

Branch-and-Bound for Soft Constraints Based on Partially Ordered Degrees of Preference

Nic Wilson¹ and H el ene Fargier²

Abstract. The handling of partially ordered degrees of preference can be important for constraint-based languages, especially when there is more than one criterion that we are interested in optimising. This paper describes branch and bound algorithms for a very general class of soft constraint optimisation problems. At each node of the search tree, a propagation mechanism is applied which generates an upper bound (since we are maximising) of the preference degrees of all complete assignments below the node. We show how this propagation can be achieved using an extended mini-buckets algorithm. However, since the degrees of preference ordering are only partially ordered, such an upper bound can be very uninformative, and so it can be desirable to instead generate an upper bound *set*, which contains an upper bound for the degree of preference for each complete assignment below the node. It is shown how such propagation can also be achieved using this extended mini-buckets approach.

1 Introduction

Degrees of preference can be partially ordered in many situations, especially where there is more than one criterion we are interested in optimising. For example, consider a situation where we are trying to optimise two criteria such as time and money. $(-3, 4)$ might mean that the choices imply that the job will take 3 days and lead to a net gain of 4K euros. A common standard ordering is the Pareto ordering: so that $(a_1, a_2) \leq (b_1, b_2)$ if and only if $a_1 \leq b_1$ and $a_2 \leq b_2$. However, the Pareto ordering often seems excessively weak. A natural way to remedy this is to add trade-offs. For example, the user might indicate that they prefer $(-5, 6)$ to $(-4, 1)$. We can then define a new ordering by adding a set of such extra trade-offs to the Pareto ordering, and generating its transitive closure.

In a system of soft constraints based on partially ordered preferences, each soft constraint evaluates a solution by a degree of preference taken on a partially ordered scale $(A, <)$ (the higher a w.r.t. $<$, the better). The global evaluation of the solution is obtained by the combination of the individual degrees, and the aim is typically to find a solution which is optimal (i.e. maximal) with respect to $<$.

The basic scheme for obtaining such solutions is a branch-and-bound tree search; at each node, some form of propagation is used to generate upper bound information (since we are maximising) regarding all the complete assignments extending the partial assignment associated with the node. This is compared with the current best complete assignment; if no such complete assignment can be better than our current best solution then we can backtrack.

However, when a partial order $<$ is used, an upper bound can be very uninformative, since the requirement of a unique covering value can easily generate a very high element as an upper bound. For instance, consider a situation where we are trying to find a single optimal solution with respect to a Pareto ordering; suppose the current best assignment has preference degree $(-3, 4)$, and we find that every complete assignment extending the current partial assignment either has preference degree no better than $(-6, 6)$, or no better than $(-1, 1)$. Then exploring this subtree has no benefit since the current degree of $(-3, 4)$ cannot be beaten. However, if we use a single upper bound then this upper bound is at least $(-1, 6)$, which is better than $(-3, 4)$, and so doesn't justify backtracking; we would then need to explore the subtree, perhaps having to do exponential amount of work (we might even need to instantiate every variable). But if we instead keep a set of two upper bounds $\{(-6, 6), (-1, 1)\}$, meaning that each complete assignment has preference degree either no better than $(-6, 6)$, or no better than $(-1, 1)$, then this gives us enough information to backtrack at this node. Clearly, keeping such an *upper bound set* can sometimes enable much stronger pruning.

In Section 2 we define a framework for soft constraint optimisation problems; we consider a very general definition of soft constraints; in particular, it is very much more general than semiring-based constraints [3], and enables us to make use of user-defined orderings of degrees of preference, for example, adding extra tradeoffs to a Pareto ordering may well lose nice properties such as being a distributive lattice or monotonicity, so we do not assume these. Also in Section 2 we describe the structure of the branch-and bound algorithm. Section 3 describes the approach to pruning domain elements when we are interested in finding a single optimal solution. This involves use of a propagation to generate upper bounds or upper bound sets. Section 4 describes an extended mini-buckets [4] approach for generating an upper bound. Section 5 shows how this approach can be extended to generate upper bound sets. Section 6 considers other optimisation tasks, and Section 7 discusses extensions.

