Constraints, 9, 289–310, 2004
© 2004 Kluwer Academic Publishers. Manufactured in The Netherlands.

Decision Tradeoff Using Example-Critiquing and Constraint Programming

PEARL PU pearl.pu@epfl.ch
Human Computer Interaction Group, Swiss Federal Institute of Technology (EPFL), CH-1015 Lausanne, Switzerland

BOI FALTINGS boi.faltings@epfl.ch
Artificial Intelligence Laboratory, Swiss Federal Institute of Technology (EPFL), CH-1015 Lausanne, Switzerland

Abstract. We consider *constructive* preference elicitation for decision aid systems in applications such as configuration or electronic catalogs. We are particularly interested in supporting decision tradeoff, where preferences are revised in response to the available outcomes. In several user-involved decision aid systems we have designed in the past, we were able to observe three generic tradeoff strategies that people like to use. We show how a preference model based on soft constraints is well-suited for supporting these strategies. Such a framework provides an *agile preference model* particularly powerful for preference revision during tradeoff analysis. We further show how to integrate the constraint-based preference model with an interaction model called example-critiquing. We report on user studies which show that this model offers significant advantages over the commonly used ranked-list model, especially when the decision problem becomes complex.

Keywords: constructive preference elicitation, preference revision, agile preference model, preference-based search, user-involved decision tradeoff, example critiquing interfaces, soft constraints, configuration, electronic catalogs, user studies

1. Introduction

To perform complex tasks, such as searching in an online catalog for configurable products, planning a trip, or scheduling resources, people increasingly rely on computerized systems to find outcomes that best satisfy their needs and preferences. However, needs and preferences get into conflicts and choosing the best solution is often a tradeoff problem. Many automated decision support systems do not satisfactorily help users make tradeoffs. Those that do require a fully specified value function. On the other hand, a complete value function model is difficult to establish for the following reasons:

- Users' preference models are not likely to be complete [28]. Therefore, it is hard to state value functions for preferences that do not exist.
- Users' beliefs about desirability can be ephemeral and uncertain. For example, as we studied our users in choosing vacation packages, we observed that they often started with a high value function on low-cost deals. However, as they discovered other features that were more important to them than price, they would change the initial value function.

The situation is even worse for users of consumer e-commerce sites because they are under severe time constraints:

- They are unlikely to make the *effort* to express preferences for decision criteria until a pertinent context appears. For example, a user normally will not express a preference for intermediate airports between Geneva and Boston until outcomes show that all flights make a stop somewhere. Only then would he evaluate whether Munich, Frankfurt, or London is a more desirable transit airport.
- Their expressions of preferences are not likely to follow the order *imposed* by a system because users' decision objectives evolve as they learn more about the options. Pu and Faltings [31] explained that such rigid elicitation procedures will lead them to state preferences for means objectives, rather than fundamental ones.
- Their domain knowledge is often *insufficient* for stating certain preferences correctly. For example, when a travel e-commerce tool asks for a preference on departure time, a user who knows that he has to be at his destination by 2 PM may not be able to translate this into a correct preference on departure time, since this would also require knowing the total travel time.

A study made in 2000 [25] has shown that only 18% of the users felt that they found the best solution, and less than 50% were sufficiently convinced to use search tools.

Based on the view that a decision maker is quite adaptive and constructive in expressing his preference model (see also [28]), we believe that tradeoff is an opportunity rather than an obstacle in the preference elicitation process. That is, through the tradeoff process, we will learn more about our users, and refine the initially stated preference model by adding hidden preferences, and tradeoff rules which resolve preference conflicts. We propose to design decision support systems that provide "any-effort" interactions, in that users increase their effort in making tradeoffs, we will refine the preference model to achieve reasonable accuracy. We show that a preference model based on soft constraints is particularly suitable for such an interaction model. At the same time, constraint-based preference models effectively support an any-attribute/any-order interaction that avoids rigid elicitation procedures. Finally, we show that we can easily support tradeoff analysis using constraint programming.

1.1. Any-Effort Interaction

It is clear that the more effort a user makes in articulating preferences, the more accurate the obtained results are. However, to require a user to make a tremendous effort without giving him the choice of taking an "easy" outcome is not likely to provide a good human computer interaction strategy. A sensible solution is an *any-effort* interaction model for preference elicitation. A user should be able to make little effort (stating only the most obvious preferences), and still obtain a set of reasonable decision outcomes. On the other hand, the interaction model should allow those users who want more accuracy to continue with preference refinement. They should be able to *incrementally* construct preference models, but quit at any time they want. This gives users the total control of the equilibrium between effort and accuracy.

This any-effort interaction model is in line with recent findings in decision behavior theories. Decision makers are found to more likely *construct* preferences, rather than

revealing them as if they possessed innate preferences [27, 36]. They are also more likely to accept outcomes when they are involved in the preference construction process and have an understanding of how the outcome arose, rather than accepting an "optimal" outcome that is given by the system. Secondly, although individuals clearly aim at maximizing the accuracy of their decisions, they are often willing to compromise on decision accuracy in order to reduce cognitive effort [28].

Preference models based on soft constraints are particularly well suited for any-effort interaction, as preference models can be incrementally constructed by adding one soft constraint at a time, and at any time the partial model constructed so far can be used to identify the most preferred outcomes.

Furthermore, soft constraints can be added in any order, and the order in which they are stated does not affect the final result. The formalism also places no limit on the form that soft constraints can take: they can express any criterion that comes to mind as long as it can be formulated using the primitives of the interface. Thus, they are a good basis for implementing the any-attribute/any-order elicitation process and avoiding the focus on means objectives that plagues rigid elicitation procedures.

1.2. Importance of Tradeoff

Preference construction is rather straightforward as long as outcomes more or less satisfy all of the users' preferences. However, in most practical situations, there is no outcome that satisfies all preferences. In this case, finding a solution requires making a *tradeoff*: accepting an outcome that is undesirable in some respects but advantageous in others. In fact, the presence of such preference conflicts is a fundamental aspect of decision processes [28]. Human decision makers are observed to use more rational and accurate decision strategies, called compensatory strategies, for confronting these conflicts by making explicit tradeoffs based on processing all relevant information (see [19, 28]).

A preference model must not only indicate what preferences to apply, but also how to make tradeoffs when not all of them can be satisfied. The tradeoffs that people are willing to make are an important indication of their preference model; in fact, many procedures for eliciting utility-based preference models are based on testing people's reactions to possible tradeoffs [18, 23]. However, they use a rigid question-answering approach that is not compatible with an any-effort system.

Instead, a constructive preference elicitation system should support various tradeoff *strategies* and exploit them to update the preference model. A tradeoff strategy is a sequence of actions where a user refines the preference model to make its tradeoffs compatible with his own. From observing user behaviors in our task analysis (see Section 2), we have identified the following 3 strategies:

- *value* tradeoff: the user changes the preference value of a particular attribute value combination, for example stating a preference for a certain airline.
- *utility* tradeoff: the user changes the weight of a preference in the combined ranking.
- *outcome* tradeoff: motivated by a certain outcome, the user adds additional preferences that increase the utility of that outcome; for example, seeing a particularly favored airline in an outcome reminds him to state an airline preference.

