Tradeoff Generation using Soft Constraints

Stefano Bistarelli[1,2], Jerome Kelleher[3], and Barry O'Sullivan[3]

[1] Istituto di Informatica e Telematica, CNR, Pisa, Italy
Stefano.Bistarelli@iit.cnr.it
[2] Dipartimento di Scienze
Universitá degli Studi "G. D'annunzio" di Chieti-Pescara, Italy
bista@sci.unich.it

[3] Cork Constraint Computation Centre
Department of Computer Science, University College Cork, Ireland
{j.kelleher|b.osullivan}@4c.ucc.ie

Abstract. Tradeoffs have been proposed in the literature as an approach to resolving over-constrainedness in interactive constraint-based tools, such as product configurators. 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, respectively, represented as soft constraints and as an entailment operator. The entailment operator is used to interactively generate new constraints representing tradeoffs. The framework we present is well-motivated by real-world approaches that exploit tradeoff generation in online buying and configuration processes.

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 car, the user may find that it is impossible to have one "*with an engine size more that 2 litres and having a minimum fuel consumption of 15 km per litre*", but could accept a tradeoff: "*I will reduce my engine size requirement to 1.6 litres if I can have a minimum fuel consumption of 20 km per litre.*" Ideally, we would like the configurator to suggest appropriate tradeoffs to the user. There are a number of web-sites that attempt to help users to make tradeoffs between conflicting preferences and desires given the space of possible products that are available. The most well-known of these is the very successful *Active Buyer's Guide* web-site[1], which takes a similar two-stage approach to the one

[1] See http://www.activebuyersguide.com.

proposed here. Initially, user preferences are acquired with tradeoffs being proposed/elicited where necessary. Finally, users make their choices for the features of the desired product.

In this paper we extend and formalize previous work on tradeoffs. In [13] 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 by using soft constraints to represent preferences. Representing 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 [13].

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 [4, 5, 7] that has been shown to be capable of representing both crisp and soft constraints in a uniform way.

Thus, the primary contribution of this paper is 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.

The remainder of the paper, which extends [8], 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 example is presented in Section 4. In Section 5 an evaluation of the framework is presented with a discussion of how this can be deployed in a real world context. Some concluding remarks are made in Section 6.

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, 12, 14, 19]. 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 [13]. For example, consider a set, $U = \{c_1, \ldots, c_k\}$, 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 according 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 [4, 5, 7], which can be shown to generalize and express both crisp and soft constraints [3].

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.

More precisely, they are based on a semiring structure $S = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ and a set of variables V with domain D. In particular the semiring operation $\times$ is used to combine constraints together, and the $+$ operator for projection.

Technically, a *constraint* is a function which, given an assignment $\eta : V \to D$ of the variables, returns a value of the semiring that is $\mathcal{C} : \eta \to A$ is the set of all possible constraints that can be built starting from S, D and V (values in A are interpreted as level of preference or importance or cost).

Note that in this *functional* formulation, each constraint is a function (as defined in [7]) and not a pair (as defined in [5]). 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 his 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).

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

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 [7] we can also compare constraints that have different support (in [4, 5] only constraints with the same support could be compared).

Sometimes it may be useful to find only a semiring value representing the least upper bound among the values yielded by the solutions. This is called the *best level of consistency* of an SCSP P and it is defined by $blevel(P) = Sol(P) \Downarrow_{\emptyset}$.

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 constraint in P as a hard (crisp) constraint.

We model tradeoffs as a special *entailment operator* [20]. As shown in [7], 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, $blevel(P \cup U - C \cup \{c'\}) \not< blevel(P \cup U)$.*

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[2].

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);*
- *$blevel(\{P \cup \{U - C\} \cup c'\}) \not< \alpha$ (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).*

[2] 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.

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 to eliminate, C – we will denote this selector $!^{out}$, and
2. select a tradeoff, c', from the set of potential tradeoffs computed by the entailment operator – we will denote this 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 $\mathit{blevel}(\{P \cup U\}) < \alpha$. In a configuration context α can represent a threshold in utility, cost or some other application-specific metric for measuring the quality or suitability of the outcome.

