A Theoretical Framework for Tradeoff Generation using Soft Constraints

Stefano Bistarelli
Istituto di Informatica e Telematica, CNR, Pisa, Italy
Dipartimento di Scienze
Universitá degli Studi "G. D'annunzio", Pescara, Italy

Barry O'Sullivan
Cork Constraint Computation Centre
Department of Computer Science
University College Cork, Ireland

Abstract

Tradeoffs have been proposed in the literature as an approach to resolving over-constrainedness in interactive constraint-based tools, such as product configurators, that reason about user preferences. It has been reported how tradeoffs can be modeled as additional constraints. This paper presents a formal framework for tradeoff generation based on the semiring approach to soft constraints. In particular, user preferences and tradeoffs are represented as soft constraints and as an entailment operator, respectively. The entailment operator is used to interactively generate new constraints representing tradeoffs. We also introduce a novel definition of substitutability for soft constraints upon which we present a relaxed definition of tradeoffs.

1 Introduction

A typical interactive configuration session is one where a human user articulates preferences for product features to a configurator which ensures consistency between the constraints of the problem and the user's desires. During such a session a point may be reached where all of the user's desires cannot be met. At this point the user could consider "tradeoffs" between his preferences. For example, in configuring a camera, the user may find that it is impossible to have one "*weighting less that 10 ounces with a zoom lens of 10X or more*", but could accept a tradeoff: "*I will increase my weight limit to 14 ounces if I can have a zoom lens of 20X or more.*" Ideally, we would like the configurator to suggest appropriate tradeoffs to the user.

In this paper we extend and formalize previous work on tradeoffs. In [7] tradeoffs are crisp binary constraints that are interactively generated to substitute strict unary crisp constraints representing user desires. In this paper we increase the utility of tradeoff generation since the amount of information gathered from the user is increased using soft constraints to represent preferences. The ability to capture the preferences of the user in a formal way was not addressed in the earlier work in this area. Furthermore, the extended framework presented here is general enough to deal with any arity of preference constraints, not only unary ones as reported in [7].

The task of tradeoff generation is also formalized in this paper. The possible tradeoffs are given as the result of an "entailment" function. A filter function is used to select one of the possible entailed tradeoff constraints. The final generalization in our framework is that tradeoff constraints are not necessarily limited to being binary. This provides us with a richer model of tradeoff. For example, we may wish to add a non-binary tradeoff constraint in certain situations, such as when we are prepared to have a pair of constraints, c_x and c_y, replaced by a ternary constraint, $c'_{(x,y,z)}$. The additional constraining influence on z could be regarded as an imposed constraint.

To handle both of these extensions we use the semiring-based framework [3, 4, 5] that has been shown to be able to represent both crisp and soft constraints in a uniform way. This general framework also gives us the possibility of expressing *approximate* tradeoffs. When it is not possible to find a tradeoff resulting in a solution better than some fixed level of consistency, we can consider the possibility of suggesting to the user solutions that are not worse than a given *degradation* factor. To do this we extend the notion of soft substitutability [**?**] from domain values to constraints.

Thus, the contributions of this paper are as follows:

1. a formal and general theoretical framework for tradeoff generation for interactive constraint processing that uses soft constraints to represent user preferences and an entailment operator to generate tradeoffs;

2. a notion of substitutability for soft constraints and a relaxed definition of tradeoff based on substitutability and solution degradation.

The remainder of the paper is organized as follows. Section 2 presents the necessary background on the semiring-based approach to handling soft constraints and on the tradeoff generation schema. Section 3 presents our general framework for tradeoff generation. An extension to our approach, for computing approximate tradeoffs, is presented in Section 4. Some concluding remarks are made in Section 5.

2 Background: Tradeoffs and Soft Constraints

Product configuration is becoming a well studied design activity which is often modeled and solved as a constraint satisfaction problem [1, 6, 10, 11]. In this paper we present a formal framework for tradeoff generation in interactive constraint processing. In the existing literature on this topic, a tradeoff is a binary constraint which substitutes a pair of unary preference constraints; the tradeoff constraint representing a satisfactory compromise for the user [7]. For example, consider a set, $U = \{c_1, \dots, c_k\}$, of user-specified unary preference constraints, and a set P of constraints defining the underlying problem, such that $U \cup P$ is inconsistent. A tradeoff constraint, T_{ij}, is a binary constraint involving a pair of variables, v_i and v_j, on which the user has specified a pair of unary preference constraints, $c_i \in U$ and $c_j \in U$, such that $U \cup T_{ij} - \{c_i, c_j\}$ is consistent and the user's preference on one variable has been strengthened and relaxed on the other. Therefore, currently, a tradeoff is a binary constraint which replaces a pair of unary preference constraints defined over the same pair of variables.