Related work

There is a growing literature on soft constraints based on partially ordered preferences, and related optimisation techniques. The soft constraints framework explored in this paper is somewhat similar in structure to semiring-based CSPs [3], though is much more general. A general propagation framework for semiring-based CSPs have been derived in [3, 2, 1], but mini-buckets propagation has not been explicitly considered, nor has the use of upper bound sets. However, the use of mini-buckets and a kind of upper bound set has been developed by Roll on and Larrosa [10] (see also [5, 12, 11]) for the special case of multiobjective weighted constraint optimisation.

Junker [8, 9] considers multi-criteria optimisation, which corre-

¹ Cork Constraint Computation Centre, Department of Computer Science, University College Cork Cork, Ireland. n.wilson4c.ucc.ie

² IRIT-CNRS, 118 Route de Narbonne 31062 Toulouse Cedex, France. Helene.Fargier@irit.fr

sponds to situations in our framework where the set A of preference degrees has a combinatorial structure. Particular orderings are considered including lexicographic and Pareto orderings. [6, 7] also deal with multi-criteria optimisation especially for the Pareto ordering; [13] consider a soft constraint approach to Pareto optimisation.

2 Branch-and-Bound Tree Search for General Soft Constraint Optimisation

Let \otimes be a commutative and associative operation on a set A . The elements of A will be interpreted as degrees of preference and compared using a transitive and irreflexive relation³ We define \leq in the obvious way: $r \leq s$ if and only if either $r < s$ or $r = s$.

Let V be a set of variables. For $X \in V$ let $D(X)$, the domain of X , be the set of possible value of X . For $U \subseteq V$, define $D(U)$, the domain of U to be $\prod_{X \in U} D(X)$. (An element of $D(U)$ is formally a function on U which maps $X \in U$ to an element of $D(X)$.) Elements of $D(V)$ will be referred to as *complete assignments*.

An *A-constraint* (also called a *soft constraint*) has an associated set of variables V_c , known as its *scope*. c is a function from $D(V_c)$ to A . If $\alpha \in D(U)$ where $U \supseteq d(V_c)$, then we use $c(\alpha)$ as an abbreviation of $c(\alpha \upharpoonright V_c)$, where $\alpha \upharpoonright V_c$ is the projection (i.e., restriction) of α to V_c . For $\alpha \in D(V_c)$, the value $c(\alpha)$ is intended to represent a benefit or cost (or more general preference degree) associated with this soft constraint, when tuple α is chosen. The operation \otimes is used to combine preference degrees and it enables us to combine soft constraints. The combination C_\otimes of C is given by, for $\alpha \in D(V)$, $C_\otimes(\alpha) = \otimes_{c \in C} c(\alpha)$. Its scope V_{C_\otimes} equals $\bigcup_{c \in C} V_c$.

A soft constraint (as defined here) is more general than a c-semiring-based constraint [3]. The neutral element (if it exists) is not required to be the top of the scale A (so, that both costs and benefits can be represented) and the usual kind of monotonicity property for \otimes is not assumed. This allows flexibility in the ordering of preference degrees, which is important if it is based on user inputs, for example, extra trade-offs added to a Pareto ordering.

A *soft constraints system* is defined to be a tuple $\langle V, A, \otimes, <, C \rangle$ where V is a set of variables, A is a set, \otimes is a commutative and associative binary operation on A , $<$ is an irreflexive and transitive relation on A , and C is a multiset of A -constraints.

The general soft constraints optimisation problems. Let $\langle V, A, \otimes, <, C \rangle$ be a soft constraints system. A complete assignment $\alpha \in D(V)$ is said to be *optimal* if there does not exist $\beta \in D(V)$ with $C_\otimes(\beta) > C_\otimes(\alpha)$. We consider the following optimisation problems; for space reasons we will focus mainly on the first of these, with a brief discussion of the second in Section 6.

(SOS): Single Optimal Solution Problem. Find a complete assignment $\alpha \in D(V)$ which is optimal.

(AOS): Find all complete assignments $\alpha \in D(V)$ which are optimal.

The structure of the tree search for SOS

The basic scheme for solving soft constraint optimisation problems is a branch-and-bound tree search. For the Single Optimal Solution problem we maintain a value $best \in A$ which is the preference degree of a best solution α_{best} found so far, so that $C_\otimes(\alpha_{best}) = best$. We can initialise α_{best} to be some arbitrary complete assignment.

³ Relation $<$ being irreflexive means that for all $a \in A$, $a \not< a$. Often an ordering is expressed as a reflexive and transitive relation \preceq ; we can then define $<$ to be the strict part of \preceq so that $r < s$ if and only if $r \preceq s$ and $s \not\preceq r$.