Note that all of these strategies refer to a decomposition of the preference model into individual preferences that can be manipulated independently. Decompositions into attributes are used in multi-attribute utility theory (MAUT, [23]), as well as in strategies investigated in behavioral decision theories such as the "elimination by aspects" strategy [39] that selects an outcome by considering its attributes in a strict priority order. A preference model based on soft constraints is a better basis for supporting tradeoff decision strategies because it allows a preference model to be decomposed into more general preferences, each represented by a soft constraint. In contrast to earlier models that limit such decompositions to individual attributes, constraints can be complex and involve several attributes as well. They offer the flexibility required to correctly express user's tradeoff preferences.

1.3. Example Critiquing

Tversky and Simonson [40] showed that users' preferences are largely context-dependent. This contradicts established belief in the theory of rational choice that preference between options does not depend on the presence or absence of other options. To account for this phenomenon, the decision support system should provide many competing choices for users to examine, rather than showing only one or a few "optimal" outcomes based on the preference model. At the same time, this exploits tradeoff opportunities for eliciting preferences, rather than asking users to state preferences without providing them any context.

Our proposal for an interactive decision support system that satisfies these requirements is *example critiquing*. A system implementing this interaction model elicits a partial preference model and generates a set of "example" outcomes. The user can accept one of these example solutions, in which case the interaction stops (any-effort). Otherwise, he can indicate what is wrong with one or several of the example outcomes by formulating critiques. Critiquing can be performed either by adding additional preferences, or by following one of the tradeoff strategies mentioned above. When preferences are modeled by soft constraints, these tradeoff strategies can be implemented by either 1) revising the current set of soft constraints, i.e., adding and/or retracting constraints, or 2) changing the weight of a soft constraint, such as "I really want to fly with Lufthansa." This leads to an interaction model that implements the example-critiquing model in a simple yet very effective manner.

Through these successive reactions to shown examples, users make more effort to express preferences without feeling burdened. The system, at the same time, should support the following actions:

- revising users' preferences;
- providing tradeoff scenarios to resolve users' preference conflicts;
- revising the importance attached to preferences; and
- eliciting hidden preferences

This paper explains how constraint satisfaction techniques as an underlying decision framework successfully implement this any-effort process for constructing preferences, and how effectively it supports tradeoff analysis. We organize the rest of this paper as follows. Section 2 describes how we derived a reference task model for example

critiquing and explains the example critiquing interface, including how each interaction changes preferences and utility functions. Section 3 formalizes the decision framework as a constraint programming problem, where preferences expressed by the user are mapped to soft constraints, and shows how the three kinds of tradeoff strategies are modeled in this framework. Section 4 describes a comparative user study in order to show how significantly example critiquing improves user's performance in terms of tradeoff tasks. Section 5 on related works describes several other example-based interfaces and how they compare to this work. It also reviews several activities in the constraint problem solving domain concerning user interaction issues. Section 6 concludes this paper by outlining the advantages offered by the example critiquing model using constraint programming as a decision framework in eliciting preferences.

2. Task Models

To construct systems for users, we must first understand how they perform tasks. This process, known as task analysis, is the first step in interaction design. The objective is to derive a task model for the interaction to be constructed. We combine several efforts from user scenario analysis, interviews, and surveys of existing systems.

For scenario analysis, we write scripts such as the following one in the domain of travel planning:

> George Smith is a business traveler who lives near Lausanne, Switzerland. He is closer to the Geneva airport than the one in Zurich, and the trains to Geneva are less crowded in the morning and offer better services. He wants to be in Hamburg on a particular day for a meeting starting at 2 o'clock in the afternoon. So far he is able to articulate three strong preferences: the date of travel, the arrival time, and the arrival airport. He is ambivalent about the departure airport, because it depends on the availability of flights out of Zurich and Geneva respectively, and the transit time. His preference over these two airports also depends on whether he could come back the same day, the comfort of the train ride to the airports, and etc. He is unsure about the exact time to leave Hamburg, because even though he likes to come back on the same day, he'd rather leave a sufficient amount of time in Hamburg. If the last flight is too late for making the last train, he'd rather stay over night in Hamburg. The preference of airlines also depends on whether they offer good connections at intermediate airports, and price. In summary, he feels that he could only state three strong preferences, and would only state the rest of preferences in a future decision context, because they depend so much on information from the underlying catalog and preferences on other attributes or constraints.

Most of the user scenarios came from interviewing users and observing how they make decisions in existing systems [21] and the early prototypes that we have built. Our initial task model was used to build various prototypes for multi-attribute product search and decision systems in several domains: conceptual design, resource allocation, travel

planning, and vacation package selection. Cognitive walkthroughs were performed in each prototype system in order to discover discrepancies between user's task flow and the task model. After some iteration, the task model has achieved a stable condition, which we call the reference model. This reference model has been recently implemented in an apartment search tool, which was used in our user studies (see Section 5 for details).

2.1. Reference Model for Decision and Tradeoff

Users first express constraints and strong preferences, such as "I must arrive in Hamburg by 2 PM," and "I strongly prefer to fly with Lufthansa." At this point, they would like to see example solutions. Reactions to inaccuracies in the examples can be naturally expressed as critiques, such as "I would prefer a later departure," or "I would prefer to transit in Munich (rather than Frankfurt)." Such critiques often imply tradeoff decisions: when a later departure is preferred, other preferences such as the arrival time are likely to suffer. In choosing the outcome, a process called tradeoff resolution, the user is volunteering useful information about his true preferences. We therefore view critiquing as a tradeoff navigation process, and use tradeoff resolution as a way to elicit preferences (Figure 1).

In each tradeoff navigation, a user is performing one of three kinds of tradeoff:

- *value* tradeoff: the user changes one of the preference values in his preference model, increasing or decreasing the relative utility of a certain value combination in comparison with the others. For example, a traveler may find that flights with his preferred airline are too expensive, and would like to change his preference to an acceptable one with lower fares.
- *utility* tradeoff: the user changes the weight of a preference in the combined ranking. For example, a traveler may make airline preferences less important.
- *outcome* tradeoff: the user adds additional preferences so to increase the relative utility of certain outcomes to make them acceptable. For example, faced with equally good solutions that differ in transit airports, a traveler may notice a difference in shopping

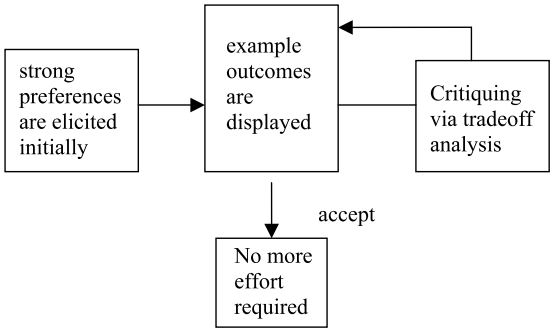

Figure 1. Reference model for tradeoff decision.

facilities at the transit airport and add this as an additional criterion, thus making one of the outcomes stand out.