In this paper we regard each constraint in U as a *soft constraint* whose preference levels can be combined accordingly to the specific notion of combination for the problem. Soft constraints associate a qualitative or quantitative value either to the entire constraint or to each assignment of its variables. Such values are interpreted as a level of preference, importance or cost. The levels are usually ordered, reflecting the fact that some levels (constraints) are *better* than others. When using soft constraints it is necessary to specify, via suitable combination operators, how the level of preference of a global solution is obtained from the preferences in the constraints.

Several formalizations of the concept of *soft constraints* are currently available. In the following, we refer to the formalization based on c-semirings [3, 4, 5], which can be shown to generalize and express both crisp and soft constraints [2].

A semiring-based constraint assigns to each instantiation of its variables an associated value from a partially ordered set. When dealing with crisp constraints, the values are the booleans, *true* and *false*, representing the admissible and non-admissible values; when dealing with soft constraints the values are interpreted as preferences.

The framework must also handle the combination of constraints. To do this one must take into account such additional values, and thus the formalism must provide suitable operations for combination ($\times$) and comparison ($+$) of tuples of values and constraints. This is why this formalization is based on the concept of c-semiring. Below we present an overview of semiring-based constraint satisfaction.

Semirings.

A semiring is a tuple $\langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ such that:

- A is a set and $\mathbf{0}, \mathbf{1} \in A$;
- $+$ is commutative, associative and $\mathbf{0}$ is its unit element;
- $\times$ is associative, distributes over $+$, $\mathbf{1}$ is its unit element and $\mathbf{0}$ is its absorbing element.

A c-semiring is a semiring $\langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ such that: $+$ is idempotent, $\mathbf{1}$ is its absorbing element and $\times$ is commutative. Let us consider the relation $\leq_S$ over A such that $a \leq_S b$ iff $a + b = b$. Then it is possible to prove that (see [4]):

- $\leq_S$ is a partial order;
- $+$ and $\times$ are monotone on $\leq_S$;

- $\mathbf{0}$ is its minimum and $\mathbf{1}$ its maximum;
- $\langle A, \leq_S \rangle$ is a complete lattice and, for all $a, b \in A$, $a + b = lub(a, b)$.

Moreover, if $\times$ is idempotent, then: $+$ distribute over $\times$; $\langle A, \leq_S \rangle$ is a complete distributive lattice and $\times$ its glb. Informally, the relation $\leq_S$ gives us a way to compare semiring values and constraints. In fact, when we have $a \leq_S b$, we will say that b *is better than* a. In the following, when the semiring will be clear from the context, $a \leq_S b$ will be often indicated by $a \leq b$.

Constraint Problems.

Given a semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ and an ordered set of variables V over a finite domain D, a *constraint* is a function which, given an assignment $\eta : V \to D$ of the variables, returns a value of the semiring.

By using this notation we define $\mathcal{C} = \eta \to A$ as the set of all possible constraints that can be built starting from S, D and V.

Note that in this *functional* formulation each constraint is a function (as defined in [5]) and not a pair (as defined in [4]). Such a function involves all the variables in V, but it depends on the assignment of only a finite subset of them. We call this subset the *support* of the constraint.

Consider a constraint $c \in \mathcal{C}$. We define its support as $supp(c) = \{v \in V \mid \exists \eta, d_1, d_2 . c\eta[v := d_1] \neq c\eta[v := d_2]\}$, where

$$\eta[v := d]v' = \begin{cases} d & \text{if } v = v', \\ \eta v' & \text{otherwise.} \end{cases}$$

Note that $c\eta[v := d_1]$ means $c\eta'$ where η' is η modified with the association $v := d_1$ (that is the operator $[\ ]$ has precedence over application).

A *soft constraint satisfaction problem* is a pair $\langle C, con \rangle$ where $con \subseteq V$ and C is a set of constraints: con is the set of variables of interest for the constraint set C, which however may concern also variables not in con.

