Implementing Example-based Tools for Preference-based Search *

Paolo Viappiani
Artificial Intelligence Laboratory (LIA)
Ecole Polytechnique Fédérale de Lausanne (EPFL)
1015 Lausanne, Switzerland
paolo.viappiani@epfl.ch

Boi Faltings
Artificial Intelligence Laboratory (LIA)
Ecole Polytechnique Fédérale de Lausanne (EPFL)
1015 Lausanne, Switzerland
boi.faltings@epfl.ch

ABSTRACT

Preference-based search is the problem of finding an item that matches best with a user's preferences. User studies show that example-based tools for preference-based search can achieve significantly higher accuracy when they are complemented with suggestions chosen to inform users about the available choices. We present `FlatFinder`, an implementation of an example-based tool and discuss how such a tool as well as suggestions can be efficiently implemented even for large product databases.

1. INTRODUCTION

People frequently use the world-wide web to search through a large collection of items. The most common search facility available on the web is based on a form that is directly mapped to a database query and returns a ranked list of the most suitable options. The user has the option to return to the initial page and change his preferences and then carry out a new search. This is the case for example when searching for flights on the most popular travel web sites[1][2]. Such tools are only as good as the query the user formulates. A study [6] has shown that among the users of such sites only 18% are satisfied with their final choice.

The goal of preference-based search should be to enable the user to make an *accurate* decision, meaning that the chosen item is actually the most preferred one. Accuracy is important in e-commerce for many reasons: it leads to an increase of user satisfaction and confidence and thus more repeated visits; customers would even be willing to pay more.

In most cases, users do not know exactly what they are looking for: they might consider different trade-offs or they might even have conflicting desires about the features the item should have. In fact psychological studies have shown that people construct their preferences [14] while learning about the available products. Therefore preference-based search should also help users in formulating accurate preferences.

We believe that the key issues for implementing successful preference-based search systems are:

1. *preference modeling:* the formalism chosen to model preferences
2. *preference elicitation:* how to acquire or learn preferences from the user
3. *usability:* ease of use of the interface
4. *scalability:* do the algorithms used by the tool scale up for large databases?

The first point, *preference modeling*, requires the designer to choose from possible preference representations. The user expresses the preferences using an interface by qualitative statements; these are then translated into the internal preference model.

Preference elicitation is crucial. Decision theory [11] provides a method that guarantees perfect decision accuracy by first eliciting a model of the user's preferences through a series of questions, and then determining the optimal choice based on this model. However, even for simple items such as cameras, eliciting such a model would require hundreds of questions, and few users would be ready to undergo such a lengthy process. A major challenge is that in an interaction on the WWW, few users are willing to tolerate more than 5-10 interaction cycles before reaching a result. Therefore, we need to provide users a concise way to express preferences that would be *usable*. At the same time we would benefit from approaches able to mitigate the inaccuracies inevitably created when translating a user's qualitative statement into a quantitative model of preferences suitable for ranking items.

First, we describe the *example-critiquing* approach to preference based search. We then discuss how to model preferences and to compute suggestions. Finally we will discuss

*(Produces the permission block, copyright information and page numbering). For use with ACM_PROC_ARTICLE-SP.CLS V2.6SP. Supported by ACM.

[1] `http://www.travelocity.com/`

[2] `http://www.expedia.com`

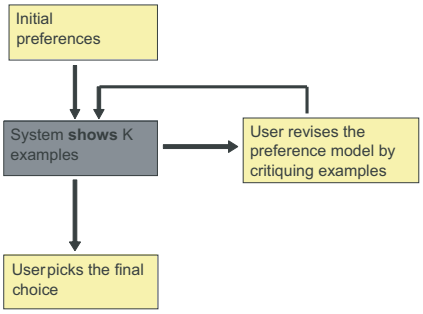

Figure 1: *Example-critiquing interaction. The dark box is the computer's action, the other boxes show actions of the user.*