To each node in the search tree is associated a set of variables which have been assigned (in ancestors of the node), along with the associated assignment to those variables. The *root node* has associated set of variables \emptyset . We say that a node is a *leaf node* if its associated assignment is a complete assignment, and so involves all the variables.

When we create a node N which is not a leaf node, we choose a variable X to assign next, and choose an element x of its domain; on backtracking we choose another value of X ; when there are no remaining values of X to choose, we backtrack (up to its parent node). The search finishes on backtracking from the root node.

When we consider value x of variable X we first check if x will be pruned (see Section 3); if so, we move onto the next value of X . If we cannot prune x then we generate (and move down to) a new node (a child of the current node) associated with the previous set of assignments plus assignment $X = x$.

Behaviour at leaf nodes: Consider a leaf node with associated complete assignment α . Let t be the preference degree of α , i.e., $t = C_\otimes(\alpha)$. If $t > best$, we set $best := t$ and set $\alpha_{best} := \alpha$. Otherwise we leave $best$ and α_{best} unchanged. In either case we then backtrack to its parent node.

Associated with each node is a sub-problem: the original collection of soft constraints (partially) instantiated with the assignment associated with a node. More explicitly, let U be the set of uninstantiated variables, and let θ be the assignment of $V - U$ associated with the node. Each soft constraint c generates a soft constraint by instantiating with θ and only considering variables U . That is, from c we generate constraint c_θ with scope $V_{c_\theta} = U \cap V_c$ defined by for $\beta \in D(U \cap V_c)$, $c_\theta(\beta) = c(\theta\beta)$. For example, suppose c has scope $\{X, Y, Z\}$ and allocates preference degree a to assignment $(X = x_2, Y = y_1, Z = z_2)$. If θ assigns $Y = y_1$ and $Z = z_2$ then $c_\theta(X = x_2) = a$.

3 Pruning Domain Values at a Node for SOS

Throughout this section we will be considering a given node, with associated multiset of soft constraints C involving variables U (the uninstantiated variables).

Precise pruning condition. If for all $\alpha \in D(U)$ extending $X = x$ we have $best \not\prec C_\otimes(\alpha)$ then we can prune value x from the domain of X , since no complete assignment below this node can have better preference degree than the current best preference degree, $best$.

Given C and $best$ we say that pruning $X = x$ is *sound* if for all $\alpha \in D(U)$ extending $X = x$, we have $best \not\prec C_\otimes(\alpha)$.

Of course, testing this pruning condition exactly will typically be very hard; we would like sufficient conditions.

Sufficient conditions for pruning

As mentioned in the introduction, this paper we will consider two kinds of upper bounds, single upper bounds and set upper bounds. This calls for a few formal preliminaries. Let \preceq be a pre-order on A (i.e. a reflexive and transitive relation):

- We say that *upper* is an *upper bound function* with respect to \preceq for variable $X \in U$ if $C_\otimes(\alpha) \preceq upper(x)$ for all elements x in the domain of X , and for all $\alpha \in D(U)$ such that $\alpha(X) = x$.
- We say that *SetUpper* is an *upper bound set function* w.r.t. \preceq for variable $X \in U$ if for all $x \in D(X)$ and for all $\alpha \in D(U)$ such that $\alpha(X) = x$, there exists $r \in SetUpper(x)$ such that $C_\otimes(\alpha) \preceq r$. (Note that the notion of upper bound set used in this paper is similar to (though not exactly the same as) the lower bound set defined in [5, 10].)

- For relations \preceq and \preceq' on A we say that \preceq is a *weakening* of \preceq' (and \preceq' is stronger than \preceq) if $\preceq \subseteq \preceq'$, i.e., $r \preceq s$ implies $r \preceq' s$.
- The lower bound relation \preceq_l associated with $<$ is the relation on A defined by $r \preceq_l s$ if and only if every lower bound of r is a lower bound of s , i.e., for all $t \in A$, $[t < r \Rightarrow t < s]$. Analogously we define \preceq_u on A by $r \preceq_u s$ if and only if every upper bound of r is an upper bound of s , i.e., for all $t \in A$, $[s < t \Rightarrow r < t]$. Relations \preceq_l and \preceq_u are both pre-orders. Moreover, if $r < s$ then both $r \preceq_l s$ and $r \preceq_u s$.
- We say that \preceq *respects* \otimes (or, equivalently, “ \otimes is monotone over \preceq ”) if the following condition holds:
for all $r, s, t \in A$, if $r \preceq s$ then $r \otimes t \preceq s \otimes t$

(I) Pruning based on single upper bound. As in classical upper bound based pruning, we could compute upper bounds with respect to \leq . However, we can also use other pre-orders \preceq instead, which is important, for example, when \otimes is not monotone over \leq . The key property we need on \preceq for pruning to be sound is that $r \preceq s$ implies that any lower bound of r is a lower bound of s ; in other words: \preceq is a weakening of \preceq_l .