Note that a classical CSP is a SCSP where the chosen c-semiring is: $S_{CSP} = \langle \{false, true\}, \vee, \wedge, false, true \rangle$. Fuzzy CSPs can instead be modeled in the SCSP framework by choosing the c-semiring $S_{FCSP} = \langle [0, 1], max, min, 0, 1 \rangle$. Many other "soft" CSPs (Probabilistic, weighted, ...) can be modeled by using a suitable semiring structure, such as, ($S_{prob} = \langle [0, 1], max, \times, 0, 1 \rangle$, $S_{weight} = \langle \mathcal{R}, min, +, 0, +\infty \rangle$, ...).

Example 1 Figure 1 shows the graph representation of a fuzzy CSP.Variables X and Y, and constraints are represented, respectively, by nodes and by undirected (unary for c_1 and c_3 and binary for c_2) arcs, and semiring values are written to the right of the corresponding tuples. The variables of interest (that is the set con) are represented with a double circle. Here we assume that the domain D of the variables contains the values a, b and c.

If semiring values represent probability/fuzziness then, for instance, the tuple $\langle a, c \rangle \to 0.2$ in constraint c_2 can be interpreted as the probability/fuzziness of X and Y having values a and c, respectively, is 0.2. $\triangle$

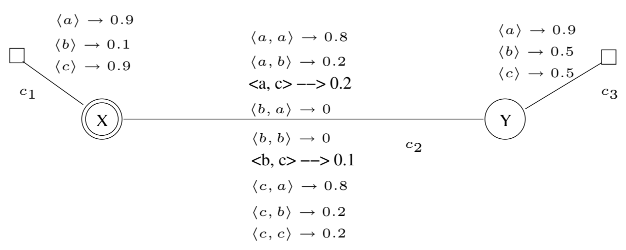
$\langle a \rangle \rightarrow 0.9$
$\langle b \rangle \rightarrow 0.1$
$\langle c \rangle \rightarrow 0.9$
c_1
X
$\langle a, a \rangle \rightarrow 0.8$
$\langle a, b \rangle \rightarrow 0.2$
<a, c> --> 0.2
$\langle b, a \rangle \rightarrow 0$
$\langle b, b \rangle \rightarrow 0$
<b, c> --> 0.1
$\langle c, a \rangle \rightarrow 0.8$
$\langle c, b \rangle \rightarrow 0.2$
$\langle c, c \rangle \rightarrow 0.2$
c_2
Y
$\langle a \rangle \rightarrow 0.9$
$\langle b \rangle \rightarrow 0.5$
$\langle c \rangle \rightarrow 0.5$
c_3

Figure 1: A fuzzy CSP.

Constraints can be defined in both extensional and intensional form. Constraints can be defined extensionally by listing each possible tuple with its preference level. An alternative way is to define constraints in an intensional way (for instance $x \neq y$, or $x \leq y$) and assign a level to each instantiation by using a function. As an example, consider the following fuzzy constraints:

$$c : \{x, y\} \rightarrow \mathcal{R}^2 \rightarrow [0,1] \qquad \text{s.t. } c(x,y) = \frac{1}{1 + |x - y|}$$

and

$$c' : \{x\} \rightarrow \mathcal{R} \rightarrow [0,1] \qquad \text{s.t. } c'(x) = \begin{cases} 1 & \text{if } x \leq 10, \\ 0 & \text{otherwise.} \end{cases}$$

Notice that the domain of both variables x and y is, in this example, the set of real numbers. As any fuzzy CSP, the definition of the constraints is instead in the interval $[0,1]$.

Combining constraints.

When there is a set of soft constraints $\mathcal{C}$, the combined weight of the constraints is computed using the operator $\otimes : \mathcal{C} \times \mathcal{C} \rightarrow \mathcal{C}$ defined as $(c_1 \otimes c_2)\eta = c_1\eta \times_S c_2\eta$.

Given a constraint $c \in \mathcal{C}$ and a variable $v \in V$, the *projection* of c over $V - \{v\}$, written $c \Downarrow_{(V-\{v\})}$, is the constraint c' s.t. $c'\eta = \sum_{d \in D} c\eta[v := d]$. Informally, projecting means eliminating some variables from the support. This is done by associating with each tuple over the remaining variables a semiring element which is the sum of the elements associated by the original constraint to all the extensions of this tuple over the eliminated variables. In short, combination is performed via the multiplicative operation of the semiring, and projection via the additive one.

Solutions.

The *solution* of a SCSP, $P = \langle C, con \rangle$, is the constraint $Sol(P) = (\bigotimes C) \Downarrow_{con}$. That is, we combine all constraints, and then project over the variables in *con*. In this way we get the constraint with support (not greater than) *con* which is "induced" by the entire SCSP. Note that when all the variables are of interest we do not need to perform any projection.