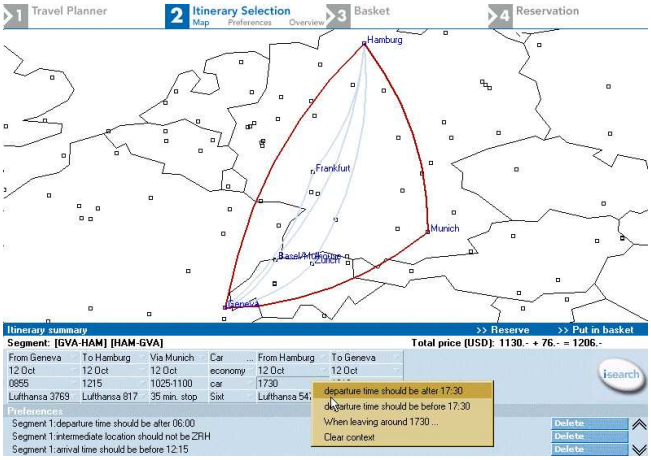

Figure 2: *Isy-travel is an example-critiquing tool for planning business-trips. Here the user is planning a one-day trip from Geneva to Hamburg. The preference model is shown at the bottom, and the user can state additional preferences by clicking on features of the shown example.*

how to implement scalable tools and present a prototype for student accommodation search called *FlatFinder*.

1.1 Example-critiquing

People are usually not able to state preferences up-front and behavioral decision theory studies [14] have shown that people *construct* their preferences as they see the available options. The elicitation through an interaction based on examples is therefore faster and often more precise than a comprehensive preference elicitation procedure; the latter may construct its model based on an incoherent set of answers!

Figure 2 shows Isy-travel, a commercial tool for business travelers [16]. Here, the user is shown examples of options that fit the current preference model well. The idea is that an example either is the most preferred one, or there is some aspect in which it can be improved. Thus, on any of the examples, any attribute can be selected as a basis for critiquing. For instance, if the arrival time is too late, then this can be critiqued. The critique then becomes an additional preference in the model.

This form of example critiquing has been proposed by various researchers, including the ATA system of Linden et al. [13], SmartClient [16], and more recently dynamic critiquing [20].

The advantage of such a system in the elicitation of preferences is that examples help users reason about their own preferences, revise them if are inconsistent, have an idea of which preferences can be satisfied and make trade-off decisions.

Example-critiquing has been shown to provide a means for effective preference elicitation. Pu and Li [15] have shown that example-critiquing with its tradeoff support enables consumers to more accurately find what they want (with up to 57% increase in accuracy) and be more confident in their choices, while requiring a level of cognitive effort that is comparable to simple interfaces such as a ranked list [18].

2. INCREMENTAL PREFERENCE MODEL ACQUISITION

In example-critiquing, a preference is stated as reaction to displayed options. A critiques can either be *negative reactions* to the options shown, when none of them satisfy the preference, or *positive reactions*, when an option satisfies the preference.

For instance, if the tool shows the user examples that all arrive at London Stansted airport, and she requests to land in Heathrow, that critique would be a *negative reaction*. If the system indeed showed one flight landing in Heathrow, by stating that preference she would be *positively* reacting to the shown examples.

If certain preferences are missing from the current model of the user, the system provides examples that do not satisfy those unknown preferences. If the user is aware of all of her preferences, she can realize the necessity to state them to the system by posting what we have called a *negative reaction* critique. However our intuition is that this is not always the case, because the user might not not know all the available options. Moreover, stating a preference costs some user effort (in our prototype, 2 selections and 2 clicks) and rationally she would do that only if she perceives this as beneficial.

To use a metaphor, the process of example-critiquing is hill-climbing: the user states preferences as long as he perceives it as bringing to a better solution. However, the process might end in a local optimum: a situation in which the user can no longer see potential improvement. For example, a user looking for a notebook computer might start looking for a low price, and thus find that all models weigh about 3 kg. Since all of the presented models have about the same weight, he or she might never bother to look for lighter models. This influence of current examples prevents the user from refocussing the search in another direction; this is known as the *anchoring effect* [23].

For these reasons we display two sets of examples:

- **candidate examples** that are optimal for the preference model, and

- **suggested examples** that are chosen to stimulate the expression of preferences.

We conducted user studies to evaluate the decision accuracy of example critiquing with and without suggestions. We found [19] that with the aid of suggestions, users state more preferences and, more importantly, achieve a much higher decision accuracy (up to 70%). Subsequently, we looked at the logs of the user study to check frequency of the different type of critiquing described before. In most of the cases (55%) a preference is stated as positive critiques.

2.1 Optimal candidate examples

Preference-based search looks for the option that best satisfies a preference model. The best options can be found by sorting the database items according to their cost. In the database community [7] this is known as the *top-k query*.