Below, we give here some possible instantiations of $!^{out}$ and $!^{in}$. Firstly we consider heuristics for implementing the $!^{out}$ selector. Of course, in addition to the heuristics we present below, a variety of others are possible which can be made application- or user-specific.

Random Selection: Random selection is always a possibility. In this case, the set C containing the preference constraint(s) to remove is randomly selected from amongst those in $!^{out}(U) = \{c_j \in U : P \cup U - \{c_j\} \text{ is } \alpha\text{-consistent}\}$; This heuristic is an obvious naive approach.

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

Has not appeared in a tradeoff: The preference constraint we want to modify has not already been affected by a tradeoff. In this case, the set C containing the preference constraint(s) to remove is selected from amongst those in $!^{out}(U) = \{c_j \in U : P \cup U - \{c_j\} \text{ is } \alpha\text{-consistent} \wedge \nexists \text{ a tradeoff } t \in U : \mathit{supp}(t) \supseteq \mathit{supp}(c_j)\}$; 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 to proposed to the user, by filtering from the set of potential tradeoffs using $!^{in}$, has the properties that it will:

1. reflect a relaxation of the user's preference for constraint(s) C (selected by using $!^{out}$), and

2. 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 also $\bar{C} \subseteq U$ the greatest set s.t. $supp(\bigotimes\{C, \bar{C}\}) = supp(c')$; Then,*

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

We now give some examples of $!^{in}$ filters that select one tradeoff from among all possible potential tradeoffs. Some possible approaches we could adopt, based on [13] 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 not exists $c'' \in Trades^{\alpha}_{\langle P,U,\eta,!\rangle}$ 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 the preferences made by the user on this assignment are lost. Essentially, using the first approach we maximize the possible solutions that the user will be able to choose from, but we cannot guarantee that the user's original set of preferences will be satisfied to the maximal extent.

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 required 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. However, in this case we run the risk of being misled by the user, in the sense that we may have to generate many tradeoffs in order to recover from a set of strong and over-constraining preferences.

It is worth pointing out at this point that a good user-interface could assist in the detection of preference constraints to strengthen 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.

Furthermore, the approach we are advocating in this work can be seen as an automated version of that used on the very successful *Active Buyer's Guide* web-site.

4 An Example

The example configuration problem that will be studied here is based on the soft n-queens problem [6]. However, we will assume, for space reasons, that all of the problem constraints in P are hard in the traditional sense. Furthermore, for this example we have chosen the fuzzy semiring, $S = \langle [0,1], max, min, 0, 1 \rangle$, so constraints in U have preference between 0 and 1 and they are combined using min.

The n-queens problem exhibits many of the features one finds in a real-world configuration problem. Specifically, consider the formulation of the problem that we use here, where each column is represented as a variable and the row position of a queen in a particular column is denoted by the value assigned to the appropriate variable. Between the variables are constraints encoding the requirement that no pair of queens attack each other. In terms of a configuration problem, the variables represent the features over which the user/customer make choices. The domain of each variable represents the possible choices that the user can make for a particular feature. The constraints between the variables represent the compatibilities between combinations of instantiations of the features of the product.

The user attempts to solve this configuration problem by interactively specifying some preference constraints to a constraint-based configurator. During the interactive session with the configurator, the user may specify a preference constraint which causes the problem to become "over-constrained", identified by the problem becoming less than α-consistent (for some fixed α). At this point our configurator attempts to recommend tradeoff constraints to the user which he/she can accept before continuing.

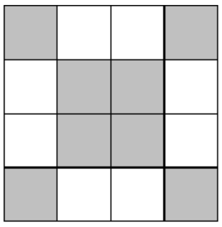
Thus, the interactive solving process, based on [13], can be summarized as follows:

```
Repeat until preferences are specified for all variables in the problem:
  – Repeat until over-constrained – $blevel(P \cup U) < \alpha$:
      • the user specifies a preference constraint, $c \in U$;
  – Repeat until user is satisfied:
      • the system proposes a tradeoff
```

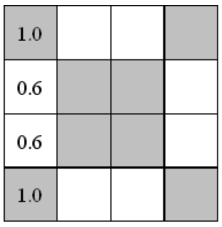
For the purposes of this example let's assume we wish to solve the 4-Queens problem with a crisp set of problem constraints, P, i.e. the configuration problem constraints are hard. This means that the only preference/costs we have to consider are given by the user (inside the set U).