2.2. The Example Critiquing Interaction Model

Figure 2 shows a typical example critiquing interface implemented in a decision aid tool for apartment search. We will focus on the interface for apartment search from hereon because it has been most recently implemented and it conforms closest to the reference model.

The ''search panel'' is an area for specifying initial queries, as well as posting critiques. In Figure 2, a user has entered a set of preferences. He used default weights for all preference variables except for price and distance where he selected the ''most important'' weight value. At this point, the system returned 7 recommended apartments.

These examples are selected with respect to the user's current preference model. They reveal domain knowledge, show partially satisfied outcomes in the case of conflicting preferences, and provide a basis for further critiquing. If he chooses to critique one of the outcomes, the interface will first show a pop-up window (Figure 3) where he can compare his current selection with others and perform tradeoff analysis. For example, suppose that the current selection is apartment 34. In the comparison window, he can specify his desire

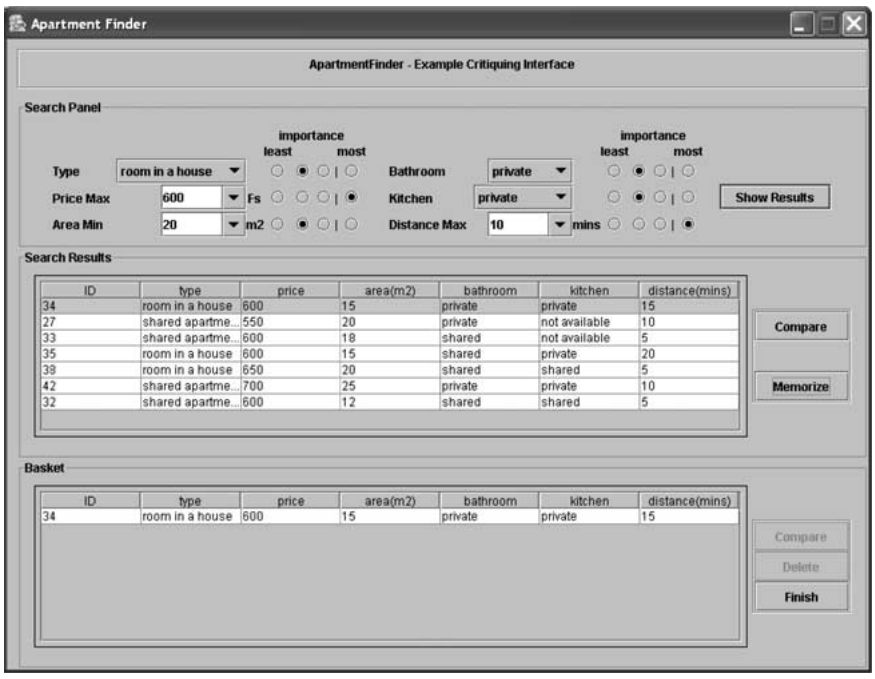

Figure 2. Step one in example critiquing: System showing a set of 7 results after a user query.

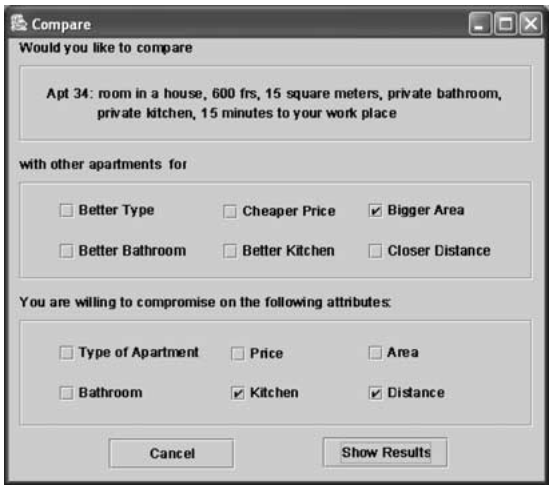

Figure 3. Step two in example critiquing: Guiding users to find tradeoff alternatives in the product comparison pop-up window.

for a bigger apartment by selecting the "bigger" menu option next to the Area (m^2) label. However, knowing that he may sacrifice something for a bigger apartment, he specifies "compromises" for both distance and kitchen attributes by selecting the checkboxes next to these two attributes.

Once a set of critiques has been composed, the system will show another set of matching examples (Figure 4). Apartment 31 seems to him quite interesting, since it is around the same price, but 5 square meters bigger, although it is 10 minutes more commuting time and he has to share the bathroom. The system does not resolve tradeoffs for the user, but provides enough information for him to understand his decision context. The final choice is left up to the user to make.

Previous versions of our system used the name "modify" instead of "compare" as a label for the tradeoff process. User studies found that the name was not very suitable because people associate the word "compare" rather than the word "modify" with tradeoff processes.

When stated preferences are in conflict, such as when a user wants a large apartment and has entered a rather tight budget, the system resolves it by generating partially satisfied outcomes. It would show two examples, each satisfying either the size or the budget constraint, but not both. This approach requires less effort from the user than systems that simply indicate that no solution has been found, or those which require the users to explicitly change preference values without contextual knowledge.

In addition to generating partial solutions, we must also carefully consider the following design issues: 1) there must be a certain diversity to make the DM recognize tradeoff opportunities; 2) there must be sufficiently many optimal solutions so that the DM can select the most preferred one in spite of inaccuracies in the model; and 3) there is a limit on the number of solutions so that he is not overwhelmed by the information.

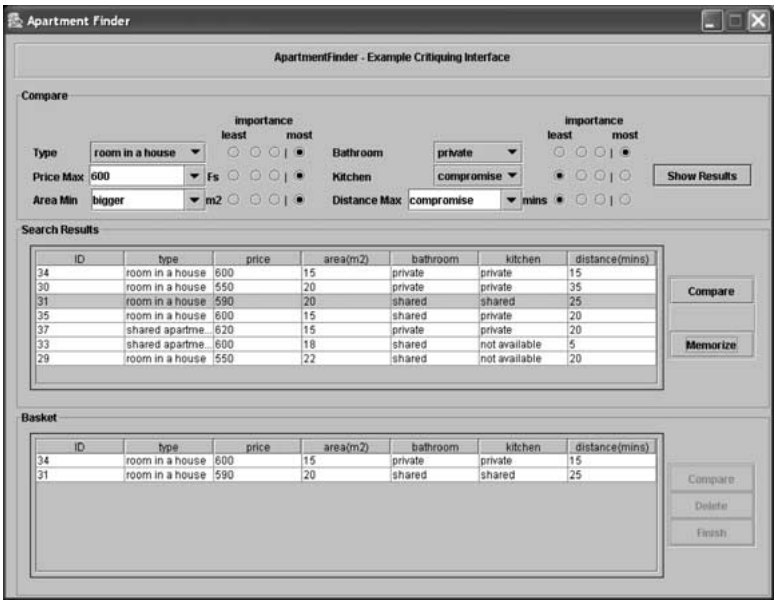

Figure 4. Step three in example critiquing: The system showing tradeoff alternatives.