Note that because of the translation from qualitative statements into a quantitative preferences, the resulting cost function and the induced ranking are inaccurate. A common approach in many web search applications is to compensate by showing not just one, but a set of k options. Faltings et al. in [8] show that given a bound on the error of the representation, it is possible to compute a minimum k such that the user can find the truly best option among this set of k possibilities. Interestingly, while k grows with the error and the number of preferences, it is independent of the total number of options available, so that the approach scales even for searches in large collections.

2.2 Suggestions: diversity & lookahead principle

The importance of the diversity of the example shown was recognized by Linden, S. Hanks and N. Lesh ([13]) who explicitly generated examples that showed the extreme values of certain attributes, called *extreme examples*. However, an extreme example might often be an unreasonable choice: it could be a cheap flight that leaves in the early morning, a student accommodation where the student has to work for the family, an apartment extremely far from the city. Moreover, in problems with many attributes, there will too many extreme or diverse examples to choose from, while we have to limit the display of examples to few of them.

We assume that user is minimizing her own effort and will add preferences to the model only when she can expect them to have an impact on the solutions. This is the case when:

- she can see several options that differ in a possible preference, and
- these options are relevant, i.e. they could be reasonable choices, and
- these options are not already optimal, so a new preference is required to make them optimal.

In all other cases, stating an additional preference is likely to be irrelevant. When all options would lead to the same evaluation, or when the preference only has an effect on options that would not be eligible anyway, stating it would only be wasted effort. This leads us to the following *look-ahead* principle as a basis for suggestion strategies:

> Suggestions should not be optimal under the current preference model, but should provide a high likelihood of optimality when an additional preference is added.

We stress that this is a heuristic principle based on assumptions about human behavior that we cannot formally prove. However the high decision accuracy achieved in user studies [19] is an important motivation. Furthermore, examining the incremental critiques more in detail in the user studies, we found that in most cases there was a displayed example that became optimal because of the addition of a preference. This provides another clue of the validity of the principle.

In the following sections we present our model of treating preferences, the suggestion strategies, the implementation and the scalability issues.

3. THEORETICAL MODEL

3.1 Modeling items and preferences

We assume that items are modeled by a fixed set of m attributes that each take values in associated domains. Domains can be *enumerated*, consisting of a set of discrete elements, or *numeric*. In this paper, we consider preferences on individual attributes and independent of one another (i.e. we do not consider conditional preferences). A preference r is an order relation of the values of an attribute a.

For a practical preference-based search tool, it is convenient to express preferences in a concise way. We consider total orders (each pair is comparable) and express them by a numerical cost function c, $d_k \rightarrow \Re^+$, that maps a domain value d_k of an attribute a_k to a real number. A preference always applies to the same attribute a_k; we use the notation $c_i(o)$ to express the cost that the function assigns to the value of option o for that attribute.

Whenever o_1 is preferred to o_2 according to preference i, the first will have lower cost (for preference i) than the second: $c_i(o_1) < c_i(o_2)$.

An overall ranking of options can be obtained by combining the penalty functions for all stated preferences. Some researchers [9] have proposed the use of machine learning algorithms for finding the best aggregate function for a particular user. In our systems, we combine them using a weighted sum, which corresponds well to standard multi-attribute utility theory [11]. Thus, if $\mathcal{R}_c = \{c_1, .., c_s\}$ is the set of the cost functions of all preferences that the user has stated, we compute the cost $C(o) = \sum_{c_i \in \mathcal{R}_c} w_i \cdot c_i(o)$. Option o_1 is preferred over option o_2 whenever it has a lower cost, i.e. $C(o_1) < C(o_2)$.

The user states preferences in a qualitative way (for example "the price should be less than 500 dollar"). We map these qualitative statements into parameterized functions that are standardized to fit average users. These are chosen with respect to the application domain.

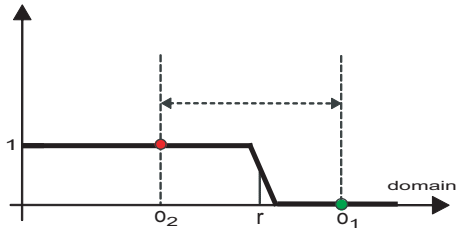

Figure 3: *For an ordered attribute, a new preference will prefer o_1 over o_2 if the reference value r falls between the values of the attribute, and the preference is of the right polarity.*

Similar models for modeling preferences in databases have been proposed in [12]. Preference modeling for example-critiquing is discussed in more detail in [17].