Solutions are constraints in themselves and can be ordered by extending the $\leq_S$ order. We say that a constraint c_1 is at least as constraining as constraint c_2 if $c_1 \sqsubseteq c_2$, where for any assignment η of variables then $c_1 \sqsubseteq c_2 \equiv c_1\eta \leq_S c_2\eta$. Notice that using the functional formulation [5] we can easily compare constraints eventhough they have different supports (in [3, 4] only constraints with the same support could be compared).

Example 2 Consider again the solution of the fuzzy CSP of Figure 1. It associates a semiring element with every domain value of variable X. Such an element is obtained by first combining all the constraints together and then projecting the obtained constraint over X.

For instance, for the tuple $\langle a, a\rangle$ (that is, $X = Y = a$), we have to compute the minimum between 0.9 (which is the value assigned to $X = a$ in constraint c_1), 0.8 (which is the value assigned to $\langle X = a, Y = a\rangle$ in c_2) and 0.9 (which is the value for $Y = a$ in c_3). Hence, the resulting value for this tuple is 0.8. We can do the same work for tuple $\langle a, b\rangle \rightarrow 0.2$, $\langle a, c\rangle \rightarrow 0.2$, $\langle b, a\rangle \rightarrow 0$, $\langle b, b\rangle \rightarrow 0$, $\langle b, c\rangle \rightarrow 0.1$, $\langle c, a\rangle \rightarrow 0.8$, $\langle c, b\rangle \rightarrow 0.2$ and $\langle c, c\rangle \rightarrow 0.2$. The tuples obtained are then projected over variable X, obtaining the solution $\langle a\rangle \rightarrow 0.8$, $\langle b\rangle \rightarrow 0.1$ and $\langle c\rangle \rightarrow 0.8$. $\triangle$

Sometimes it may be useful to just find the semiring value representing the least upper bound of the values yielded by the solutions. This is called the *best level of consistency* of an SCSP P and it is defined by $\mathit{blevel}(P) = \mathit{Sol}(P) \Downarrow_{\emptyset}$. We say that P is α-consistent when $\mathit{blevel}(P) = \alpha$.

3 Tradeoff as an Entailment Operator

In this section we define a general notion of tradeoff using the semiring-based framework presented above. We use hard constraints to represent the strict and unmodifiable conditions of the problem (denoted P). We use soft constraints to represent, modifiable, user preferences (denoted U).

While, in general, we may have some constraints in P that are soft, we would regard this softness as a cost that would not be handled in the same way as user-specified preference constraints. In our framework we model softness in the user's desires, U, as preferences. The user makes statements like *"I prefer to have a petrol engine over a diesel one"* in a quantifiable manner by associating semiring values with each option. However, any softness in the physical constraints in the problem, P, represent the costs, or penalties, associated with relaxing them. These costs can be thought of as problem statements such as *"a diesel engine is not normally available for the small chassis, but for an additional cost, we can make the option available"*. Note that these types of softness are semantically different and we would treat them as such. For the remainder of the paper we will simply regard each problem constraints in P as a hard (crisp) constraint.

We model tradeoffs as a special *entailment operator* [12]. As shown in [5], an entailment operator for soft constraints, given a set of constraints C, generates constraints c' s.t. $\bigotimes C \sqsubseteq c'$ (written as $C \vdash c'$).

Tradeoffs specialize entailments in two respects:

- firstly, constraints c' generated by the tradeoff operator are *substituted* for the preference constraints, C, while entailed constraints are usually *added* to the problem.
- secondly, when we add tradeoffs, we do not necessarily obtain a globally better solution, but we may obtain a *Pareto incomparable* one. Specifically, while low preference constraints, $C \in U$, are substituted by c', thus increasing the overall level of preference, c' usually also lowers the preference of some other constraints $\bar{C} \in U$.