These topics are discussed in more detail in [11, 12]. In this paper, we focus our attention on how the example/critiquing interaction model is implemented using the constraint framework, and show via our user studies that it is more suitable to decision tasks, especially tradeoff tasks, than the standard ranked list model.

3. A Decision Framework Based on Constraint Programming

We now consider how to model preferences as soft constraints and argue that this approach makes it easier to support tradeoff strategies than classical utility functions.

In a multi-attribute decision problem, outcomes are modeled using a set of *attributes*. We model both component and feature attributes. A component attribute is a physical aspect of a choice (e.g., the screen size of a PC), while a feature attribute is an abstract concept such as the suitability of a product for outdoor use (good, bad). A user makes decisions by considering a set of *criteria* that involve these attributes. Each criterion implements a stated preference. For example, a criterion could be how tiring a trip is likely to be, or how well its schedule fits the tasks to be accomplished. Each criterion is a function of one or several attributes. Many criteria are simple functions of a single attribute: for example, whether the arrival time is early enough for a 18:00 meeting, or whether the airline fits the user's preference. Others can be more complex; for example, how tiring a trip is depends on the total travel time, the departure and arrival times, the number of stops, etc.

A constraint satisfaction problem (CSP) is characterized by a set of n variables $X_1,\ldots,X_n$ that can take values in associated discrete domains $D_1,\ldots,D_n$, and a set of m hard constraints $C_1,\ldots,C_m$, which we assume to be either unary (one variable) or binary (two variables), and which define the tuples that are allowed for the variable or pair of variables. Solving a constraint satisfaction problem means finding one, several or all combinations of complete value assignments such that all hard constraints are satisfied. Besides hard constraints (also known as feasibility constraints) that can never be violated, a CSP may also include soft constraints. These are functions that map any potential value assignment into a numerical value that indicates the preference that this value or value combination carries. Solving a CSP with soft constraints also involves finding assignments that are optimally preferred with respect to the soft constraints. There are various soft constraint formalisms that differ in the way the preference values of individual soft constraints are combined (combination function). For example, in weighted constraint satisfaction [16, 34], the optimal solution minimizes the weighted sum of preferences. In fuzzy constraint satisfaction [33], the optimal solution maximizes the minimum preference value. Most approaches can be understood as an instantiation of a semiring [3]. In our systems, we have almost always used the weighted model as we found that it corresponds best to users' intuition about the compensatory nature of tradeoffs [37]. Thus, in the following we assume the weighted constraint satisfaction model.

3.1. Need for Agility in Preference Management

During a decision process, an adaptive decision maker may reconsider the criteria and change their relative importance. For example, the first criterion for arrival time may be in relation to the meeting at 18:00. Later on, he may prefer an arrival that also avoids the rush hour between 16:00 and 19:00. A computer aided decision system should be *agile*, i.e. support such changes with minimal user effort.

In classical decision theory, outcomes are ranked according to a utility function formulated on attributes [23]. For example, a utility function on 3 attributes a_1, a_2 and a_3 could be specified as:

$$U(a_1,a_2,a_3) = w_1f_1(a_1) + w_2f_2(a_2) + w_3f_3(a_3) + w_{12}f_{12}(a_1,a_2) + w_{13}f_{13}(a_1,a_3) + w_{23}f_{23}(a_2,a_3) + w_{123}f_{123}(a_1,a_2,a_3)$$

The functions (f_1, f_2, f_3, $\ldots f_{123}$) are called value functions, and each function maps one or a set of attributes to a real number representing their utility. For most decision makers, not all terms need to be present. For example, each attribute might make an independent contribution to the utility of an outcome and in this case only the first three terms would be present. Note that if there are several criteria on the same attributes, they would be combined into a single function. When certain independence assumptions hold, decision theory gives procedures where the weights w and the functions f can be determined based on systematic queries to the decision maker. However, the problem with this framework is that several criteria might be part of the same value function. When criteria change, the elicitation process has to start over again. Thus, it is hard to construct an agile decision support tool.

3.2. Soft Constraints for Agile Preference Models

We apply the constraint satisfaction framework to multi-attribute decision problems by formulating each criterion as a separate constraint. Using the weighted CSP model, we are able to express the general format of a utility function in classical decision theory, where each of the value functions is a soft constraint. However, the CSP approach differs from the classical approach when there are several criteria expressing the utility of the same attribute, especially when criteria come and go in the preference expression. For example, a user may initially consider the criterion that the arrival time needs to be early enough for the 18:00 meeting, and this would be formulated as a constraint. When he later on considers that he wants to avoid arriving during the rush hour, he can add this as an additional constraint on the arrival time. In a classical preference elicitation approach, it would be necessary to re-elicit the combined value function for the arrival time attribute.

Hard constraints can also be modeled in the same framework. However, we mainly use them to express integrity conditions that must be satisfied in the configuration process. For example, integrity constraints would state that there must be at least 45 minutes for changing flights within Europe, and 60 minutes for international ones. Integrity constraints are part of domain knowledge, and are elicited from domain experts during the construction of the system.

The constraint programming framework replaces preference elicitation by asking specific valuation questions with preference construction where users manipulate individual constraints. Some of these constraints could be inferred from a personal profile or using methods such as collaborative filtering, demographical analysis, default logic, and case-based reasoning [18]. For example, knowing that a traveler is booking a business trip, we can infer that his most important criterion is to be at his destination on time. However, most constraints will be specific to the particular scenario, and need to be explicitly stated by the user. In this paper, we focus on user stated preferences.

Allowing users to state arbitrary criteria as soft constraints is a significant user interface challenge. We have found it sufficient to decide a certain number of useful parameterized criteria during system design and make these accessible through the user interface. While this choice limits the preferences that can be stated by a particular user, the framework itself places no limit on what criteria can be used for preferences. Depending on how much interface complexity a user is willing to accommodate, it is possible to provide a very rich vocabulary for formulating them; for example, a travel planning system we have built allows users to specify unary and binary constraints on any combination of attributes visible in the display. This flexibility is an important advantage over database-based systems where the set of possible preferences is strongly restricted by the schema of the underlying database.

For each criterion, the system designer devises a parameterized function (criterion function) that maps each value combination of the involved attributes to a numerical value. This function is chosen to fit the majority of users. For example, in apartment search, price differences are counted linearly, while preferences for location are counted with a step function that assigns 0 utility to non-preferred values and utilities in increasing steps of 100 to the more preferred ones. Each preference has a weight that the user is able to change. The function is inaccurate because it is standardized for all users. However, we

have shown in earlier work that such inaccuracies can be compensated by generating a set rather than a single optimal solution [11].