3.2 Model-based Suggestion Strategy

In [19] we proposed different strategies that use the concept of pareto-optimality to implement the look-ahead principle stated in the introduction: suggestions should not be optimal yet, but have a high likelihood of becoming optimal when an additional preference is added. We call them *model-based* suggestion strategies because they specifically choose examples to stimulate the expression of additional preferences based on the current preference *model*.

We model preferences by standardized functions that correctly reflect the preference order of individual attribute values but may be numerically inaccurate. When generating suggestions, we would like to use a model that is not sensitive to this numerical error. Pareto-optimality is the strongest concept that would be applicable regardless of the numerical details of the penalty functions.

An option o is **dominated by** another option $\bar{o}$ (equivalently we say that $\bar{o}$ dominates o) if

- $\bar{o}$ is not worse than o according to all preferences in the preference model: $\forall c_i \in R : c_i(\bar{o}) \leq c_i(o)$
- $\bar{o}$ is strictly better than o for at least one preference: $\exists c_j \in R : c_j(\bar{o}) < c_j(o)$

An option o is **pareto-optimal** if it is not dominated by any other option. The dominance relation is a partial order of the options that we will denote with the $\succ$ operator; Pareto-optimal options can also be seen as the set of maximal options with respect to the dominance relation.

In our applications, users initially state only a subset R of their true preference model $\overline{R}$. When a preference is added, dominated options with respect to R can become Pareto-optimal. The look-ahead principle can be formulated as follows: an ideal suggestion is an option that is Pareto-optimal with respect to the full preference model $\overline{R}$, but is dominated in R, the partial preference model.

The model based suggestions try to guess the chance that a (dominated) option has to become Pareto optimal. To become pareto-optimal when a new preference is added to the model, an option has to be strictly better than any dominating option with respect to this new preference.

We use a heuristic estimation of the probability that a hidden preference on attribute a_i makes o better than o^+ according to that preference, hence escaping the dominance relation. Such a heuristic considers the difference between the attribute values: the higher this difference, the more likely that new preference will make the option preferred. The reasoning is illustrated in Figure 3. The chances that a new preference will treat o_1 and o_2 differently depends on the difference between their values. Assuming that the shape of such a penalty function is a step function with sharp increase from 0 to 1, if the reference point falls at any point with equal probability, the chance of breaking the dominance is directly proportional to this difference.

4. ALGORITHMS AND SCALABILITY

In this section we analyze the algorithms to compute the candidates and suggestion examples and their complexity; we consider the problem of scalability and show some faster approximations.

4.1 Generation of Candidates

As said earlier, candidates are the best examples corresponding to the current set of preferences R. The generation of candidates can be addressed by a a *top-k query* to the database. The set of options retrieved $\{o_1, .., o_k\}$ is such that $C(o_1) \leq C(o_2) \leq .. \leq C(o_k)$ and for any other option $\bar{o}$ in the database $C(\bar{o}) \geq C(o_k)$.

While the trivial approach would compute the score of each option in the database, the *Fagin* algorithm [7] can do this with middleware complexity $O(N^{(m-1)/m}k^{1/m})$ where m is the number of attributes and k the number of candidates we want to generate. This algorithm can be applied to a different aggregate function as long as it satisfies some properties (monotonicity, strictness).

4.2 Generation of Suggestions

The model based strategy requires the analysis of the dominance binary relation, by making a series of pairwise checks between the options of the catalog. Each one evaluates two options, say o_1 and o_2, to determine whether o_1 dominates o_2, o_2 dominates o_1, they are equally preferred for all the preferences, or they are not comparable (i.e. there is no dominance in either direction). This is done by considering iteratively each of the preferences and comparing o_1 and o_2 for at most m comparisons (as soon as two preferences give opposite order of o_1 and o_2, they are not comparable), where m is the number of preferences, so the complexity is $O(m)$.

Since the dominance relation is a partial order, we can exploit asymmetry and transitivity to save some of the pairwise checks, however in the worst case we have to make $n(n-1)/2$ checks. So the complexity of the complete dominance analysis is $O(n^2m)$.

The algorithm for model-based suggestions is presented in the Algorithm 1. *Update* is responsible for updating the

value for the estimation of the probability $p(o, a_i)$ of becoming Pareto optimal given that the missing preference is on a_i; its precise definition depends on the particular assumptions on the possible preferences.

In the *probabilistic* strategy, the update multiplies the current value by $w_{a_i} * \delta(o, o_d)$, where $\delta(o, o_d)$ is the heuristic estimation presented in the previous section (based on the normalized distance between the attribute values) and w_{a_i} is a weight representing the probability that there is a preference on that attribute. Intuitively the more dominating options there are, the more $p(o, a_i)$ decreases. For more details, refer to Viappiani et. al. [24].