Given pre-order \preceq which is a weakening of \preceq_l , we generate an upper bound function $upper$ for X . For each $x \in D(X)$ we prune x if $best \not\preceq upper(x)$. If $best \preceq upper(x)$ then pruning $X = x$ is sound given C and $best$ since for any $\alpha \in D(U)$ extending $X = x$, we have $C_\otimes(\alpha) \preceq upper(x)$ which implies $best \not\preceq C_\otimes(\alpha)$, since $best$ is not a lower bound of $upper(x)$. An approach to generating upper bound functions will be described in Section 4.

(II) Pruning based on upper bound set. As mentioned in the introduction, pruning based on a single upper bound may be very inefficient when $<$ is partial. Given \leq , or more generally given any pre-order \preceq which is a weakening of \preceq_l , we generate an upper bound set function $SetUpper$ for X . For each $x \in D(X)$ we prune x if for all $r \in SetUpper(x)$, $best \not\preceq r$, i.e., $best$ is not worse than any element of $SetUpper(x)$. Again, pruning $X = x$ given C and $best$ is sound: consider any $\alpha \in D(U)$ extending $X = x$, and let r be such that $C_\otimes(\alpha) \preceq r$, and so $C_\otimes(\alpha) \preceq_l r$. By assumption, $best \not\preceq r$, and hence $best \not\preceq C_\otimes(\alpha)$.

Generating appropriate relation \preceq from $<$ and \otimes

In Section 4 and 5, we propose a mini-bucket approach for generating upper bound functions and upper bound set functions. Like constraint propagation algorithms and variable elimination techniques, this requires a monotonicity property that is not assumed in the soft constraint framework, i.e., that \preceq respects \otimes .

Therefore, given commutative and associative \otimes on A and transitive and irreflexive $<$ on A we want to generate a pre-order \preceq which respects \otimes and is a weakening of \preceq_l (because of the pruning conditions described above).

Obviously, if \leq respects \otimes then we can use $\preceq = \leq$. More generally, given that A contains a neutral element⁴ we can set \preceq to be \leq defined as follows: For $r, s \in A$, let $r \preceq s$ hold if and only if for all $t, u \in A$, $[u < r \otimes t \Rightarrow u < s \otimes t]$. Relation \preceq is then the strongest relation which is a weakening of \preceq_l and respects \otimes .

Even if \leq respects \otimes , constructing \preceq may give a stronger relation than \leq , and hence cause stronger pruning.

Very often it is valid to use a stronger relation \preceq in the upper bound computations leading to more pruning (and \preceq may even be updated

⁴ That is, an element $1 \in A$ such that for all $a \in A$, $a \otimes 1 = a$. It isn't really restrictive to assume the existence of a neutral element since otherwise we could artificially add one.

during the search): we could modify the definition of \preceq by restricting the range of u to take a current lower bound into account, and also restrict the range of t .

4 Generating an Upper Bound Function for a Multiset of Generalised Constraints

As mentioned in the last section, for pruning at each node in the branch-and-bound algorithm we require a method of generating upper bound functions. That is, we need to be able to solve the following task.

Task A. Let \otimes be a commutative and associative operation on A , and let \preceq be a pre-order on A which respects \otimes . Let C be a multiset of A -constraints over variables U . Let X be a variable in U . For each $x \in D(X)$ generate $upper(x) \in A$ such that for all complete assignments α with $\alpha(X) = x$,

$$\bigotimes_{c \in C} c(\alpha) \preceq upper(x).$$

Mini-buckets [4] is a powerful and flexible approach for propagation. We describe an extended mini-buckets approach for generating an upper bound function, by making use of a procedure implementing the following task.