In the following example we will consider only unary preference constraints representing the columns, $\{c_1, c_2, c_3, c_4\}$ of the chess-board. The user proposes unary preference constraints on each of the columns, sequentially from the first to the fourth, representing wishes for the placement of queens.

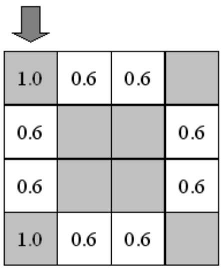
Furthermore, we also assume that only binary tradeoff constraints c' will be generated. Finally, lets assume that the user wishes to achieve a minimum level of 0.5-consistency, i.e. $\alpha = 0.5$.



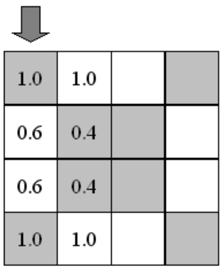
(a) Chessboard with crisp constraints.



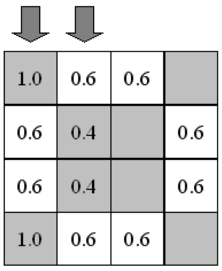
(b) Preferences for the 1st column.



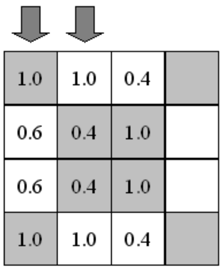
(c) Evaluation of the 1st preference.



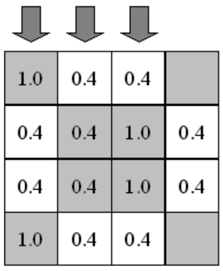
(d) Added preferences for the 2nd column.



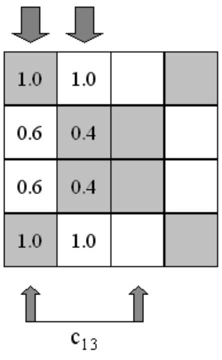
(e) Evaluation of the preference for the 1st and 2nd columns.



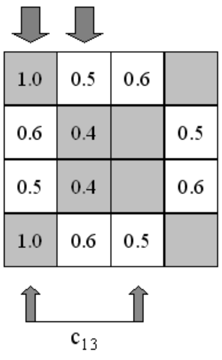
(f) Added preferences for the 3rd column.



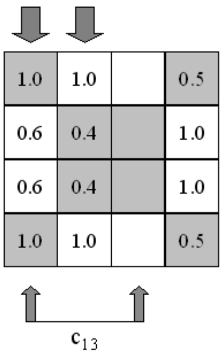
(g) Evaluation of the preferences for 1st,2nd and 3rd columns.



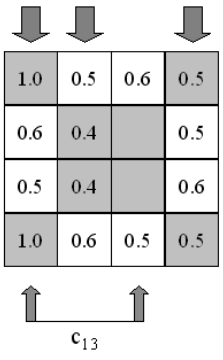
(h) Added a tradeoff between preferences on the 1st and 3rd column.



(i) Evaluation of the preferences for 1st and 2nd columns and of the tradeoff.



(j) Added preferences for the 4th column.



(k) Evaluation after the preferences over the 4th column.

Fig. 1. An example interaction with all hard constraints in P.

Problem constraints in P: Figure 1(a) the *no-attack* crisp constraints in$P$ are represented. The grey squares represent impossible configurations (i.e. with preference level 0); the white squares represent consistent positions for the queens.

User Decision #1: The user states a unary preference constraint on the values for column 1: $c_1(1) = 1.0, c_1(2) = 0.6, c_1(3) = 0.6, c_1(4) = 1.0$ (see Figure 1(b)). The constraint network representing this problem is still α-consistent, because it is possible to achieve a solution having a semiring value of 0.6 (Figure 1(c)) by putting the queen in row 2 or 3, so the user is free to continue articulating preference constraints. Notice that queens cannot be positioned in row 1 or row 4 because the crisp constraints in P do not admit any solution with queens in these positions.