In another model-based strategy, the *attribute* strategy, we assume that the preferences that the users can state are only of the kind `LessThan` or `GreaterThan` (the user cannot express preferences for a value in the middle), therefore we check whether the current option has a value that is either smaller or bigger than any value of the dominating options: only in this case can a preference break all the dominance relations simultaneously. In this strategy, `update` will take the minimum of the absolute values of the $\bar{\delta}$, and returns 0 if they are of different sign.

Algorithm 1: **Model-based suggestions(int n)**
δ heuristics based on the normalized differences

```
for all option ∈ OPTIONS do
    p(option) = 0
    for all a_i ∈ A_u = {attributes with no preferences} do
        p(option, a_i) = 1 //contribution for attribute a_i
        for all o_d ∈ O_D = {o_d ∈ O : o_d ≻ option} do
            //we iterate over the set of options that
            dominates o
            δ̄ ← δ_{a_i}(o, o_d)
            p(option, a_i) = update(p(option, a_i), δ̄)
    p(option) = 1 − ∏_{a_i} (1 − p(option, a_i))
suggList ← order options according p
return first n options in suggList;
```

Our model based strategies have complexity $O(nmd)$, where m is the number of attributes and d is the number of dominating options. It is difficult to calculate an average value for d in function of n, because it depends on the data and correlation between attribute values. While generally the set of dominating options is much smaller than the set of options, in the worst case they can be a linear fraction of n. Sorting the options according to the resulting probability (to select the best n) costs $nlogn$ in term of complexity.

The overall complexity is $O(n^2)$: while this complexity was not a problem for our prototype, we expect it to be more problematic as the item collections grow. Approximations will be necessary for large databases.

We propose the following approximations:

- to select suggestions from the top k options
- to replace pareto-dominance with utility dominance
- to assume dominating options as a fixed number of options at the top

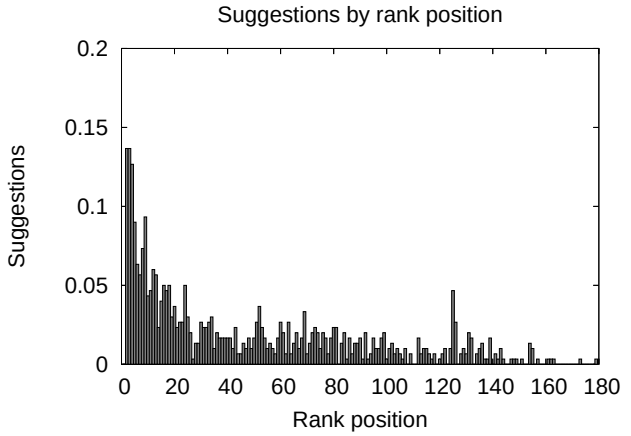

Figure 4: *The position of suggestions in the overall ranking; repeated queries are performed on a database of student accommodations with 187 options, 3 model-based suggestions are retrieved. On average,* 50% *of the suggestions are in the top* 16% *of the overall ranking,* 80% *in the top* 47%*.*

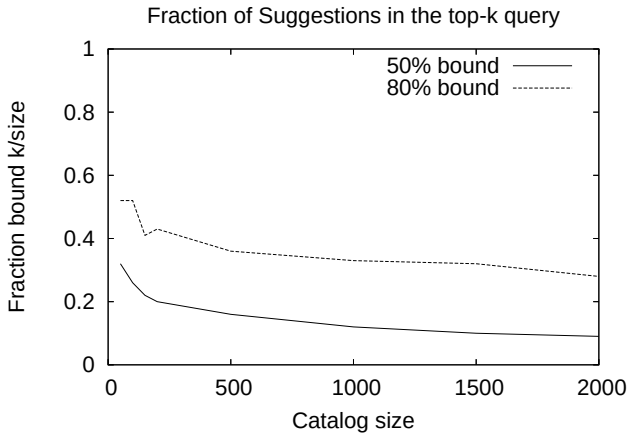

Figure 5: *For different size of catalogs, the bound k required to guarantee that respectively* 50% *and* 80% *of the suggestions are in the top k positions.*

4.2.1 Approximation 1: select suggestions from the top

From experimental tests, it emerged that suggestions are not evenly distributed according to their current costs, but they are often at the top. This is not surprising: since suggestions are options that should be reasonable, we will expect them to be not too far in the ranking from the current candidates.

