an example of an unstable constraint using the weighted c-semiring $\langle \Re^+, min, +, +\infty, 0 \rangle$, we can consider a constraint with $con = V$ and cost function $g(X = a, Y = a) = 100$, $g(X = a, Y = b) = (50, [60, 20])$, $g(X = b, Y = a) = 80$, $g(X = b, Y = b) = (30, [50, 10])$. Notice that according to the ordering induced by the weighted c-semiring, we have, for example, $50 < 10$, since 10 is better than 50 in the weighted semiring.

Definition 2 (unstable SCSP). *An unstable SCSP (USCSP) is a tuple* $\langle S, V, D, C \rangle$*, where* V *is a set of variables with domain* D *and* C *is a set of unstable soft constraints over the variables in* V *over the c-semiring* S*.*

An USCSP where all the preferences are stable corresponds to an SCSP.

Definition 3 (solution). *A solution of an USCSP is an assignment to all its variables.*

We now introduce our running example. Consider the problem related to building a piece of furniture with some iron. Assume that for iron we may have high, medium, or bad quality, with costs 50, 30, and 20. We also assume that the processing time for the piece of furniture is 2 or 3 days, and that the processing cost depends on the quality of the iron and on how many work days are needed. This problem can be modelled by an USCSP over the weighted c-semiring with:

- two variables Q_i and T representing the quality of the iron and the processing time, with domains $D(Q_i) = \{b, m, h\}$ and $D(T) = \{2, 3\}$;
- an unstable soft constraint on Q_i with cost function f_i defined by $f_i(b) = 20$, $f_i(m) = 30$, and $f_i(h) = 50$;
- an unstable soft constraint on Q_i and T, with cost function f defined by $f(h, 2) = 10$, $f(h, 3) = 20$, $f(m, 2) = (30, [60, 5])$, $f(m, 3) = (35, [100, 20])$, $f(b, 2) = 80$, $f(b, 3) = 100$. Thus, for example, if the iron is of bad quality, the processing cost is 80 if the work is done in 2 days, and 100 is it is done in 3 days. Also, if the quality is medium, and the work is done in 2 days, we expect the processing cost to be 30. However, this value may change in the range $[60, 5]$. Similarly, if the work is done in 3 days, we expect the processing cost to be 35, but it can change in the range $[100, 20]$. A solution is, for example, $(Q_i = h, T = 3)$: high quality iron is used and three days of work are needed.

Clearly, not all solutions are equally desirable. In order to discriminate among them, we will define some optimality notions for USCSPs, as well as algorithms to handle them. To do this, we start by giving some basic notions which will be useful in what follows.

Given an USCSP P, a *scenario of* P is an SCSP obtained from P by replacing every unstable preference with a value in its range. $SC(P)$ denotes the set of all possible scenarios of P.

In terms of defining the notions of optimality, a special role will be played by the *default scenario*, denoted by P_d, where only default values are considered. Such a scenario represents the problem given by the user when instability is ignored. Moreover, it will be useful to consider the *worst scenario*, denoted by P_l, where only the worst elements in the ranges are considered, and the *best scenario*, denoted by P_u, where only the best elements in the ranges are considered. We will also denote the preference value of the optimal solutions of P_d, P_l, and P_u, by, respectively, $pref_d$, $pref_l$, and $pref_u$.

In the running example, we have that $pref_d = 60$, obtained by solutions $(Q_i = m, T = 2)$ and $(Q_i = h, T = 2)$. Also, $pref_u = 35$, obtained by solution $(Q_i = m, T = 2)$. Finally, $pref_l = 60$, obtained by $(Q_i = h, T = 2)$.

4 Optimal and Optimality-Robust Solutions (O-ROB)

The first kind of solutions that we consider are optimal solutions that are robust w.r.t. optimality. This means that their status of being optimal does not change, regardless of any variation of the unstable preference values within their ranges.

Such a notion of optimality is useful when it is necessary to adopt a safe attitude: we want our decision to be optimal no matter what happens to the unstable parts of the problem.