Task B (Approximate Variable Elimination). Let C be a multiset of A -constraints involving variables W , i.e., $W = \bigcup_{c \in C} V_c$, and let Y be a variable in W (the variable to be eliminated). Generate a multiset of A -constraints, which we label C^{-Y} , involving variables $W - \{Y\}$, such that for all assignments β to $W - \{Y\}$ and all $y \in D(Y)$,

$$\bigotimes_{c \in C} c(\beta y) \preceq \bigotimes_{c \in C^{-Y}} c(\beta),$$

where C^{-Y} is a singleton if $|W| \leq 2$.

Solving Task A given procedure for Task B

Label the variables in U as X_1, \dots, X_n where $X_1 = X$; the idea is to eliminate variables from X_n to X_2 through approximate variable elimination. Let $C_n = C$ and suppose now for some $i \in \{2, \dots, n\}$ we have defined C_i which only involves variables $\{X_1, \dots, X_i\}$. We will use a procedure implementing Task B to generate $C_{i-1} = (C_i)^{-X_i}$ which therefore satisfies the following condition for any assignment β to variables $\{X_1, \dots, X_{i-1}\}$ and all $y \in D(X_i)$: $\bigotimes_{c \in C_i} c(\beta y) \preceq \bigotimes_{c \in C_{i-1}} c(\beta)$, where C_1 is a singleton. Write C_1 as $\{c_1\}$. Using induction, and the transitivity of \preceq , we then have: for all $x \in D(X_1)$ and $\alpha \in D(U)$ such that $\alpha(X_1) = x$, $\bigotimes_{c \in C_n} c(\alpha) \preceq \bigotimes_{c \in C_1} c(x) = c_1(x)$. This shows that c_1 is an upper bound function for X , hence solving solving Task A.

Solving Task B

We use an algorithm of the following form to perform the approximate elimination of variable Y (Task B).

If $|W| > 2$ we do the following:

1. Let $C_Y = \{c \in C : V_c \ni Y\}$ be the soft constraints that involve variable Y . Partition C_Y into G_1, \dots, G_m and for $j = 1, \dots, m$, let U_j denote the set of variables involved in G_j (so U_j contains Y).
2. For each $j = 1, \dots, m$, generate a soft constraint c_j on variables $U_j - \{Y\}$ such that for all $\beta \in D(U_j - \{Y\})$ and $y \in D(Y)$, $\bigotimes_{c \in G_j} c(\beta y) \preceq c_j(\beta)$. Then we set $C^{-Y} = (C - C_Y) \cup \{c_1, \dots, c_m\}$.

If $|W| \leq 2$ we let $C^{-Y} = \{c_1\}$, where c_1 is chosen to have scope $W - \{Y\}$ and so that for all $\beta \in D(W - \{Y\})$ and $y \in D(Y)$, $\bigotimes_{c \in C} c(\beta y) \preceq c_1(\beta)$.

Step 1: choosing a partition. We can use the usual mini-buckets approach to choosing a partition. The basic idea is to limit the size of the product domains associated with each G_j since the complexity is linearly related to this. We can choose a partition such that each U_j has cardinality at most L , for some constant L (i.e., each mini-bucket involves at most L variables). For example, using $L = 2$ leads to a DAC-like propagation. A similar kind of approach, but which takes into account differing domain sizes for variables, is to choose constant M and the partition such that we always have $|D(U_j)| \leq M$. A simple alternative is to choose each G_j to be a singleton, giving propagation which is a little like forward checking. The other extreme is to choose $G_1 = C_Y$, as used in exact variable elimination.

Step 2: generating the constraints c_j : For each β , we thus need to compute $c_j(\beta)$ such that, for each y , $\bigotimes_{c \in G_j} c(\beta y) \preceq c_j(\beta)$. This is managed by use of a procedure implementing Task C below. The case when $|W| \leq 2$ is managed similarly.

Task C . For all $h = 1, \dots, k$ and $i = 1, \dots, l$ let r_h^i be an element of A . Find an element r of A such that for each $i = 1, \dots, l$, $r_1^i \otimes \dots \otimes r_k^i \preceq r$.

(In the context of Step 2, $k = |G_j|$ and $l = |D(Y)|$.)

The following result, which is proved easily, shows that this achieves Task B .