So, for instance, if $U = \{c_1, \ldots, c_n\}$, a tradeoff for constraints $C = \{c_1, c_2\}$ could be a constraint c' s.t. $C \vdash c'$ and $supp(c') \supseteq supp(c_1) \cup supp(c_2)$ (usually we have $supp(c') \supset supp(c_1) \cup supp(c_2)$). Formally, we can define the notion of potential tradeoffs as follows:

Definition 1 (Potential Tradeoffs) *Given a configuration problem $\{P \cup U\}$ and a subset of user preference constraints $C \subseteq U$. We say that c' is a* Potential Tradeoff *for C ($c' \in Trades_{\langle P,U \rangle}(C)$) if*

- *$supp(\bigotimes C) \subseteq supp(c')$; let's call $\bar{C} \subseteq U$ the greatest set of preference constraints s.t. $supp(\bigotimes\{C, \bar{C}\}) = supp(c')$;*
- *$C \vdash c'$ (that is $C \sqsubseteq c'$);*

The meaning of this definition is that a potential tradeoff will increase the level of some user preference constraints (those in $C = \{c_1, c_2\}$), and possibly lower some other ones whose support is in $\bar{C}$.

Notice that after the application of the tradeoff operator, we never obtain a best level of consistency worse than before.

Theorem 1 *Given a configuration problem $\{P \cup U\}$ and a subset of user preference constraints $C \subseteq U$. If c' is a* Potential Tradeoff *for C ($c' \in Trades_{\langle P,U \rangle}(C)$) Then, $blevel(P \cup U - C \cup \{c'\}) \not< blevel(P \cup U)$;*

Proof Easily follows from the monotonicity of the $\otimes$ operator and from the hypotheses that $C \sqsubseteq c'$. □

3.1 Computing Tradeoffs

The way the preference constraints $C \subseteq U$ are selected, and the way a specific tradeoff c' is filtered from all the potential ones, and thus, $\bar{C}$ computed, is one of the most important issues when dealing with configuration problems. The potential tradeoffs can be restricted in various ways, which may be problem or context-specific. For example, we may wish to select a tradeoff in a way which could be regarded as "user-friendly". In the context of an interactive constraint-based configuration tool we may need to ensure that the user "trusts" the configurator, which may imply that previously accepted tradeoffs are not revisited. Some possible approaches to selecting preferences and filtering tradeoff constraints are presented in Section 3.2.

Notice that the filtered tradeoff could depend on the presence of a particular (partial) assignment, η, of the variables of the problem whose association has to be maintained[1].

Usually the configuration process first requests preference constraints from the user, and only if a solution better than a threshold α cannot be found with the acquired constraints, then a tradeoff is computed. To perform this check we always compute $blevel(P \cup U) = Sol(P \cup U) \Downarrow_{\emptyset}$ and compare this value with the semiring level α.

Therefore, tradeoff generation can be characterized in this context as a function:

$$Trades^{\alpha}_{\langle P,U,\eta,!\rangle} : \mathcal{C} \rightarrow \mathcal{C}$$

where:

- P is the set of *Problem constraints*;
- U is the set of *User-specified preference constraints*;
- ! is a filter (cut function) that first selects a set of preference constraints, $C \subseteq U$, and then selects one from among the potential tradeoffs giving c' (see Section 3.2).
- η is a (partial) assignment of the variables whose association has to be maintained;
- α represents the minimum best level of consistency we wish to achieve in the SCSP. The level α can be seen as the minimum level of global preference satisfaction we want to achieve;

Definition 2 (Tradeoffs) *Given a configuration problem $\{P \cup U\}$, and a subset of the user's preference constraints, $C \subseteq U$. We say that c' is a* Tradeoff *for C using the threshold α, the filter ! and the partial assignment η ($c' \in Trades^{\alpha}_{\langle P,U,\eta,!\rangle}(C)$), if the following are satisfied:*

- *c' is a* Potential Tradeoff*, that is:*
  - *$supp(\bigotimes C) \subseteq supp(c')$;*
  - *$C \vdash c'$;*
- *$blevel(\{P \cup U\}) < \alpha$ (i.e. the problem is no longer α-consistent);*

[1]In this paper η is fixed and never changes, so in all the definitions it could be omitted. Nevertheless it is important to include it in the notion of preference/tradeoff because in the next step of a configuration problem it will play an important role. After giving the user the opportunity to specify constraint preferences, he will be asked to make some more precise choices. In that phase of the configuration process, η becomes a partial assignment. We plan to address this second phase of the configuration problem as part of our research agenda in this area.

- *blevel*$(\{P \cup \{U - C\} \cup c'\}) \not< \alpha$ *(starting from a solution with an insufficient level of consistency, we want an assignment that gives a solution with a best level of consistency not worse than* α*).*

- ! *is used as a filter (see Section 3.2).*

3.2 Heuristics for selecting preference and tradeoff constraints

To completely define a tradeoff function we need to specify the selection and filtering heuristic, !, we will use to:

1. select the user preference constraints C to be eliminated – we will denote the selector $!^{out}$), and

2. select a tradeoff, c', from the set of potential tradeoffs computed by the entailment operator – we will denote the filter $!^{in}$; notice that this also implies the selection of the preference constraints $\bar{C}$ whose level of preference could be reduced.