Definition 4 (O-ROB). *Given an USCSP P, a solution is in O-ROB(P)) iff*

- *it is optimal in P_d, and*
- *it is optimal in all other $P' \in SC(P)$.*

Note that these solutions were called necessarily optimal in [8], that considers problems where every interval is the largest one, and there were no default values. While considering any range adds expressiveness, the presence of default values is not important for this notion of optimality. In fact, the above definition could easily be replaced by an equivalent one (more compact but less easy to relate to the problem definition) where we only require s to be optimal in all $P' \in SC(P)$. In fact, P_d is just one of the problems in $SC(P)$.

Proposition 1. *Given an USCSP P, the set O-ROB(P) may be empty.*

In fact, in the running example, the optimal solutions of P_d are $(Q_i = m, T = 2)$, and $(Q_i = h, T = 2)$. However, $(Q_i = m, T = 2)$ is not optimal in P_l, and $(Q_i = h, T = 2)$ is not optimal in P_u. Thus O-ROB(P) is empty.

Algorithm 1 shows a procedure to find solutions in O-ROB(P).

Find-OROB takes in input an USCSP P and returns either a solution in O-ROB(P), or nil. To do this, it computes an optimal solution s_l of P_l and an optimal solution s_u of P_u, with preferences $pref_l$ and $pref_u$. This can be done via any of the solving techniques for SCSPs (denoted with *Solve* in the pseudocode). Then, if $pref_l = pref_u$, Find-OROB returns the optimal solution of P_l, otherwise it returns nil. It is possible to show that Algorithm Find-OROB is sound, but not complete. Thus, if it returns a solution, it is in O-ROB(P). If instead it returns nil, this does not necessarily mean that O-ROB(P) is empty.

Algorithm 1. Find-OROB

Input: an USCSP P; **Output**: a solution or nil
$(s_l, pref_l) \leftarrow Solve(P_l)$; $(s_u, pref_u) \leftarrow Solve(P_u)$
if $pref_l = pref_u$ **then**
  return s_l
else
  return *nil*

Theorem 1. *Given an USCSP P, if Find-OROB(P)* $= s_l$*, then* $s_l \in$ *O-ROB(P).*

Proof. Assume Find-OROB(P) $= s_l$. Due to the monotonicity of the combination operator off the semiring, the preference of s_l in any scenario can only be higher than, or equal to, its value $pref_l$ in P_l. Since $pref_l = pref_u$, this means that whatever preference values are assigned to unstable tuples within their ranges, the preference of s_l is always $pref_l$ and the preference of any other solution is never greater than $pref_l$. Thus s_l is optimal in every scenario, and therefore it is in O-ROB(P). Q.E.D.

If Find-OROB(P) $= nil$, consider a very simple USCSP P with one variable X with domain $\{a, b, c\}$ and with a unary unstable constraint over X with cost function $f(a) = (10, [20, 5])$, $f(b) = f(c) = 30$. It is easy to see that $pref_u = 5$, $pref_l = 20$, but $(X = a)$ is in O-ROB(P). Thus Find-OROB(P) $= nil$ but O-ROB(P) is not empty.

Notice that, if $pref_l = pref_u$, not every solution of P_u is in O-ROB(P), since there might be ways to set the unstable preferences that make that solution not optimal in some scenarios. This is why we can only take the solutions in P_l.

Algorithm Find-OROB is therefore sound and not complete. However, finding a solution in O-ROB(P) with algorithm Find-OROB requires just solving two optimization problems. In [8] it is shown that this approach is both sound and complete when we restrict the preference ranges to be all equal to the $[\mathbf{0}, \mathbf{1}]$ interval, where $\mathbf{0}$ and $\mathbf{1}$ are the worst and the best preference values, and $pref_l > \mathbf{0}$.

5 Optimal and Preference-Robust Solutions (P-ROB)

Another kind of optimal solutions that we consider are those that are robust w.r.t. their preferences. That is, solutions that are optimal in the default scenario, and that do not require additional cost if the scenario changes. However, they could loose their optimality status if the scenario changes.

This notion is useful when we act under severe cost restrictions: we would like our decisions to be optimal at least in the default scenario, and be sure that no additional cost is needed if the unstable costs turn out to be different from the default ones.

Definition 5 (P-ROB). *Given an USCSP P, a solution s is in P-ROB(P) iff*

- *it is optimal in* P_d *and*
- $\forall P' \in SC(P)$, $pref(s, P') \geq pref(s, P_d) = pref_d$.