Proposition 1 *With the above definitions, Task B is achieved, i.e., for all assignments $\beta \in D(W - \{Y\})$ and all $y \in D(Y)$, $\bigotimes_{c \in C} c(\beta y) \preceq \bigotimes_{c \in C^{-Y}} c(\beta)$, where C^{-Y} is a singleton set if $|W| \leq 2$.*

Solving Task C

We describe one approach for achieving Task C . Let us assume⁵ that we have access to two functions: firstly $ub(\cdot, \cdot)$ returning an upper bound of its two arguments so that $r, s \preceq ub(r, s)$; and secondly a function $ub_{\otimes}(\cdot, \cdot)$ returning an upper bound of the product of its two arguments so that $r \otimes s \preceq ub_{\otimes}(r, s)$. Very often we will just define $ub_{\otimes}(r, s)$ to be $r \otimes s$. However, below in Section 5 we deal with a situation where combination can be expensive, so we then need to consider an upper approximation.

Task C can clearly be implemented by repeated use of procedures ub and ub_{\otimes} . For each $i = 1, \dots, l$, we apply $k-1$ times the operation ub_{\otimes} to generate an upper bound $r(i)$ of $r_1^i \otimes \dots \otimes r_k^i$. We then apply $l-1$ times the operation ub to generate an upper bound of each $r(i)$, which, by transitivity of \preceq is an upper bound of $r_1^i \otimes \dots \otimes r_k^i$ for each i .

Complexity of propagation with single upper bounds (solving Task A)

The complexity of the procedure depends on the variables involved in the sets U_j . We can choose a positive number M which is at least as large as $|D(V_c)|$ for every $c \in C$. Then we can choose the partitions in the mini-bucket elimination such that we always have $|D(U_j)| \leq M$. Increasing M makes the computation more expensive, but generates stronger propagation, potentially causing more

⁵ The existence of function ub clearly requires that any pair of elements in A have an upper bound with respect to \preceq . However, if we do not have this property then we can enforce it by adding an extra element \top to A , with $a \preceq \top$ for all $a \in A$.

pruning. It can be shown that the number of operations (applications of \otimes or ub) to produce the upper function is then at most $M|C| \times |U|$.

Alternatively, if d is the maximum domain size of any of the variables, we can choose the partitions in the mini-bucket elimination such that we always have $|U_j| \leq L$. In this case, the number of operations is at most $d^L|C| \times |U|$.

5 Generating an Upper Bound Set Function

We describe here how one can generate an upper bound set function, which can be used for pruning as described above in Section 3. Our method for generating an upper bound set function requires that \preceq respects \otimes . Hence we want to solve the following task:

Task A^* . Let \otimes be a commutative and associative operation on A and let \preceq be a pre-order on A which respects \otimes . Let C be a multiset of A -constraints over U . Let X be a variable in U . For each $x \in D(X)$ generate $SetUpper(x) \subseteq A$ such that for all complete assignments α with $\alpha(X) = x$, there exists $r \in SetUpper(x)$ with $\bigotimes_{c \in C} c(\alpha) \preceq r$.

In the remainder of this section we assume that \otimes is a commutative and associative operation on set A and that \preceq is a pre-order on A which respects \otimes .

Extending \otimes and \preceq to subsets

Let $A^* = 2^A$ be the set of all subsets of A . Define relation \preceq^* on A^* and operation \otimes^* on A^* as follows, where $R, S \subseteq A$:

- $R \preceq^* S$ holds if and only if for all $r \in R$ there exists $s \in S$ with $r \preceq s$.
- $R \otimes^* S = \{r \otimes s : r \in R, s \in S\}$.

Proposition 2 *Let \preceq be a pre-order on A which respects associative and commutative operation \otimes . Then, (i) \preceq^* is a pre-order on A^* ; (ii) \otimes^* is an associative and commutative operation on A^* ; (iii) \preceq^* respects \otimes^* .*

Solving Task A^* given method for solving Task A

Operation \otimes^* and relation \preceq^* restricted to singleton sets correspond to \otimes and \preceq , respectively. Because of this, A -constraints can be embedded as A^* -constraints. For any $c \in C$ define c^* as follows: $V_{c^*} = V_c$, so that c^* involves the same variables as c ; for $\alpha \in D(V_{c^*})$, define $c^*(\alpha) = \{c(\alpha)\}$. Let $C^* = \{c^* : c \in C\}$.