User Decision #2: The user states a unary preference constraint on the values for column 2: $c_2(1) = 1.0, c_2(2) = 0.4, c_2(3) = 0.4, c_2(4) = 1.0$ (Figure 1(d)). The constraint network representing this problem is still α-consistent: setting $c_1 = 2$ and $c_2 = 4$ could yield a solution with preference 0.6 (Figure 1(e)). In fact, possible solutions are obtained by putting the queen in row 1 or 4[3] Therefore, the user is free to continue articulating preferences.

User Decision #3: The user states a unary preference constraint on the values for column 3: $c_3(1) = 0.4, c_3(2) = 1.0, c_3(3) = 1.0, c_3(4) = 0.4$ (Figure 1(f)). The constraint network representing this problem is no longer α-consistent: the best solution has a semiring value of 0.4 (for example setting $c_1 = 2$, $c_2 = 4$ and $c_3 = 1$) (Figure 1(g)). Therefore, the user must consider tradeoffs.

Tradeoff #1: The configurator needs to select a preference constraint to remove and a tradeoff to add. In this case all the heuristics presented in Section 3.2 for $!^{out}$ will select the preference constraint on column 3. In fact $C = \{c_3\}$ is the only set of preference constraint s.t. $P \cup U - C$ is 0.5-consistent.
Furthermore, lets assume that the tradeoff constraint selected using $!^{in}$ involves column 1 and 3. Let's suppose our heuristic select: $c_{13} = \{(1,1) = 0.6, (2,1) = 0.6, (3,1) = 0.6, (4,1) = 0.6, (1,2) = 1.0, (2,2) = 1.0, (3,2) = 1.0, (4,2) = 1.0, (1,3) = 1.0, (2,3) = 1.0, (3,3) = 1.0, (4,3) = 1.0, (1,4) = 0.5, (2,4) = 0.5, (3,4) = 0.5, (4,4) = 0.5\}$ (Figure 1(h)).
Assuming the user accepts this tradeoff constraint, the network is once again α-consistent (Figure 1(i)). The user could set $c_1 = 2$, $c_2 = 4$ and $c_3 = 1$ for a semi-ring value of 0.5. Note that by adding the tradeoff we also remove the unary preference constraint on column 3 and relax the preferences on column 1.

[3] In this example all of the constraints in P are crisp, so the possible positions of the queens are strictly imposed. Instead, if P contained soft constraints, more possibilities (with different costs) would be possible.

User Decision #4: The user states a unary preference constraint on the values for column 4: $c_4(1) = 0.5, c_4(2) = 1.0, c_4(3) = 1.0, c_4(4) = 0.5$ (Figure 1(j)). The constraint network representing this problem is still α-consistent (Figure 1(k)). The user could set $c_1 = 2$, $c_2 = 4$, $c_3 = 1$ and $c_4 = 3$ for a semi-ring value of 0.6. Preferences have now been defined for each column.

5 Evaluation and Discussion

To inform intuition on our approach to tradeoff generation based on the framework presented in this paper, we implemented a concretisation which satisfies the requirements of the framework in a minimal way. Specifically, we generated random entailed constraints c' such that $\forall \eta . c'\eta \geq \alpha$. In this way we guarantee that the problem will always be alpha-consistent after we perform the tradeoff operation. Obviously this approach is not intended to reflect any "real-world" tradeoffs, however, it does provide us with some basic insights as to the number of tradeoffs will be required with different levels of α in a real-world instantiation of this framework.