Before presenting some specific examples of the selection heuristic, $!^{out}$, and the filtering heuristic, $!^{in}$, recall that the trigger for generating tradeoff constraints is the detection that there does not exist a solution which is α-consistent, i.e. that *blevel*$(\{P \cup U\}) < \alpha$. Below, we give here some possible instantiations of $!^{out}$ and $!^{in}$.

Random Selection: Random selection is always a possibility. In this case, the set C containing the preference constraint(s) to remove is randomly selected among those in $!^{out}(U) = \{c_j \in U : P \cup U - \{c_j\}$ is α-consistent$\}$;

Strictly related to P: The preference constraints we want to modify are strictly connected to the problem definition. In this case, the set C containing the preference constraint(s) to remove is randomly selected among those in $!^{out}(U) = \{c_j \in U : P \cup U - \{c_j\}$ is α-consistent $\wedge\ \exists c_i \in P : supp(c_i) \cap supp(c_j) \neq \emptyset\}$;

Has not appeared in a tradeoff: The preference constraints we want to modify have not already been affected by a tradeoff. In this case, the set C containing the preference constraint(s) to remove is randomly selected among those in $!^{out}(U) = \{c_j \in U : P \cup U - \{c_j\}$ is α-consistent$\wedge \nexists$ a tradeoff $t \in U : supp(t) \supseteq supp(c_j)\}$; this heuristic can be regarded as "user-friendly" since we do not ask the user to consider tradeoffs on variables which have already been involved in a tradeoff.

The tradeoff constraint, c', that we will choose, by filtering using $!^{in}$, has the properties that it will: (a) reflect a relaxation of the user's preference for constraint(s) C (selected by using $!^{out}$, and (b) a strengthening of the user's preference for constraint(s) $\bar{C}$.

Proposition 1 *Given a configuration problem $\{P \cup U\}$, a subset of user preference constraints $C \subseteq U$ and a tradeoff c' for C; let $\bar{C} \subseteq U$ be the greatest set s.t. $supp(\bigotimes\{C, \bar{C}\}) = supp(c')$; Then,*

- $(P \cup \{U - C\} \cup c') \Downarrow_{supp(C)} \sqsupseteq (P \cup U) \Downarrow_{supp(C)}$;
- $(P \cup \{U - C\} \cup c') \Downarrow_{supp(\bar{C})} \sqsubseteq (P \cup U) \Downarrow_{supp(\bar{C})}$.

Proof The first item easily follows from the monotonicity of the $\otimes$ operator and from the hypotheses that $C \sqsubseteq c'$. The second follows from the properties of projection and from the fact that $C \cap \bar{C} = \emptyset$. □

We now give some examples of $!^{in}$ filters that among all possible potential tradeoffs will select one. Some possible approaches we could adopt, based on [7] are:

Maximum viability: *c' is maximal w.r.t. P, U and η (that is for all $c'' \in Trades^{\alpha}_{\langle P,U,\eta,!\rangle}$ we have $\bigotimes\{P \cup \{U - C\} \cup c''\}\eta \not\geq \bigotimes\{P \cup \{U - C\} \cup c'\}\eta$;*

Minimum viability: *c' is minimal w.r.t. P, U and η (that is there does not exist $c'' \in Trades^{\alpha}_{\langle P,U,\eta,!\rangle}$ s.t. $\bigotimes\{P \cup \{U - C\} \cup c''\}\eta \leq \bigotimes\{P \cup \{U - C\} \cup c'\}\eta$.*

Notice that the first approach will be less tasking on the configurator since it always selects the less restrictive tradeoff, i.e. we will have $c' \Downarrow_{supp(C)} = \mathbf{1}$; the tradeoff will give to all domain values of the variables in C the best preference. However, in this way, several preferences made by the user on this assignment are lost.

On the other hand, the second approach will try to always stay as close as possible to the user's preferences, i.e. we will have $c' \Downarrow_{supp(C)} \sqsupset C$. The tradeoff will increase the preference on C just sufficiently to reach the prefixed level of consistency α. Therefore, such a minor increment could result in a significant number of tradeoff interactions during the configuration process. In fact, the constraint c' inserted by the configurator could be too strict to be α-consistent when the user will insert new preference constraints in future interactions.