3.3. Preference Revision Via Tradeoffs

We now consider how to support preference revision via tradeoff in a decision framework. In traditional decision theory, a preference *elicitation* process poses queries to the user. These queries are based on the assumption that for all users, the same terms are present in the utility function. Consequently, the function would include many terms and thus require many elicitation steps. More seriously, the process assumes that the user does not change his preferences during the elicitation process, thus ruling out tradeoffs during the decision process. Contrarily, in a soft constraint model, the user can *construct* his ranking function by manipulating the different criteria directly. Tradeoffs give rise to individual, relatively simple soft constraints that can be easily added and removed. This makes it easy to construct a preference model that focuses the user's effort only on specifying preferences for the parts of the outcome space where tradeoffs are actually required.

As an example, consider the following choices for apartments (Table 1):

A user may state the following 3 initial criteria:

1. (weight = 1): *I am willing to pay up to 700.*:

$$\text{u1} = 700 - \text{price}$$

2. (weight = 20): *Surface has to be at least 30.*:

$$\text{u2} = \text{Surface} - 30$$

3. (weight = 100): *I work in the center, so I prefer Center over Morges over Renens*:

$$\text{u3} = (\text{case location} : \text{Center} : 2, \text{Morges} : 1, \text{Renens} : 0)$$

We assume that preferences are combined by forming a weighted sum, so that the initial ranking assigned to the 4 outcomes would be:

- u (1) = −100 + 20*−5 + 100*2 = 0
- u (2) = 100 + 5*−6 + 0 = −20
- u (3) = −200 + 0 + 100*1 = −100
- u (4) = −100 + 20*5 + 0 = 0

Table 1. Several apartment choices

	Price	Surface	Location	Bus
1	800	25	Center	12 min
2	600	24	Renens	15 min
3	900	30	Morges	8 min
4	800	35	Renens	2 min

Outcomes 1 and 4 are preferred over 2 and 2 is preferred over 3. However, none of the outcomes provides a positive utility, so the decision maker is not likely to pick any of them as a final solution.

Tradeoffs are required whenever none of the outcomes achieves a sufficiently high ranking according to the preference model. This then requires a user to change the preference model so that an acceptable solution can be found. In the constraint-based decision framework, the three different tradeoff strategies described in the introduction can be modeled as follows:

1. Value tradeoff: the user changes one of the soft constraints in his preference model, increasing or decreasing the utility of a certain value combination in comparison with the others.
2. Utility tradeoff: the user changes the weight that a particular soft constraint is given in the combined ranking.
3. Outcome tradeoff: the user adds additional soft constraints that increase the relative utility of certain choices to make them acceptable.

In the example, none of the outcomes is ranked with a utility greater than his threshold of 0. Now the decision maker can engage in the three kinds of tradeoff:

1. an example of a value tradeoff would be to reverse the preferences for location after realizing that Morges offers the best school for the decision maker's children and thus should be most preferred. This can be done by adding a soft constraint with a weight of 200 that gives a value of 1 to Morges and none to the other locations. Since both soft constraints are added up, this has the effect of making Morges the most preferred location, but requires no revision of the earlier soft constraints. Now, the third solution has a value of 100 and is thus acceptable.
2. an example of a utility tradeoff would be to change the weight of surface preference from 20 to 30. Now choice 4 has a positive utility of 50 and could be chosen. This can be done by modifying the weight of the preference in the weighted combination function.
3. an example of an outcome tradeoff would be to notice that choice 4 is in fact very close to public transport, and add the following additional preference with weight 10:

   u4 = (if Bus <3 min: 10, otherwise 0).

   Now choice 4 has a positive utility of 100 and is acceptable.

Note that in each case, the required modification of the ranking is achieved by adding a preference or changing its weight in the combination model. In the soft constraint model, these kinds of modifications are easy to carry out and provide an agile modeling process. Such agility would be very hard to achieve in models based, for example, on classical preference elicitation, where the impact of answering a particular query is hard to evaluate.

Note furthermore that each of these three tradeoff types requires that the decision maker has a good understanding of the available outcomes and how they are affected by preferences. This understanding is provided, at least to some degree, by the

example-critiquing framework: by showing examples that are already close to the user's desires, it increases his understanding of the possibilities in the outcome space that is relevant to his desires. In further work, we have also shown techniques for generating additional examples that are likely to stimulate users to express preferences that have not been stated so far [12]. These methods are based on a probabilistic analysis of what examples are likely to become optimal should the user discover new criteria.

Conversely, the constraint-based decision framework provides the agile and easily modifiable preference model that is required to implement example critiquing. The synergy between the two models leads to powerful practical systems.

4. Evaluating Example Critiquing

To be certain that example critiquing is indeed an adequate tool for decision tradeoff analysis and that it increases a user's task performance, we have conducted a user study comparing user's performance using example critiquing against the leading interaction model on most e-commerce sites, a ranked list. A ranked list gives users the option of viewing outcomes ordered according to one or possibly several predefined ranking criteria, most commonly price.

Several criteria were important in choosing the ranked list as the model to compare. First, the model must offer a way to allow users to express preferences; second, it must provide users with partially satisfied solutions for resolving tradeoff conflicts; third, the model is commonly used in current decision environments for finding and comparing a set of outcomes. We did not choose a search tool based on a Boolean search method because it does not handle cases of conflicting preferences. We opted for the ranked list because it allows a user to compare results by sorting them on any selected attribute, one at a time. The selected attribute is similar to expressing a strong preference on that attribute. For example, if a user selects the price to sort, then his current preference on price is the strongest. If arrival time later becomes more important, he would examine all the outcomes using that attribute. It lists all outcomes so that partially satisfied ones are automatically displayed. Tradeoff navigation, in this case, is done by visually scanning the entries. For example, looking for a cheaper apartment involves scanning entries earlier in the price-ordered ranked list. The scanning requires more and more effort if a user wants to tradeoff more of one attribute for less of several attributes (i.e., distribute the losses). Finally, a ranked list is the current norm in online environments for comparing products. Comparing example critiquing against the ranked list is to measure how much performance gain we are able to obtain by supporting tradeoff navigation explicitly.

We chose the apartment search domain for this study, because our participants (see details below) were more likely to be familiar decision tasks in this domain rather than the travel domain. Each user would be using the ranked list and example critiquing interfaces to perform the same set of tasks respectively. The order in which the tools were given to the user always alternates in order to balance out any biases. Furthermore, the entries in these two tools were not identical, but equivalent, to avoid any learning effects. The rental

properties used in the experiment were based on real data with slight modification. For instance, each property, regardless of its type, was normalized for the purpose of accommodating one person only.

All of the 22 (8 females) participants were recruited from our university (EPFL), consisting primarily of graduate students, who were looking for places to stay. Since EPFL does not provide dormitory rooms for graduate students, apartment search is a familiar task, and is sometimes done via online environments. Significant attention has been given to make the participant group as diverse as possible. Our participants have been selected from different nationalities (Swiss, Algerian, American, Indian, Vietnamese, Chinese, Mexican), and different educational background (undergraduate students, graduate students, research assistants).

4.1. Experimental Procedure

Before the actual user study, each participant was debriefed about the objectives of the experiment, the meaning of labels on each of the interfaces, and the fact that we will record their task performances. We then gave them a 5–10 minutes warm-up period to get familiar with the tools.