In words, a solution is in P-ROB(P) iff it is optimal in P_d and its preference ($pref_d$) may only improve if the scenario changes. Such solutions are interesting whenever the optimal cost of the default problem is attractive, and we want to make sure that in all other scenarios no additional cost will be required.

In the running example, among the optimal solutions of P_d, only $(Q_i = h, T = 2)$ is in P-ROB(P).

Proposition 2. *Given an USCSP P, P-ROB(P) may be empty.*

In fact, if we consider the USCSP R obtained from the USCSP defined in the running example by changing the stable preference of $(Q_i = h, T = 2)$ from 10 to 20, the only optimal solution of R_d is $(Q_i = m, T = 2)$, but its preference worsens in R_l. Thus P-ROB(R) is empty.

Algorithm 2 shows a method which, given in input a USCSP P, returns a solution in P-ROB(P) if there is any.

Algorithm 2. Find-PROB

Input: an USCSP P; **Output**: a solution, or nil
$(s_l, pref_l) \leftarrow Solve(P_l)$; $(s_d, pref_d) \leftarrow Solve(P_d)$
if $pref_d = pref_l$ **then**
  return s_l
else
  return *nil*

We will now prove that Algorithm Find-PROB is sound and complete.

Theorem 2. *Given an USCSP P, if Find-PROB(P)=s then $s \in$ P-ROB(P) and if Find-PROB(P)=nil then P-ROB(P)=$\emptyset$.*

Proof. Assume Find-PROB(P) returns a solution s. This happens if and only if s is optimal in P_l. Since $pref_d = pref_l$, and due to monotonicity of the multiplicative operator of the c-semiring, s is optimal also in P_d. Again due to monotonicity, $\forall P' \in SC(P)$, $pref(s, P_d) = pref(s, P_l) \leq pref(s, P')$. Thus, by definition, $s \in$ P-ROB(P).

If Find-PROB(P) returns *nil* then $pref_d > pref_l$, that is, for any optimal solution s of P_d, $pref(s, P_d) > pref(s, P_l)$. This means that P-ROB($P$)=$\emptyset$. Q.E.D.

To find a solution in P-ROB(P) with algorithm Find-PROB, it is enough to solve two optimization problems. This was true also for Find-OROB, but Find-PROB is both sound and complete.

6 Optimality-Robust and Preference-Robust Solutions (OP-ROB)

A solution is robust w.r.t. to both optimality and preferences if it is optimal in the default scenario, and both its optimality status and its cost do not worsen if the scenario changes. This is the strongest and most desirable notion.

This notion of optimality is useful when we have both cost restrictions and stringent user requirements: the user wants a solutions which is optimal no matter what, and the company wants to make sure that there is no additional costs if a scenario different from the default one occurs.

Definition 6 (OP-ROB). *Given a USCSP P, a solution $s \in$ OP-ROB(P) iff*

- *it is optimal in P_d,*
- *it is optimal in all other $P' \in SC(P)$, and*
- *$\forall P' \in SC(P)$, $pref(P', s) \geq pref(P_d, s)$.*

It is easy to see that a solution is in OP-ROB(P) iff it is in O-ROB(P) $\cap$ P-ROB(P). In the running example, since we have shown in Section 4 that OROB(P) is empty, also OP-ROB(P) is empty.

Proposition 3. *Given an USCSP P, the set OP-ROB(P) may be empty.*

This follows immediately from the fact that O-ROB(P) $\cap$ P-ROB(P) = OP-ROB(P) and that both O-ROB(P) and P-ROB(P) may be empty.

To find such solutions, we combine the two procedures Find-OROB and Find-PROB as shown in Algorithm 3.

Algorithm 3. Find-OPROB

Input: an USCSP P; **Output**: a solution s, or nil
$(s_d, pref_d) \leftarrow Solve(P_d)$; $(s_l, pref_l) \leftarrow Solve(P_l)$; $(s_u, pref_u) \leftarrow Solve(P_u)$
if $pref_d = pref_l = pref_u$ **then**
  return s_l
else
  return *nil*

Algorithm Find-OPROB, given in input an USCSP P checks if $pref_d = pref_u = pref_l$. If this is so, it returns an optimal solution of P_l, otherwise it returns *nil*. This method is sound but not complete, as shown in the following theorem.