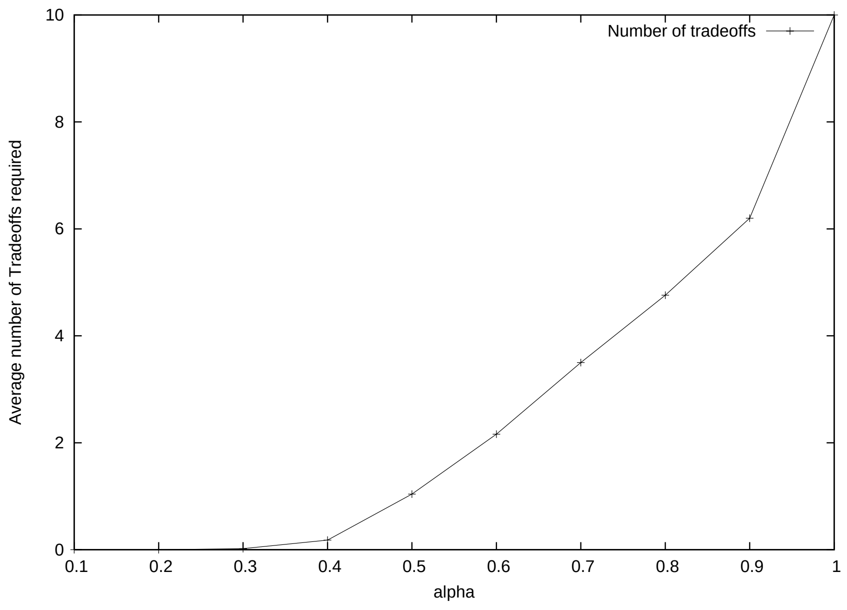


Fig. 2. Average number of tradeoffs required over a fifty user-interaction simulations at various levels of α.

In Figure 2 we show results obtained by averaging the number of tradeoffs required over fifty user-interaction simulations. In this experiment, we use the

10-queens problem. The constraints that define this problem represent the crisp set of constraints P. We randomly generate unary fuzzy constraint functions to represent each of the user preference constraints in U. To simulate a general user-interaction we take a sequential approach to eliciting user preferences and determine if the user has stated a preference which over-constrains the overall problem (i.e. $P \cup U$). If the user has over-constrained, identified by the problem becoming α-inconsistent, then it is necessary to generate a tradeoff constraint.

Thus, in this experiment we sequentially generate 10 unary preference functions and evaluate $blevel(P \cup U)$ after each constraint is added. If this evaluates to greater than α, we continue on to generate the next preference function, if not we generate a tradeoff to make the problem α-consistent again.

If we consider Figure 2 we see that at low levels of α we do not need to generate many tradeoffs, but at higher levels of α, as the user gets more demanding, we find that the number of tradeoffs required to find a solution increase dramatically. This is to be expected since at lower levels of α the likelihood of over-constraining the problem are very low, while as α increases we are more likely to over-constrain the problem after each user preference constraint is added to the model.

An interesting question here is how many α-consistent solutions of the problem do we find as α changes. Interestingly, we observed that at very low and very high levels of α we "lost" very few α-consistent solutions. By the term "lost" solution, we mean how many of the solutions to the problem are no longer α-consistent, at the end of the simulated interaction, due to tradeoffs. Specifically, we observed that as α increased, the number of solutions we "lost" gradually increased from 0 to some maximum, less than the number of solutions to the problem, and after some level of α began to decrease again. From Figure 2 we can see that low (high) α implies fewer (more) tradeoffs. This suggests that having to rely either lightly or heavily on tradeoffs is likely to be a successful strategy to find satisfactory solutions to a problem. Of course the extent to which this is possible is determined by the level of α sought by the user. Therefore, we are more likely to be able to satisfy the expectations of easy-to-please users, who require low α, or very selective users, who require high α. Note that this is a similar phenomenon to that observed by Freuder and O'Sullivan in the earlier work on tradeoff generation for crisp CSPs [13]. They noted how both greedy and easy-to-please users were easy to satisfy, but non-committal users were difficult.

While the instantiation of the tradeoff framework for soft constraints studied in this evaluation is based on very naive random methods to generating user preferences and tradeoff constraints, we can regard the results presented in Figure 2 as a worst-case scenario for this problem. While we always succeed in finding a solution, any form of intelligence that can be built into the selection heuristics, $!^{out}$ and $!^{in}$, will improve upon these results. This raises the interesting issue of how our work can be deployed in a real-world context. We address that issue in some detail below.

Many approaches to dealing with compromises in user preferences or constraints exist [2, 9, 10, 16–18]. In our work we are concerned with attempting to automate tradeoff generation in order to take the onus from the user and place it with the system. The objective is that users will be assisted find satisfactory tradeoffs to their preference constraints in a search-style interaction while the system monitors consistency. Ideally, the tradeoff generation mechanism in such an interactive system will be capable of presenting tradeoffs which are likely to be accepted by the user, so that he or she can continue to search for his or her solution. This style of interaction is search focused, rather than the solution focused approaches of Recommender Systems or example critiquing systems [11].