The objective of this experiment was to measure the task performance and error rate while users perform tradeoff navigations in example critiquing (henceforth EC) and ranked list (henceforth RankedList) interfaces respectively. Since users were likely to have different preferences for choosing apartments, we would measure their tradeoff tasks performance by instructing them to carry out a set of generic tasks. An error in task performance was defined as a wrong response provided by a particular user against the correct one previously determined.

The detailed tasks are follows:

1. Find your most preferred apartment.
2. Can you find something closer? You can compromise on one and only one attribute.
3. Can you find something bigger than what you found for question #1? You can compromise on one and only one attribute.
4. Find something which is roughly 100 francs less than the answer to question #1. You can compromise on up to two attributes, but not more.
5. Find an apartment which is 5 square meters bigger than the answer to question #1. You can compromise on up to two attributes but not more.

The questions can be broadly divided into three categories. The first question was an identification task of a multi-attribute product from a list of products, allowing users to express freely their preferences. In addition, this question gave an idea of the user's comfort level with the interfaces and a starting point for answering subsequent tradeoff questions. The second category of questions (Question 2, 3) dealt with multi-attribute tradeoff tasks with one attribute in each direction of gain and compromise. The third category of questions (Question 4, 5) dealt with making tradeoff tasks when a user gains on one attribute, and compromises on up two attributes. The test results were summarized in Figure 3 and 4 for task performance and error rate data respectively.

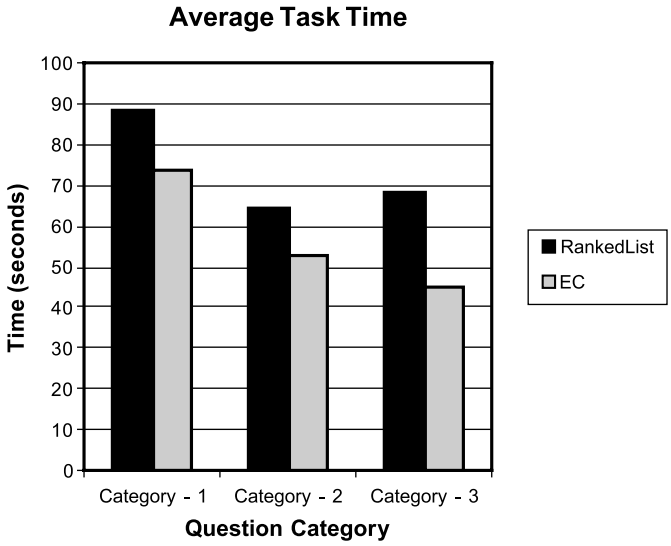

Figure 5. Average task completion times in seconds for the three categories of tasks evaluating RankedList and EC respectively.

4.2. Multi Attribute Searching Task

The first question required users to find an apartment of their choice. Although our data (Figures 5 and 6) showed that there were no significant improvement of response time for answering that question using EC interface ($p = 0.382$), we believe that this was largely due to the fact that participants took much longer to get familiar with the new interface than the ranked list interface, especially under testing conditions. There were no errors recorded in either interfaces, since we do not know what the user's most preferred solution is.

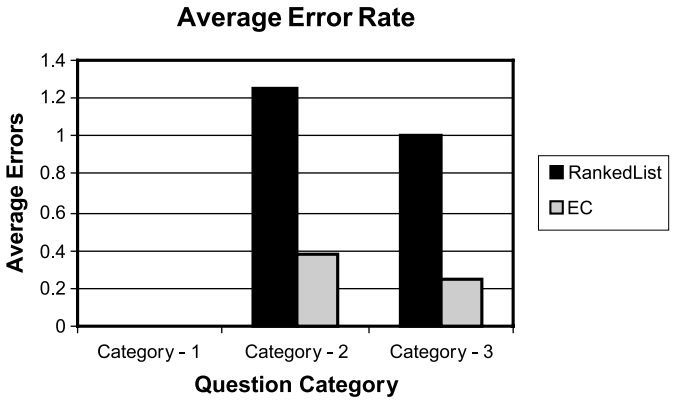

Figure 6. Average error rates for the three categories of tasks evaluating RankedList and EC respectively.

4.3. Trade-off with 2 Attributes

These questions (#2 and #3) required the participants to achieve a better value on one identified attribute while compromising the values of one of the four remaining attributes. The improvement for response time in using EC was not significant ($p = 0.382$), although improvement of the error rate was much greater ($p < 0.001$). As predicted, the relatively high error rate confirms that participants had to do a significant amount of visual search in RankedList to perform tradeoffs, hence its susceptibility to errors also increased at the same time.

4.4. Trade-off with More Than 2 Attributes

We see an increased complexity in this set of questions (#4 and #5) because users had to perform tradeoff tasks on more than two attributes. An analysis of the statistics gives interesting observations. Not only there was a decrease in average performance times in EC as compared to RankedList ($p < 0.001$), but the overall error rate also dropped by a significant margin (approx. 66.67%).

4.5. Discussion

An overview of the results reinforces our belief that users perform relatively the same in both interfaces for simple identification tasks. Users performed tradeoff tasks much better in example critiquing interfaces than ranked lists, especially when the tradeoff tasks involve an increasingly higher number of attributes. Further, users were more prone to make mistakes using the ranked list interface than using example critiquing.

An interesting finding in the user study was that most users say that they prefer using the ranked list than the EC, although a high number of users who showed higher satisfaction with the ranked list were found to commit more errors using the ranked list. On further questioning, users expressed the fact that they felt more in control in the ranked list because they could see all outcomes. We believe that such opinions will change once the domain is about more configurable products, where the space of all possible outcomes is too huge to browse in a ranked list.

5. Related Work

Several decision support systems have employed similar example-based interfaces for preference elicitation, in particular FindMe [7], ATA [24], and Apt Decision [35]. All of them are concerned with decision support for the search of multi-attribute products. FindMe promotes assisted browsing, revealing domain knowledge in the process, and aims at reducing complexities in the multitude of product dimensions and data sources for users. Tradeoff analyses and tweaking (vs. example critiquing) are two main components from their system that are similar to ours. However, we investigate a precise concept for the minimal critiquing context in which tradeoff can effectively happen, while their system uses the tradeoff component mainly for explanation, and tweaking for adjusting values. Apt Decision uses learning techniques to synthesize a user's preference model through

observing their critiques of apartment features. Important features can be discovered when users browse through the displayed examples as well as using examples to revise their preference models. The main objectives of Apt Decision are, however, preference measurement and prediction using established preference models.

Linden et al. [24] describe a preference elicitation method using travel planning as an example domain. Initially only few user preferences need to be expressed. The ATA system (automated travel assistant) uses a constraint solver to obtain several optimal solutions. Five of them are shown to the user, three optimal ones in addition to two extreme solutions (least expensive and shortest flying time). User preferences are modeled as a kind of soft constraints. A candidate critiquing agent (CCA), similar to our example critiquing, constantly observes user's modification to the expressed preference, and refines the elicited model in order to improve solution accuracy.