Theorem 3. *Given an USCSP P, if Find-OPROB(P) = s, then $s \in$ OP-ROB(P). If Find-OROB(P)=nil, then OP-ROB(P) might be not empty.*

Proof. If Find-OPROB(P)=s, $pref_d = pref_u = pref_l$. By Theorem 1, $s \in$ O-ROB(P). Also, by Theorem 2, $s \in$ P-ROB(P). Thus $s \in$ O-ROB(P) $\cap$ P-ROB(P) = OP-ROB(P).

In order to show that, when Find-OROB(P)=*nil*, OP-ROB(P) might be not empty, let us consider an USCSP P with one variable X with domain $\{a, b, c\}$ and with a unary unstable constraint over X with cost function $f(a) = (20, [20, 5])$, $f(b) = f(c) = 30$. It is easy to see that $pref_u = 5$, $pref_l = pref_d = 20$, but $(X = a)$ is in OP-ROB(P). Thus Find-OROB(P)=*nil* but OP-ROB(P) is not empty. Q.E.D.

Finding a solution in OP-ROB(P) using algorithm Find-OPROB amounts to solving three SCSPs.

7 The Best Preference-Robust Solutions (Best-ROB)

Solutions in P-ROB(P) are optimal in the default scenario and their cost never increases if the scenario changes. If the main focus is avoiding additional costs when the scenarios changes, rather than the optimality in the default scenario, we can relax the first requirement. The set of solutions of this kind will be denoted by Best-ROB(P). A solution s is in Best-ROB(P) if its cost in P_d can only decrease by changing the scenario. Also, among the solutions with such a property, it is the one with lowest cost in P_d.

Thus, a solution in Best-ROB(P) could be non-optimal in the default scenario. However, there is no better solution in Best-ROB(P) whose cost does not increase in some other scenarios.

Solutions of this kind are useful, for example, when budget limitations guide the operations of a company more than solutions quality. In fact, such solutions assure that no additional cost is needed, although they may sacrifice solution optimality to achieve this.

Definition 7 (Best-ROB). *Given an USCSP P, a solution $s \in$ Best-ROB(P) iff*

- $s \in F = \{s |\ pref(s, P_d) > \mathbf{0}$ *and* $\forall P' \in SC(P),\ pref(s, P') \geq pref(s, P_d)\}$ *and*
- $\forall s' \in F,\ pref(s, P_d) \geq pref(s', P_d)$.

Notice that, in general, if P-ROB(P) $\neq \emptyset$, then Best-ROB(P) = P-ROB(P). This is the case of our running example, where both sets contain only solution $(Q_i = h, T = 2)$.

Proposition 4. *Given an USCSP P, the set Best-ROB(P) may be empty.*

To see this, let us consider the USCSP P with one variable X with domain $\{a, b, c\}$ and with a unary unstable constraint over X with cost function $f(a) = f(b) = f(c) = (10, [20, 5])$. In such a case, all solutions have a cost in P_l which is strictly higher than that in P_d. Thus Best-ROB(P)=$\emptyset$.

Algorithm 4 shows the procedure for the strictly monotonic case which uses an SCSP denoted with P_{fix}. P_{fix} is obtained from the USCSP P in input by just fixing the unstable preferences as follows: for each unstable preference $(d, [l, u])$ in P, we put in P_{fix}, $\mathbf{0}$ if $l < d$, and d otherwise. The intuition behind the construction of P_{fix} is to forbid those tuples associated to preferences that may worsen w.r.t. their default values when the scenario changes.

Find-BestROBm checks if SCSP P_{fix} has a solution with preference strictly better than $\mathbf{0}$. If so, it returns an optimal solution of P_{fix}, otherwise it returns *nil*. This algorithm to find solutions in Best-ROB(P) is both sound and complete.

Theorem 4. *Given an USCSP P over a strictly monotonic c-semiring, if Find-BestROBm(P) = s, $s \in$ Best-ROB(P). Find-BestROBm(P)=nil iff Best-ROB(P)=$\emptyset$.*

Algorithm 4. Find-BestROBm

Input: an USCSP P with a strictly monotonic c-semiring; **Output**: a solution s, or nil
$(s, p) \leftarrow Solve(P_{fix})$
if $p > \mathbf{0}$ **then**
  return s
else
  return nil

Proof. Find-BestROBm$(P) = s$ iff $p > \mathbf{0}$. We need to show that $Opt(P_{fix}) \subseteq$ Best-ROB(P). By construction of P_{fix}, and due to the strict monotonicity of the combination operator, we have that, $\forall s' \in Opt(P_{fix})$, if $pref(s', P_{fix}) > \mathbf{0}$, then $pref(s', P_{fix}) = pref(s', P_d)$ and, $\forall P' \in SC(P)$, $pref(s', P') \geq pref(s', P_d)$. Thus, since $s \in Opt(P_{fix})$, it satisfies this property and there is no solution with a higher preference in P_d satisfying it. This means that $s \in$Best-ROB(P).