It is worth pointing out at this point that a good user-interface could assist in the detection of preference constraints to restrict and which to relax. An appropriate user-interface could also take care of preference elicitation. For example, we could assume that importance relationships between variables is reflected by the order in which user-choices are made. This is also an issue we are investigating as part of our research agenda in this area.

4 Relaxing Tradeoffs

Sometimes a tradeoff that completely satisfies the user's desires does not exist. This results in a dead-end being reached, where the configurator cannot assist the user to achieve the desired level of consistency. One possibility to resolve this situation is to assist the user in backing-up through the tradeoffs that the user previously accepted and modify one or more of them. In this way, the inserted tradeoffs are considered as user preference constraints that can be removed and substituted during the configuration process. In any case, in our view, a good configurator must to be capable of following a strategy to find a solution for the user without changing tradeoff constraints which have already been accepted by the user. In this way the user may have more trust in the configurator.

Alternatively, a notion of less strict tradeoff is useful. In this section we define a relaxed notion of tradeoffs based on a notion of substitutability for soft constraints which extends a definition of substitutability among domain values [**?**]. In Section 4.1 we present a generalization of the notion of soft value substitutability to substitutability for soft constraints. We then use this definition in Section 4.2 to relax the earlier notion of tradeoff.

4.1 Substitutability for Soft Constraints

We propose a definition of substitutability among constraints, extending the notions of substitutability among values defined in [**?**]. Informally, according to the definition of soft value substitutability, a domain value b is substitutable for a domain value a for a variable v if all solutions involving $v := b$ have a level of preference greater than solutions involving $v := a$. We extend this definition to constraints below.

Definition 3 (constraint substitutability/interchangeability) *We say that constraint c_1 is Fully Substitutable for constraint c_2 ($c_1 \in FS(c_2)$) w.r.t. the set of constraints C when $\bigotimes\{C \cup c_2\} \sqsubseteq \bigotimes\{C \cup c_1\}$. When we have $\bigotimes\{C \cup c_2\} = \bigotimes\{C \cup c_1\}$ we say that c_1 and c_2 are Fully Interchangeable ($FI(c_1/c_2)$).*

Note that this definition of substitutability for constraints, when dealing with crisp constraints, is very similar to that defined in [9]. However, the definition above is more general since we are dealing with the soft constraint framework.

As reported in [**?**] for domain values, the above definition can be relaxed by using degradations. In particular, a constraint $c*$ can be used as a degradation factor.

Definition 4 (Substitutability with degradation) *Consider two constraints c_1 and c_2, the set of constraints C and a* degradation *constraint $c*$; we say that c_1 is* ^{c*}Substitutable *for c_2 w.r.t. C if and only if,*

$$\bigotimes\{C \cup c_2 \cup c*\} \sqsubseteq \bigotimes\{C \cup c_1\}$$

4.2 A Relaxed Definition of Tradeoffs

The above notion of degradation can be used to relax the tradeoffs that the configurator proposes to the user. In particular, we will define a notion of *approximate* tradeoff. However, first we will show that tradeoffs and substitutability constraints are strictly related.

Theorem 2 (Substitutability and Potential Tradeoffs) *Consider the sets of constraints P, U, and $C \subseteq U$, and a partial assignment of the variable η. Then, if*

1. *c' is substitutable for $\bigotimes C$ ($c' \in FS(\bigotimes C)$) w.r.t. the set of constraints $\{P \cup U\}$ and the (partial) assignment η, and*
2. *$supp(c') \supseteq supp(C)$,*

then $c' \in Trades_{\langle P,U \rangle}(C)$.

Proof By definition of substitutability for constraints we have $C \vdash c'$. With the second hypothesis of the theorem we easily have all that is required to say that c' is a potential tradeoff for C ($c' \in Trades_{\langle P,U \rangle}(C)$). $\square$

We can now relax the definition of tradeoff by using the relaxed version of substitutability.

Definition 5 (Degradation Tradeoffs) *Consider the constraint c', the (set of) constraint $C \in U$, a set of constraints $\{P \cup U\}$, and a* degradation *constraint $c*$; we say that c' is a Potential tradeoff for C w.r.t. $\{P \cup U\}$ and η with degradation $c*$ ($c' \in {}^{c*}Trades_{\langle P,U \rangle}(C)$) if and only if, c' is* ^{c*}Substitutable *for $\bigotimes C$ w.r.t. $\{P \cup U\}$.*

Degradation tradeoffs can be regarded as *approximate* tradeoffs. Notice that by using the notion of degradation, we do not impose the requirement that the configurator has to find a better configuration, but rather a configuration *similar* to that chosen by the user. In this way the solution that the configurator will find will be not too far from the solution proposed by the user in terms of its degree of preference. In this respect the solution is an approximation.