All of this earlier work focused on implementation of a system for a particular application, but did not include any theoretical analysis or quantitative user studies. For example, ATA always displayed five solutions. According to our analysis, this number is not diverse enough to guarantee the inclusion of the most optimal solutions, especially when the current user deviates from the standard one (see [11]). Furthermore, by not showing near optimal solutions, users are discouraged from opportunistic and creative discovery of outcomes. In SmartClient Travel [29, 30, 37, 38], we used compact visualization methods to effectively display up to thirty solutions, which can be shown to be more appropriate for this problem.

Another approach to establish a complete preference model is to elicit user's needs rather than their preferences on product attributes. Such approaches (e.g. IBM notebook advisor) were a popular feature on web sites for computers and other complex devices, but the wide adoption of these systems has not happened. A recent study [14] provides some empirical evidence pointing out problems of user acceptance of such needs-oriented elicitation approaches. In particular, people have difficulty understanding why certain product features are associated with their purchase needs.

Qualitative decision theory aims at automating preference modeling and decision making processes, as well as extending traditional decision theories to cover larger contexts. Different approaches have been proposed (see [8] for a detailed survey). Relevant to this is work by Boutilier et al. [5], where conditional preference networks (CP-nets) were introduced as a preference model. CP-nets allow modeling relative importance of preferences and partial orders. The computation of a single optimal solution is rather efficient. However, the work does not appear to address user issues, since it is not clear how easy it is to generate partial solutions when users' preferences are in conflict.

Others have proposed to use constraint programming in interactive systems so that user' input is elicited during constraint solving. For example, Jackson and Havens [20] investigated mixed-initiative CSPs, where users are allowed to fix values during search, and established a set of strategies for integrating such interaction in the solving process. Bowen [6] proposes *Specialization CSP* as a formal notion of interactive CSP, where a solution is given by a set of additional constraints that would make the problem have only a single consistent assignment. We could see these constraints as the missing preferences to be elicited from the user. These approaches will be particularly useful when we consider larger problems where computational complexity becomes an issue. Recently, a more

systematic approach for integrating information gathering and constraint satisfaction has been proposed by Faltings and Macho-Gonzalez [9, 10].

An alternative to the explicit constraint posting used as a feedback mechanism in our systems is to derive these constraints from users' evaluation of individual solutions. O'Sullivan et al. [26] described machine learning techniques to help users formulate new constraints. The process involves asking users to provide acceptable solutions (positive examples) from which the system attempts to generalize the constraints exemplified. Blythe [4] proposed a technique for formulating constraints on the user's utility function via inferring from relative orderings of examples. A similar approach is also taken in the CAWICOMS project [1]. In this work, the system infers new preferences from user reaction to defaults proposed by the knowledge base.

Most relevant to this work concerns tradeoff analysis in a constraint solver. Following an earlier analysis [17], Freuder and O'Sullivan [15] proposed to model tradeoffs as additional constraints. The main goal of their work is to generate appropriate tradeoffs automatically based on preferences. These tradeoffs are formulated as *tradeoff constraints* that would replace some of the constraints in the current problem. The user can then choose which replacement would come closest to his preferences, knowing that each of the proposals will make the problem solvable. Bistarelli and O'Sullivan [2] have developed this approach into a more formal framework for automatically generating possible tradeoffs. This approach is potentially very powerful, but assumes that users choose their tradeoff at the level of constraints rather than on motivating examples. In contrast, in our work users formulate tradeoffs with reference to example solutions. Market researchers have shown this to be essential for correct feedback [27]. Techniques for automatically generating tradeoff constraints can be useful to help users update the preference model once a desirable tradeoff has been identified, or for cases where users are familiar with a domain, for example if they repeatedly solve similar problems.

Finally, some recent work [13] has shown how to use tradeoffs as a starting point to a systematic diagnosis of a knowledge base. Such methods might be advantageous in conjunction with the soft constraint framework presented here in order to better support outcome tradeoffs automatically. Thus, the user could be supported explicitly in the task of modifying the preference model so that a certain outcome would become more or most preferred.

6. Conclusion

In this paper, we have outlined a theoretical framework of decision tradeoff process using soft constraint formalism. It consists of two main components: an interaction paradigm called example critiquing, and a preference modeling mechanism based on constraint satisfaction techniques. The front and back-ends together address both explicit and implicit tradeoffs. A user can perform tradeoff navigation via critiquing the current example (explicit tradeoff), or let the backend system generate a set of partial matches (implicit tradeoff). According to Payne et al., a framework that induces decision makers to perform tradeoffs is likely to be more successful at improving their decision outcomes. We have therefore conducted quantitative user studies in order to validate our belief that example

critiquing offers a significant performance gain for explicit tradeoff analysis than the most commonly used ranked list.

More concretely, we described the example critiquing paradigm in detail, showing how a preference elicitation problem can be turned into a tradeoff task. A user can then resolve each tradeoff task using any of the three strategies: value tradeoff, weight revision, and outcome tradeoff. By doing so, the system allows a user to flexibly balance between effort and accuracy. We have further shown how user preferences can be modeled using constraint satisfaction techniques using both hard and soft constraints. These preferences will be subject to revisions, depending on the examples offered. A decision aid using a constraint programming (CP) model has the potential to achieve several desirable properties:

- The interaction with the decision maker (DM) is *agile*. That is, there is no restriction on what preferences users can express, nor on the order they express them. This property, *any criteria/any order*, does not always hold if other paradigms are used as an underlying framework for decision support systems [31].
- The interaction with the DM is *robust*. When users state conflicting preferences, a CP engine with soft constraints can generate partially satisfied solutions, instead of giving a disappointing message such as "no solution is found," or asking them to diagnose the conflicts.
- The DM is *in control*. A CP engine based on soft constraints does not require that any particular set of preferences be specified. On a related note, a DM who prefers less decision effort can find a (possibly suboptimal) solution with very little effort.
- The DM will find solutions *easier to accept*, as preferences to resolve tradeoffs can be made explicit as soft constraints and will have been explicitly stated by the DM.

Several applications of our approach were implemented in the past few years. COMIND was the first such system implemented for configuration design [32]. Although it did not employ soft constraints to model the constraint solver, it emphasized solution visualization and utility function. SmartClient for travel planning was the first system that embraced the example interaction model entirely [29, 30, 38]. It was based on a lightweight architecture for configuring products whose components were from distributed catalogs. SmartClient used constraints to build the initial decision space, and based its subsequent outcome computation on user preferences. Example outcomes were offered to provoke user's critiquing. VacationPlanner emphasized more on the browsing and search aspect [22], experimenting how such interaction models resolve tradeoff conflicts. Finally, we developed the apartment search tool for performing quantitative user studies.

A recurring theme in our accumulated experience with decision systems is that we must help users model themselves, rather than devising algorithms to model them. In supporting decision tradeoff, we turn the preference model incompleteness and user belief uncertainty into an opportunity for preference discovery.