Find-BestROBm$(P) = nil$ iff $p = \mathbf{0}$, that is, iff all solutions of P_{fix} have preference $\mathbf{0}$. Thus, for all s such that $pref(s, P_d) > \mathbf{0}$, $s \notin$ Best-ROB(P). The other solutions, that have preference $\mathbf{0}$ in P_d, are not in Best-ROB(P) by definition. Thus Best-ROB(P) $= \emptyset$. Q.E.D.

This theorem shows that algorithm Find-BestROBm is both sound and complete. Finding solutions in Best-ROB(P) using this algorithm (that is, when the combination operator is strictly monotonic) amounts at solving one SCSP. However, this algorithm works only when the combination operator is strictly monotonic. It is possible to define a sound but not complete approach that works for any c-semiring.

Consider applying BE to scenarios P_d and P_l using the same variable ordering. At the end of the bucket processing phase [2] in both scenarios, we obtain two new SCSPs, say P'_d and P'_l, where there are additional constraints and possibly lower preferences in the old constraints. The first variable in the linear order, say x, has each value, say a, in its domain associated with the highest preference (or lowest cost) of a solution of the corresponding scenario where $x = a$.

We then check if there are values for x that have the same preference in both P'_l and P'_d. If this is not the case, we are not able to say anything about set Best-ROB(P). Otherwise, we pick among such values one, say a, with the highest preference, say p. Using the forward step of BE applied to $x = a$ in P'_l, assignment $x = a$ can be extended to a solution of P'_l (and thus of P_l) with preference p. Due to the monotonicity of the combination operator and to the fact that $x = a$ has the same preference in P'_l and P'_d, such a solution has the same preference p also in P'_d (and thus in P_d). Moreover, there is no solution with this property and a higher preference in P_d. This means that such a solution is in Best-ROB(P).

This algorithm can always be used, but it is possibly not complete. It requires to solve two SCSPs with BE to find a solution in Best-ROB(P).

8 The Most Preference-Robust Optimal Solutions (ROB-OPT)

Another way to be more tolerant with the conditions of P-ROB(P) is to relax the second requirement, that is, to maintain optimality in the default scenario but to allow for a decrease in the preference if the scenario changes. However, such a decrease should be the smallest possible in the worst scenario. This set of solution is called ROB-OPT(P).

Definition 8 (ROB-OPT). *Given an USCSP P, a solution $s \in$ ROB-OPT(P) iff*

- *it is optimal in P_d, and,*
- *for every other optimal solution of P_d, say s', $pref(s, P_l) \geq pref(s', P_l)$.*

In words, a solution is in ROB-OPT(P) if it is optimal in the default scenario and, among the solutions that are optimal in such a scenario, its preference value decreases the least in the worst scenario.

Contrarily to all the previous notions of optimality, this set always contains at least a solution. In general, if P-ROB(P) $\neq \emptyset$, P-ROB(P) = ROB-OPT(P). This is the case of our running example, where ROB-OPT(P) = $\{(Q_i = h, T = 2)\}$.

Proposition 5. *Given an USCSP P, ROB-OPT(P) is never empty.*

It follows immediately from the fact that the set of optimal solutions of P_d is never empty.

To find solutions in ROB-OPT(P), we propose a procedure based on defining a new SCSP. Given an USCSP P, defined over the c-semiring $S = \langle A, +, \times, 0, 1\rangle$, we consider the SCSP P_{dl} with the same variables and constraint topology as P, and defined over the c-semiring $S' = \langle A \times A, lex(+, +), (\times, \times), (\mathbf{0}, \mathbf{0}), (\mathbf{1}, \mathbf{1})\rangle$. In such a c-semiring, preferences are pairs which are combined by applying the combination operator to the corresponding components and which are ordered lexicographically with the first component being the most important one. In P_{dl}, each tuple associated with preference $(d, [l, u])$ in P is instead associated with preference (d, l). The intuition behind the definition of P_{dl} is that, by solving it, we find solutions which in the first place maximize the combination of the default preferences, and secondly maximize the combination of the lower bounds of the ranges of the unstable preferences.