In the actual use of the preference based search tool on the apartment database used by `FlatFinder` (the prototype presented in Section 5.1), in more than the 80% of the cases suggestions are among the top half of the options (ordered according to the optimal query) and in more than the 60% in the top quarter of the options.

We considered how to mitigate the impact of very large data sets by only considering suggestions in the subset of the cat-

alog that contains the *top-k* options for the preference query. Since an option can be dominated only by options that have lower cost $C()$, the preliminary phase of dominance analysis is also simplified. We can run the algorithms presented before on a database containing only the options returned by the *top-k* query with a given bound k_b. Using this approximation, the computation is going to do $O(k_b^2)$ checks.

We ran several generations of suggestions with our database of student accommodation and random databases of different size. We look for the *bound* k_b for which a certain fraction of suggestions lie in the top k_b positions in the ranking. Choosing the bound k_b in this way, we can guarantee a limited decrease in the quality of suggestions that does not depend on the size of the catalog.

Figure 5 shows that the bound grows more slowly as the catalog size increases. Using interpolation we found that the bound for ensuring 50% of suggestions approximately increases as the function $2.1n^{lg(1.5)}$. Substituting this function into k_b^2, we found that the complexity of the computation of suggestions is $O(n^{1.2})$, that is significantly lower than n^2. Empirically we found that these bounds for random data are higher than with real data, with correlation between the attributes values. Figure 4 shows the rank position of the suggestions retrieved on a database of student accommodations: 50% of the suggestions are in the top 16%, 80% in the top 47%.

This method results in a significant improvement in complexity. However, such complexity might be still costly for some applications.

4.2.2 Approximation 2: utility dominance

Instead of pareto dominance, we can use other forms of dominance. In particular, we might use the total ordering established by the combination function defined in the preference modeling formalism, such as a weighted sum $C()$. We call this *utility-domination*, and the utility-optimal option is the most preferred one.

An option can become utility-optimal only if it is strictly better than all options that currently utility-dominate it, although this is not a sufficient condition. The utility dominance approximation consists of checking the probability of breaking utility dominance. The advantage is that the dominating set is easily computed by simply checking the cost: once we have the ranking for the current preferences, the utility-dominating set will be composed by all the options prior in the ranking.

However, we still have to make, for each option, a comparison to its utility-dominators. These are 1 for the first option in the rank, 2 for the second, and n for the last. So, even if the calculation of dominators is faster, this strategy is still quadratic $O(n^2)$. We have a total complexity of $O(n^2)$, of the same order as the complete method, even if faster in practice.

4.2.3 Approximation 3: top-domination

This approximation strategy looks at the probability of breaking the dominance with the options at the top of the ranking. This method requires testing each option against a fixed number k_d of best options to check if their dominance could be broken. We don't have to look for the dominating set: these are always the options at the top in the ranking. For each option we need to make a constant number of comparisons to these top options, on m attributes. Therefore this method is much faster with a complexity $O(nm)$.

The difference with approximation 1 is that there we tested only a subset of the options against their actual dominating set, while here we test all the options against a constant set of dominators.

4.3 Evaluation of the Approximations

We evaluated the approximated techniques on the ability to find Pareto-optimal options considering the unknown preferences. We ran simulations with the apartment database and with randomly generated data. We generated a random model consisting of 5 preferences and we calculated the number of times the method successfully fulfilled our lookahead strategy (the *hit rate*) when some the preferences are not stated yet (i.e. the frequency of finding, among the suggestions selected, an option that became Pareto-optimal by the addition of one among the missing preferences).

method	note	hit rate
model-based suggestions		80.8%
approximation 1	$k_b = 50\%$	74.5%
approximation 2	utilitarian	73.2%
approximation 3	k_d=3	25%
approximation 3	k_d=6	70.1%
no suggestions	more candidates	33%

Table 1: The hit-rate according to our look-ahead principle for the different approximation strategies and the complete method of generation of suggestions. For comparison, we show the case in which we simply display more candidates.

The first approximation gives almost the same hit rate as the complete search, so it could often be a reasonable choice. In cases in which a very low complexity is required, the third approximation with $k_d = 6$ can be considered: it is much faster and still achieves a significantly greater hit rate that just showing more options from the top. Utility-dominance does not provide a good balance between computation benefit and losses and would rarely be the method of choice.

5. PROTOTYPE IMPLEMENTATION

Example-critiquing provides specific support for preference construction through suggestions and support for tradeoffs. These require separate ranking mechanisms that are not easily implementable on top of a conventional database system.