5 Conclusions and Future Work

Tradeoffs have been proposed in the literature as an approach to resolving over-constrainedness in interactive constraint-based tools, such as product configurators, that reason about user preferences. It has already been reported in the literature how tradeoffs can be modeled as additional constraints. This paper presents a formal framework for tradeoff generation based on the semiring approach to handling soft constraints. In particular, we present a formal and general definition of tradeoff generation for interactive constraint processing. We present a novel definition of substitutability for soft constraints upon which we present a relaxed definition of tradeoffs.

Our research agenda in this area involves studying intelligent interfaces for reasoning about the relative importance of the user's preferences. For example, we could assume that importance relationships between variables is reflected by the order in which user-choices are made. We are also working on an empirical evaluation of a number of heuristics for selecting preference constraints to be considered as the basis for generating tradeoffs and strategies for filtering from the set of tradeoffs generated by our entailment operator.

In summary, we have presented a formal framework for studying a very important aspect of interactive constraint processing, the ability to assist users achieve their desires to the maximal degree possible. This framework provides the basis for a research agenda in the area of interactive constraint satisfaction with practical applications in domains such as product configuration, e-commerce, interactive scheduling, negotiation and explanation. As future work we also plan to integrate the notion of tradeoffs into the CHR framework [8].

Acknowledgment

This work has received support from Enterprise Ireland under their Basic Research Grant Scheme (Grant Number SC/02/289) and their International Collaboration Programme (Grant Number IC/2003/88).

References

[1] J. Amilhastre, H. Fargier, and P. Marguis. Consistency restoration and explanations in dynamic csps – application to configuration. *Artificial Intelligence*, 135:199–234, 2002.

[2] S. Bistarelli, H. Fargier, U. Montanari, F. Rossi, T. Schiex, and G. Verfaillie. Semiring-based CSPs and Valued CSPs: Frameworks, properties, and comparison. *CONSTRAINTS: An international journal. Kluwer*, 4(3), 1999.

[3] S. Bistarelli, U. Montanari, and F. Rossi. Constraint Solving over Semirings. In *Proc. IJCAI95*, San Francisco, CA, USA, 1995. Morgan Kaufman.

[4] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based Constraint Solving and Optimization. *Journal of the ACM*, 44(2):201–236, Mar 1997.

[5] S. Bistarelli, U. Montanari, and F. Rossi. Soft concurrent constraint programming. In *Proc. 11th European Symposium on Programming (ESOP)*, Lecture Notes in Computer Science (LNCS), pages 53–67. Springer, 2002.

[6] A. Felfernig, G. Friedrich, D. Jannach, and M. Stumpter. Consistency-based diagnosis of configuration knowledge-bases. In *Proceedings of the 14h European Conference on Artificial Intelligence (ECAI'2000)*, pages 146–150, 2000.

[7] E. C. Freuder and B. O'Sullivan. Generating tradeoffs for interative constraint-based configuration. In *Proceedings of CP-2001*, pages 590–594, Nov 2001.

[8] T. Frühwirth. Constraint handling rules. In *Constraint Programming: Basics and Trends*, volume 910 of *Lecture Notes in Computer Science (LNCS)*, pages 90–107. Springer, 1995.

[9] P. Jeavons, D. Cohen, and M. Copper. A substitution operation for constraints. In *Proceedings of the Second International Workshop on Principles and Practice of Constraint Programming – CP-94*, volume 874 of *LNCS*, 1994.

[10] M. Moretti, F. Rossi, E.C. Freuder, C. Likitvivatanavong, and R. Wallace. Explanations and optimisation in preference-based configurators. In *Proceedings of the ERCIM/CologNet Workshop on Constraint Solving and Constraint Logic Programming*, pages 58–71. To appear as a volume for LNAI, Springer-Verlag, 2002.

[11] D. Sabin and R. Weigel. Product configuration frameworks – a survey. *IEEE Intelligent Systems and their applications*, 13(4):42–49, July–August 1998. Special Issue on Configuration.

[12] V.A. Saraswat. *Concurrent Constraint Programming.* MIT Press, 1993.