References

1. Ardissono, L., Felfernig, A., Friedrich, G., Goy, A., Jannach, D., Petrone, G., Schäfer R., & Zanker, M. (2003). A framework for the development of personalized, distributed web-based configuration systems. *AI Mag.* 24(3): 93–110.

2. Bistarelli, S., & O'Sullivan, B. (2004). A theoretical framework for tradeoff generation using soft constraints. In *Proceedings of AI-2003, the Twenty-third SGAI International Conference on Knowledge-Based Systems and Applied Artificial Intelligence*.
3. Bistarelli, S., Montanari, U., & Rossi, F. (1997). Semiring-based constraint solving and optimization. *J. ACM* 44: 201–236.
4. Blythe, J. (2002). Visual exploration and incremental utility elicitation. In *Proceedings of the National Conference of the American Association for Artificial Intelligence (AAAI)*, pages 526–532.
5. Boutilier, C., Brafman, R., Domshlak, C., Hoos, H., & Poole, D. (2004). CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *J. AI Res.* 21: 135–191.
6. Bowen, J. (2001). The (minimal) specialization CSP: A basis for generalized interactive constraint processing. In *Proceedings of Workshop on User-Interaction in Constraint Processing at CP-2001*.
7. Burke, R., Hammond, K., & Young, B. (1997). The findme approach to assisted browsing. *IEEE Expert* 12(4): 32–40.
8. Doyle, J., & Thomason, R. (1999). Background to qualitative decision theory. *AI Mag.* 20(2): 55–69.
9. Faltings, B., & Macho-Gonzalez, S. (2002). Open constraint satisfaction. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP 2002)*, pages 356–370. Springer LNCS 2470.
10. Faltings, B., & Macho-Gonzalez, S. (2003). Open constraint optimization. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP 2003)*, pages 303–317. Springer LNCS 2833.
11. Faltings, B., Torrens, M., & Pu, P. (2004). Solution generation with qualitative models of preferences. *Comput. Intell.* 20(2): 246–263.
12. Faltings, B., Pu, P., Torrens, M., & Viappiani, P. (2004). Designing example-critiquing interaction. In *Proceedings of the International Conference on Intelligent User Interfaces*, pages 22–29. ACM Press.
13. Felfernig, A., Friedrich, G., Jannach, D., & Stumptner, M. (2004). Consistency-based diagnosis of configuration knowledge bases. *Artif. Intell.* 152(2): 213–234 (February).
14. Felix, D., Niederberger, C., Steiger, P., & Stolze, M. (2001). Feature-oriented vs. needs-oriented product access for non-expert online shoppers. In *Proceedings first IFIP Conference on e-commerce, e-business, and Government (I3E)*, pages 399–406.
15. Freuder, E. C., & O'Sullivan, B. (2001). Generating tradeoffs for interactive constraint-based configuration. *Seventh International Conference on Principles and Practice of Constraint Programming (CP 2001)*, pages 590–594. Springer LNCS 2239.
16. Freuder, E., & Wallace, R. (1992). Partial constraint satisfaction. *Artif. Intell.* 58: 21–70.
17. Freuder, E., Likitvivatanovong, C., & Wallace, R. (2000). A case study in explanation and implication. In *Proceedings of CP-2000 Workshop on Analysis and Visualization of Constraint Programs and Solvers*.
18. Ha, V., & Haddawy, P. (1997). Problem-focused Incremental Elicitation of Multiattribute Utility Models. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, pages 215–222 (August).
19. Hogarth, R. (1987). *Judgment and Choice: The Psychology of Decision*, 2nd Edition. John Wiley.
20. Jackson, W. K., & Havens, W. S. (1995). Committing to user choices in mixed initiative CSPs. In *Proceedings of the Fifth Scandinavian Conference on Artificial Intelligence*, pages 239–248.
21. Jurca, A. (2000). *Survey of Online Travel Planning Systems*. Technical Report, Swiss Federal Institute of Technology, Lausanne.
22. Jurca, A., & Pu, P. (2001). *Algorithms for Online Vacation Planning*. Technical Report, Swiss Federal Institute of Technology, Lausanne, pages 1–12.
23. Keeney, R., & Raiffa, H. (1976). *Decision with Multiple Objectives: Preferences and Value Tradeoffs*. Cambridge University Press.
24. Linden, G., Hanks, S., & Lesh, N. (1997). Interactive assessment of user preference models: The automated travel assistant. In *Proceedings of User Modeling '97*, pages 67–78.
25. Morgan Stanley Dean Witter Equity Research Europe: *Transportation E-Commerce and the Task of Fulfilment*, March 2000.
26. O'Sullivan, B., Freuder, E., & O'Connell, S. (2001). Interactive constraint acquisition. In *Proceedings of Workshop on User-Interaction in Constraint Processing at the CP-2001*.
27. Payne, J. W., Bettman, J. R., & Johnson, E. J. (1992). Behavioral decision research: A constructive processing perspective. *Annu. Rev. Psychol.* 43: 87–131.

28. Payne, J. W., Bettman, J. R., & Johnson, E. J. (1993). *The Adaptive Decision Maker*. Cambridge University Press.
29. Pu, P., & Faltings, B. (2000). Enriching buyers' experiences: The SmartClient approach. In *Proceedings of The ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 289–296. ACM Press.
30. Pu, P., & Faltings, B. (2002). Personalized navigation of heterogeneous product spaces using SmartClient. In *Proceedings of the International Conference on Intelligent User Interfaces*, pages 212–213. ACM Press.
31. Pu, P., & Faltings, B. (2002). Effective interaction principles for user-involved constraint problem solving. *Second International Workshop on User-Interaction in Constraint Satisfaction, the Eighth International Conference on Principles and Practice of Constraint Programming*, (September).
32. Pu, P., & Lalanne, D. (2002). Design visual thinking tools for mixed initiative systems. In *Proceedings of the International Conference on Intelligent User Interfaces*, pages 119–126. ACM Press.
33. Ruttkay, Z. (1994). Fuzzy constraint satisfaction. In *Proceedings of the Third IEEE International Conference on Fuzzy Systems*, pages 1263–1268.
34. Schiex, T., Fargier, H., & Verfaillie, G. (1995). Valued constraint satisfaction problems: Hard and easy problems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 631–639.
35. Shearin, S., & Lieberman, H. (2001). Intelligent profiling by example. In *Proceedings of the Conference on Intelligent User Interfaces*, pages 145–151. ACM Press.
36. Slovic, P. (1995). The construction of preference. *Am. Psychol.* 50: 364–371 (August).
37. Torrens, M. (2002). Scalable Intelligent Electronic Catalogs. Ph.D. Thesis No. 2690, Swiss Federal Institute of Technology.
38. Torrens, M., Faltings, B., & Pu, P. (2002). SmartClients: Constraint satisfaction as a paradigm for scaleable intelligent information systems. *Int. J. Constraints* 7: 49–69.
39. Tversky, A. (1972). Elimination by aspects: A theory of choice. *Psychol. Rev.* 79: 281–299.
40. Tversky, A., & Simonson, I. (1993). Context-dependent preferences. *Manage. Sci.* 39(10): 1179–1189 (October).