Theorem 5. *Given an USCSP P and a solution s = Find-ROBOPT(P), $s \in$ ROB-OPT(P).*

Algorithm 5. Find-ROBOPT

Input: an USCSP P; **Output**: a solution s
$(s, p) \leftarrow Solve(P_{dl})$
return s

Proof. Since Find-ROBOPT(P) always returns an optimal solution of P_{dl}, it is sufficient to prove that $Opt(P_{dl}) = \text{ROB-OPT}(P)$. We first show that $Opt(P_{dl}) \subseteq \text{ROB-OPT}(P)$. Notice that, by construction of P_{dl}, if a solution s has preference (d, l) in P_{dl}, then in P_d it has preference d and l in P_l. Given a solution $s \in Opt(P_{dl})$ with preference (d, l), for every other solution s' of P_{dl} with preference (d', l'), it must be that $d \geq d'$. Thus, $s \in Opt(P_d)$. Moreover, for every other solution s'' with preference (d, l''), it must be that $l \geq l''$. Thus $s \in \text{ROB-OPT}(P)$.

Next we show that $\text{ROB-OPT}(P) \subseteq Opt(P_{dl})$. If $s \in \text{ROB-OPT}(P)$, it is optimal in P_d and thus, in P_{dl}, it is among the solutions with a maximal first component. Moreover, among the optimal solutions of P_d, s is one of the solutions with the highest preference in P_l. This means that among those with the maximal first component in P_{dl}, it has the highest second component. This corresponds to being undominated w.r.t. the ordering induced by $lex(+, +)$. Q.E.D.

9 USCSPS with Uncontrollable Variables

In unstable SCSPs all the variables are decided by the deciding agent. The only form of uncertainty is the presence of the preference ranges around the default values. However, in many real-life settings, there are also variables which are *uncontrollable*, that is, their value cannot be chosen by the deciding agent. Such a value will be decided by some other agent or by Nature. Typical examples are times of events related to weather changes. For example, we don't know when the clouds will disappear.

We will now consider the presence of some of this kind of variables in an unstable SCSP. In this generalized setting, the variables V of the problem will be partitioned in two sets, namely V_c and V_u, containing, respectively, the controllable and the uncontrollable variables. In this paper we assume to have no information on the uncontrollable variables besides their domain values. For example, we don't have a probability, and not even a possibility, distribution over such a domain.

There are many ways to reason with uncontrollable variables, which depend on the attitude to risk of the agent. Examples are the notions of strong, weak, and dynamic controllability in temporal reasoning [12]. Here we adopt a pessimistic approach (which follows the same principle as for strong controllability), where we want to make sure that the cost of the decision we take over the controllable part of the problem is guaranteed not to increase when the uncontrollable variables are instantiated.

To achieve this, we first *eliminate* the uncontrollable variables one at a time in a linear order. For each variable v in V_u, we consider all constraints $c_1, \ldots, c_n$ connecting v to other variables, say $v_1, \ldots, v_m$, and we build a new constraint c connecting $v_1, \ldots, v_m$, whose preference function f is defined as follows: $f(d_1, \ldots, d_m) = \Pi_{d \in D(v)} \Pi_{i=1}^{n} l_i(t_i, d)$, where d_i is the subtuple of $(d_1, \ldots, d_m)$ involving only values for the variables in $con(c_i)$, and $l_i(t_i, d)$ is the lower element of the range associated to tuple (t_i, d) by the preference function of constraint

c_i. In words, we associate to tuple $(d_1, \ldots, d_m)$ the greatest lower bound of the preferences that can be obtained by extending this tuple to any value in the domain of v.

Notice that this procedure can be seen as a variant of BE where we take the worst case rather than the best one (as is done in the projection step).

Uncontrollable variables are eliminated one at a time, until only controllable variables are left. Given an USCSP P with uncontrollable variables, we denote by $cont(P)$ the resulting USCSP obtained by applying this procedure to P.