Suppose we have a general method for solving Task A (based, e.g., on the approach in Section 4). Because of Proposition 2, we can apply this method to multiset C^* of A^* -constraints, operation \otimes^* and relation \preceq^* , to achieve, for $x \in D(x)$, value $upper^*(x) \in A^*$ such that for all complete assignments α with $\alpha(X) = x$,

$$\bigotimes_{c^* \in C^*} c^*(\alpha) \preceq^* upper^*(x).$$

Now, $\bigotimes_{c^* \in C^*} c^*(\alpha)$ is equal to $\{\bigotimes_{c \in C} c(\alpha)\}$. By definition, $\{\bigotimes_{c \in C} c(\alpha)\} \preceq^* upper^*(x)$ if and only if there exists $r \in upper^*(x)$ with $\bigotimes_{c \in C} c(\alpha) \preceq r$, which shows that $upper^*$ is an upper bound set function for X , hence solving Task A^* .

This construction can be written another way. A solution to Task A generates function $upper$ on $D(X)$ given $A, \otimes, \preceq, C, X$. We might write such a solution as $upper_{C, X}^{A, \otimes, \preceq}$. To solve Task A^* we define $SetUpper$ to be $upper_{C^*, X}^{A^*, \otimes^*, \preceq^*}$.

Implementing basic upper bound functions on A^*

The construction described above leads to a method for solving Task A^* since we can apply the mini-buckets method from Section 4, for solving Task A via Task B and Task C . However, the method for solving Task C will involve (for this A^* system) use of basic upper bound functions, that we call ub^* and ub_{\otimes^*} , which, for any $R, S \subseteq A$, must satisfy: $R, S \preceq^* ub^*(R, S)$, so that $R \cup S \preceq^* ub^*(R, S)$; and $R \otimes^* S \preceq^* ub_{\otimes^*}(R, S)$, i.e., for all $r \in R$ and $s \in S$ there exists $t \in ub_{\otimes^*}(R, S)$ such that $r \otimes s \preceq t$.

We are assuming operation ub on A . The main issue in implementing ub^* and ub_{\otimes^*} is to prevent the upper bound sets R becoming too large. We could, in particular, choose some fixed $K \geq 1$ and ensure that if $|R|, |S| \leq K$ then neither $ub^*(R, S)$ nor $ub_{\otimes^*}(R, S)$ contain more than K elements, which will mean that every computed upper bound set R has cardinality at most K . We give one approach of this kind below.

Implementing ub^* . The idea is to incrementally add elements of S to R ; if an element in the current set is found to be worse than another element then we can delete it. Otherwise we replace a pair of elements by an upper bound of them. We use a function $AddElement(R, s)$ which produces an upper bound set for $R \cup \{s\}$ of cardinality at most K , where $R \subseteq A$ and $s \in A$. We set $AddElement(R, s) := R$ if there exists $r \in R$ with $s \preceq r$. Otherwise, if $T = \{r \in R : r \preceq s\}$ is non-empty or $|R| < K$, we set $AddElement(R, s) := (R \cup \{s\}) - T$. Otherwise if $T = \emptyset$ and $|R| = K$ then we choose two arbitrary elements r_1 and r_2 of $R \cup \{s\}$ and set $AddElement(R, s)$ to be $((R \cup \{s\}) - \{r_1, r_2\}) \cup \{ub(r_1, r_2)\}$. (For formalisms in which tests $r \preceq s$ are relatively expensive, we could instead define T using an incomplete check.)

We now define ub^* as follows: set $ub^*(R, \emptyset) := R$. For $S \neq \emptyset$ we choose any arbitrary element s of S and let $ub^*(R, S)$ be $AddElement(ub^*(R, S - \{s\}), s)$.

Implementing ub_{\otimes^*} . We set $ub_{\otimes^*}(R, \emptyset) := \emptyset$. For non-empty S we choose an arbitrary element s of S and set $ub_{\otimes^*}(R, S) := ub^*(ub_{\otimes^*}(R, S - \{s\}), R \otimes^* \{s\})$.

Complexity of solving Task A^*

The complexity of propagation using an upper bound set function is similar to that based on an upper bound function, except with an extra (multiplicative) factor depending on K . We need no more than $K^2 M|C| \times |U|$ applications of each of \otimes and ub , and no more than $2K^3 M|C| \times |U|$ ordering tests of the form $r \preceq s$. This perhaps suggests using a fairly small value of K (e.g., 2, 3, 4, ...)—even a two-element upper bound set ($K = 2$) can sometimes cause much stronger pruning than the single upper bound ($K = 1$). In some formalisms, ordering tests can be much cheaper than the operations, and so the K^2 term will then dominate when K is small.