We therefore use a separate preference-based search layer, as shown in Figure 6. Through a *Java* `API`, it can be used for preference-based search on any collection of items defined in terms of attributes, domains and the possible penalty functions for expressing preferences. The preferences are expressed using a web interface that allows only qualitative statements and then translated to the internal representation. The platform allows the designer to choose from a set of parameterized functions that implement the possible preferences.

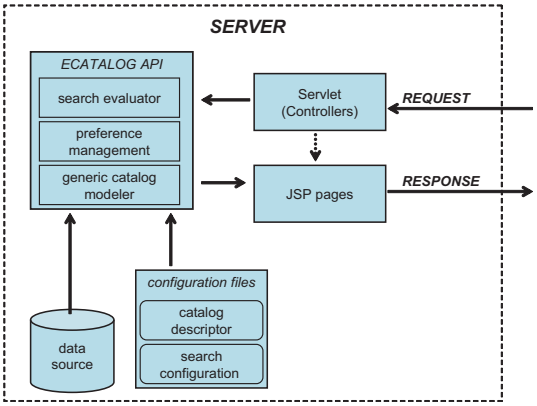

Figure 6: *The architecture of* `ECATALOG`*, the generic preference based search tool.*

We separated the design of visualization and user interaction from the preference-based search engine. Our architecture utilizes the *Model-View-Controller* (MVC) design pattern, implemented with the `Java Server Pages` (JSP model 2). It allows to separate the application logic (the decision problem model defined in our API) from the View (the JSP pages). This makes the framework easy to be extended and simplifies the programming by focusing at one aspect at time.

5.1 FlatFinder

`FlatFinder` is a web application for finding student housing that uses example-critiquing interaction. It uses actual offers of student housing from an university database that is updated daily, and usually contains about 200 options.

Each option is modeled by 10 attributes: type of accommodation (room in a family house, room in a shared apartment, studio apartment, apartment), rent, number of rooms, furnished (or not), type of bathroom (private or shared), type of kitchen (shared, private), transportation available (none, bus, subway, commuter train), distance to the university and distance to the town center.

Preferences are combined using a weighted sum. For numerical attributes, the tool allows preferences consisting of a relational operator (*less than*, *equal*, *greater than*), a threshold value and an importance weight between 1-5; for example, "distance to university less than 10 minutes" with importance 4. The penalty function `LessThan(x,k)` for numerical domains takes the form of a ramp starting at k: the penalty is 0 if x is less than k, and grows linearly after this point. The result is multiplied by the importance.

For the price attribute, the tool uses a slightly different penalty function consisting of a ramp with small slope below and high slope above a threshold value. For discrete attributes, a preference specifies that a certain value is preferred with a certain importance value (the other values are supposed to be equally disliked with the same violation degree). The system maintains the current set of preferences, and the user can state additional preferences, change the reference value of existing preferences, or remove one or more of the preferences at every interaction cycle.

The interaction starts with an empty set of preferences; the user is presented with an introductory page and has to choose one attribute on which to state the first preference. Then the dialogue proceeds with the standard page of the interface. It displays three panels: the *Preference panel* where the user sets and revises his preferences, and can change their importance. Once the changes have been made, the `Search` button updates the `Result panel`, where the results are shown: the best options (candidates) according to the user's preferences and suggestions. The user can store options in a *basket* for later comparison; at the end of the interaction, the user will chooses one among the options saved in the basket and checkout.

6. RELATED WORKS

Modeling and reasoning with preferences has been studied by a number of authors [12, 5]. Database researchers [1] have studied query systems that evaluate predicates with a continuous degree of validity and allow partial matches, as in fuzzy sql (FSQL) [3]. Top-k queries are queries that find records with highest score [7]. Preferences received lately attention by the AI community as well; algorithms have been proposed to find the best configuration based on qualitative representation of preferences in configuration [10] or as partial orders [2].

Example critiquing was first proposed in [22] and has since been used in several recommender systems, such as FindMe [4], ATA [13], SmartClient [16], ExpertClerk [21]. The ATA system [13] is particularly important, as it was the first to incorporate the notion of suggestions, which is crucial to our work. In *dynamic critiquing* [20], a popular family of example critiquing interfaces, a metaphor of navigation through the product space is implemented; the interface proposes pre-computed critiques (simple and compound) that can be selected by the users.