Theorem 6. *Consider an USCSP P with controllable variables V_c and uncontrollable variables V_u. For any assignment s to the variables in V_c, and any assignment s' to the variables in V_u, $pref(s, cont(P)) \leq_S pref((s, s'), P)$, where S is the c-semiring over which P is defined.*

Proof. It follows by monotonicity and intensivity of the $\times$ operator of the semiring. Q.E.D.

We can therefore reason on an unstable SCSP P with uncontrollable variables by first eliminating all uncontrollable variables, thus obtaining $cont(P)$, and then by reasoning on $cont(P)$ according to any one of the optimality/robustness criteria defined in the previous sections. No matter what solution we end up with, we are sure that no additional cost will be needed when the values of the uncontrollable variables will be known. For example, if we choose to use O-ROB, we find a solution of the controllable part which is optimal for the default scenario and remains optimal even if the unstable preferences change, and whose preference level cannot decrease because of how Nature decides to instantiate the uncontrollable variables.

10 Final Considerations and Future Work

For most of the notions of optimality considered in this paper, for which we give sound and complete algorithms, finding an optimal solution for an unstable SCSP P requires solving at most three SCSPs. This means that, to handle the kind of uncertainty modelled by preference ranges and default values, we don't change the complexity class. In particular, for example, if the default problem belongs to a tractable SCSP class, this is also true for any unstable SCSP with the same topology.

We did not consider probability distributions over the ranges of the possible values for the unstable costs. We believe there are several application domains where probabilistic reasoning is not suitable, and one would rather prefer to reason with exact, although unstable, information. However, we also envision domains where it makes sense to consider the expected utility of a solution or a scenario, and to take decisions based on such concepts. We plan to study this adaptation of our work.

We also plan to implement the algorithms to obtain the several notions of optimal solutions defined in this paper, and also to test experimentally how

many times those notions that may return an empty set actually do this. While these notions seems to be less appealing because there may be none of them, it may be that in certain application domains, or in classes of problems with a certain structure, there are always some of them.

References

1. Bistarelli, S., Montanari, U., Rossi, F.: Semiring-based constraint solving and optimization. Journal of the ACM 44(2), 201–236 (1997)
2. Dechter, R.: Constraint processing. Morgan Kaufmann, San Francisco (2003)
3. Dechter, R., Dechter, A.: Belief maintenance in dynamic constraint networks. In: AAAI, pp. 37–42 (1988)
4. Dechter, R., Mateescu, R.: And/or search spaces for graphical models. AI Journal 171(2-3), 73–106 (2007)
5. Dechter, R.: Bucket elimination: A unifying framework for reasoning. AI Journal 113(1-2), 41–85 (1999)
6. Faltings, B., Macho-Gonzalez, S.: Open constraint programming. AI Journal 161(1-2), 181–208 (2005)
7. Fargier, H., Lang, J., Schiex, T.: Mixed constraint satisfaction: a framework for decision problems under incomplete knowledge. In: Proceedings of the 13th National Conference on Artificial Intelligence (AAAI 1996), vol. 1, pp. 175–180. AAAI Press, Menlo Park (1996)
8. Gelain, M., Pini, M.S., Rossi, F., Venable, K.B.: Dealing with incomplete preferences in soft constraint problems. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 286–300. Springer, Heidelberg (2007)
9. Lamma, E., Mello, P., Milano, M., Cucchiara, R., Gavanelli, M., Piccardi, M.: Constraint propagation and value acquisition: Why we should do it interactively. In: IJCAI, pp. 468–477 (1999)
10. Mateescu, R., Dechter, R.: A comparison of time-space schemes for graphical models. In: Proc. IJCAI 2007, pp. 2346–2352. Morgan Kaufmann, San Francisco (2007)
11. Rossi, F., Van Beek, P., Walsh, T. (eds.): Handbook of Constraint Programming. Elsevier, Amsterdam (2006)
12. Vidal, T., Fargier, H.: Handling contigency in temporal constraint networks. JETAI 11(1), 23–45 (1999)
13. Wilson, N., Grimes, D., Freuder, E.C.: A cost-based model and algorithms for interleaving solving and elicitation of csps. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 666–680. Springer, Heidelberg (2007)
14. Yorke-Smith, N., Gervet, C.: Certainty closure: A framework for reliable constraint reasoning with uncertainty. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, pp. 769–783. Springer, Heidelberg (2003)