6 Finding All Optimal Solutions (AOS) Problem

The approach we use for solving this problem is very similar to the Single Optimal Solutions problem. Instead of a single element *best* we maintain a subset *BestSet* of A which is the current set of best preference degrees of complete assignments found so far. Again this only gets updated at leaf nodes. *BestSet* is initialised to the empty set.

For pruning at a node we make use of a pre-order \preceq which respects \otimes and is a weakening of \preceq_u defined in Section 3. In particular, (given a neutral element) we could use relation $\preceq = \preceq'$ given by $r \preceq' s$ if and only if for all $t, u \in A$, $[s \otimes t < u \Rightarrow r \otimes t < u]$.

We can then use the method of Section 4 to generate an upper bound function *upper* for the node, and prune domain element x if there exists $s \in \text{BestSet}$ with $\text{upper}(x) < s$. Alternatively, we can use the method of Section 5 to generate an upper bound set function *SetUpper* and prune x if for all $r \in \text{SetUpper}(x)$, there exists $s \in \text{BestSet}$ with $r < s$, i.e., each element of *SetUpper*(x) is worse than some element of *BestSet*.

It is also easy to amend the approach to generate k optimal solutions (*kOS*).

7 Summary

In this paper we consider a very general framework for soft constraints, making very weak assumptions on the preference combination function and on the ordering. The main contributions of the paper are

- demonstrating that a mini-buckets style approach can be used in a branch-and-bound algorithm for optimisation for any instance of the soft constraints framework;
- showing how one can also use this kind of approach in a branch-and-bound algorithm which involves propagation of *upper bound sets*, enabling stronger pruning.

The extended mini-buckets approach allows the degree of propagation to be tailored to the problem: choosing either weak propagation at a node (with smaller mini-buckets bounds M (or L), and small upper bound sets, i.e., small K), or stronger but more expensive propagation.

REFERENCES

- [1] S. Bistarelli, T. Fruewirth, M. Marthe, and F. Rossi, ‘Soft constraint propagation and solving in constraint handling rules’, *Computational Intelligence, Special Issue on Preferences in AI and CP*, (2004).
- [2] S. Bistarelli, R. Gennari, and F. Rossi, ‘General properties and termination conditions for soft constraint propagation’, *CONSTRAINTS: An International Journal*, **8**(1), 79–97, (2003).
- [3] S. Bistarelli, U. Montanari, and F. Rossi, ‘Semiring-based Constraint Solving and Optimization’, *JACM*, **44**(2), 201–236, (1997).
- [4] R. Dechter and I. Rish, ‘Mini-buckets: A general scheme for bounded inference’, *J. ACM*, **50**(2), 107–153, (2003).
- [5] M. Ehrgott and X. Gandibleux, ‘Bounds and bound sets for biobjective combinatorial optimization problems’, *Lecture Notes in Economics and Mathematical Systems*, **507**, 241–253, (2001).
- [6] M. Gavanelli, ‘Partially ordered constraint optimization problems’, in *CP 2001*, (2001).
- [7] M. Gavanelli, ‘An implementation of Pareto optimality in CLP(FD)’, in *Proc. CP-AI-OR’02*, pp. 49–63, (2002).
- [8] U. Junker, ‘Preference-based search and multi-criteria optimization’, in *Proc. AAAI/IAAI 2002*, pp. 34–40, (2002).
- [9] U. Junker, ‘Preference-based search and multi-criteria optimization’, *Annals of Operations Research*, **130**, 75–115, (2004).
- [10] E. Rollón and J. Larrosa, ‘Bucket elimination for multiobjective optimization problems’, *Journal of Heuristics*, **12**(4-5), 307–328, (2006).
- [11] E. Rollón and J. Larrosa, ‘Constraint optimization techniques for exact multiobjective optimization’, in *Proceedings of the Seventh International Conference on Multi-Objective Programming and Goal Programming*, (2006).
- [12] E. Rollón and J. Larrosa, ‘Multi-objective propagation in constraint programming’, in *Proceedings of the European Conference on Artificial Intelligence (ECAI-2006)*, (2006).
- [13] M. Torrens and B. Faltings, ‘Using soft CSPs for approximating Pareto-optimal solution sets’, in *AAAI Workshop on Preferences in Constraint Satisfaction*, pp. 99–106, (2002).