In the framework we present here, the critical components are the filter used to select constraints to relax to regain consistency ($!^{out}$) and the filter used to decide what preferences to modify to form a tradeoff ($!^{in}$). One approach we could adopt is to attempt to learn appropriate filters based on past sessions with the user, or application-specific characteristics of the problem domain.

We also do not preclude user involvement in the tradeoff process. Indeed, one could imagine that the user participates in the selection of preferences to relax and strengthen. The framework we have presented still applies.

Finally, the role of user-interfaces for preference elicitation is an important one. For example, a user-interface can help translate user-choices into appropriate semiring values for each preference constraint. Also, a user-interface could provide us with a mechanism for reasoning about the relative importance of user preferences, upon which we could develop appropriate filters.

Therefore, we would argue that we have presented a generic framework for interactive search-based tradeoff systems. The framework is general purpose and can be instantiated in a number of ways. One direction for future work is the consideration of particular instantiations of the framework targeted at particular domains, or types of interaction.

6 Conclusions

Tradeoffs have been proposed in the literature as an approach to resolving over-constrainedness in interactive constraint-based tools, such as product configurators. 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. The framework we present is well-motivated by real-world approaches that exploit tradeoff generation in online buying and configuration processes.

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 are reflected by the order in which user-choices are made. We are also interested in a detailed empirical evaluation of a number of strategies for learning appropriate filter functions for more natural user-friendly interaction.

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 tradeoff into the CHR framework [15].

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. D. Bahler, C. Dupont, and J. Bowen. Mixed quantitative/qualitative method for evaluating compromise solutions to conflict in collaborative design. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 9:325–336, 1995.
3. 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.
4. S. Bistarelli, U. Montanari, and F. Rossi. Constraint Solving over Semirings. In *Proc. IJCAI95*, San Francisco, CA, USA, 1995. Morgan Kaufman.
5. S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based Constraint Solving and Optimization. *Journal of the ACM*, 44(2):201–236, Mar 1997.
6. S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based Constraint Logic Programming: Syntax and Semantics. *ACM Transactions on Programming Languages and System (TOPLAS)*, 23:1–29, 2001.
7. 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.
8. S. Bistarelli and B. O'Sullivan. A theoretical framework for tradeoff generation using soft constraints. In *Proceedings of AI-2003*, pages 69–82. Springer, 2003.
9. Ronen I. Brafman and Carmel Domshlak. Tcp-nets for preference-based product configuration. In *Proceedings of the Forth Workshop on Configuration (in ECAI-02)*, pages 101–106, July 2002.
10. Yannick Descotte and Jean-Claude Latombe. Making compromises among antagonist constraints in a planner. *Artificial Intelligence*, 27:183–217, 1985.
11. B. Faltings, P. Pu, M. Torrens, and P. Viappiani. Designing example-critiquing interaction. In *International Conference on Intelligent User Interfaces*, 2004.
12. 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.

13. E. C. Freuder and B. O'Sullivan. Generating tradeoffs for interactive constraint-based configuration. In *Proceedings of CP-2001*, pages 590–594, Nov 2001.
14. E.C. Freuder, C. Likitvivatanavong, M. Moretti, F. Rossi, and R.J. Wallace. Computing explanations and implications in preference-based configurators. In Barry O'Sullivan, editor, *Recent Advances in Constraints*, volume 2627 of *LNAI*, pages 76–92, 2003.
15. 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.
16. R.L. Keeney and H. Raifa. *Decisions with Multiple Objectives: Preferences & Value Tradeoffs.* Cambridge University Press, 2002.
17. P. Pu, B. Faltings, and P. Kumar. User-involved tradeoff nalysis in configuration tasks. In *Proceedings of the Third CP Workshop on User-Interaction in Constraint Satisfaction*, pages 85–102, Septembet 2003.
18. S. Y. Reddy, K. W. Fertig, and D. E. Smith. Constraint management methodology for conceptual design tradeoff studies. In *Proceedings of the 1996 ASME Design Engineering Technical Conferences and Computers in Engineering Conference*, August 1996. Irvine, California.
19. 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.
20. V.A. Saraswat. *Concurrent Constraint Programming.* MIT Press, 1993.