7. CONCLUSIONS

Preference-based search is a ubiquitous problem on the web. We have presented tools based on examples that can achieve higher decision accuracy than the traditional form filling approach. User studies show that such tools can greatly help the user, especially when they integrate the display of suggestions that make the users aware of possible choices and stimulate the preferences expression. The principle we follow is that good suggestions are items that become optimal when other possible preferences are considered. Our suggestion strategies are based on the concept of Pareto optimality.

However few web sites currently support preference based search. The main issue is that such personalized search tools are hard to implement for large databases and user populations. To overcome this problem, we presented three approximations that simplify the computation, making the generation of suggestions scale.

The first approximation considers possible suggestions only in the top options returned by a *top-k query*. It maintained a hit-rate (74.5%) close to the optimum with a significant reduction of the number of pairwise checks required in the computation. The second considers utility dominance instead of the standard dominance relation, simplifying the

computation of dominating options. However, it as lower hit-rate (73%) and a higher complexity.

Finally, the third considers as dominating options the ones with lowest cost. It can be a reasonable choice for large databases as it achieves a very fast computation of suggestions with a complexity of $O(nm)$ and provides a hit rate of 70.1%

8. REFERENCES

[1] R. Agrawal and E. L. Wimmers. A framework for expressing and combining preferences. In *SIGMOD Conference 2000*, pages 207–306, 2000.

[2] C. Boutilier, R. I. Brafman, C. Domshlak, H. Hoos, and D. Poole. Preference-based constrained optimization with CP-nets. *Computational Intelligence, Special Issue on Preferences in AI and CP, to appear*, 2005.

[3] B. P. Buckles and F. E. Petry. Fuzzy databases in the new era. In *SAC*, pages 497–502, 1995.

[4] R. D. Burke, K. J. Hammond, and B. C. Young. The FindMe approach to assisted browsing. *IEEE Expert*, 12(4):32–40, 1997.

[5] C. Domshlak, F. Rossi, K. B. Venable, and T. Walsh. Reasoning about soft constraints and conditional preferences: complexity results and approximation techniques. In *Proc. IJCAI 2003*, Acapulco, Mexico, August 2003.

[6] M. S. D. W. Equity. Transportation e-commerce and the task of fulfilment, 2000.

[7] R. Fagin. Fuzzy queries in multimedia database systems. In *PODS '98: Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 1–10, New York, NY, USA, 1998. ACM Press.

[8] B. Faltings, M. Torrens, and P. Pu. Solution generation with qualitative models of preferences. In *Computational Intelligence*, pages 246–263(18). ACM, 2004.

[9] S.-w. H. Hwanjo Yu and K. C.-C. Chang. Rankfp: A framework for supporting rank formulation and processing. In *ICDE 2005*, pages 514–515, 2005.

[10] U. Junker. Preference-based search and multi-criteria optimization. In *Eighteenth national conference on Artificial intelligence*, pages 34–40. American Association for Artificial Intelligence, 2002.

[11] R. L. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Tradeoffs.* John Wiley and Sons, New York, 1976.

[12] W. Kiesling. Foundations of preferences in database systems. In *VLDB 2002*, pages 311–322, 2002.

[13] G. Linden, S. Hanks, and N. Lesh. Interactive assessment of user preference models: The automated travel assistant. In *Proceedings, User Modeling '97*, 1997.

[14] J. Payne, J. Bettman, and E. Johnson. *The Adaptive Decision Maker*. Cambridge University Press, 1993.

[15] P. Pu and L. Chen. Integrating tradeoff support in product search tools for e-commerce sites. In J. Riedl, M. J. Kearns, and M. K. Reiter, editors, *ACM Conference on Electronic Commerce*, pages 269–278. ACM, 2005.

[16] P. Pu and B. Faltings. Enriching buyers' experiences: the smartclient approach. In *SIGCHI conference on Human factors in computing systems*, pages 289–296. ACM Press New York, NY, USA, 2000.

[17] P. Pu and B. Faltings. Decision tradeoff using example-critiquing and constraint programming. *Constraints: An International Journal*, 9(4), 2004.

[18] P. Pu and P. Kumar. Evaluating example-based search tools. In *ACM Conference on Electronic Commerce (EC'04)*, 2004.

[19] P. Pu, P. Viappiani, and B. Faltings. Increasing user decision accuracy using suggestions. In *CHI*, page to appear, April 2006.

[20] J. Reilly, K. McCarthy, L. McGinty, and B. Smyth. Dynamic critiquing. In P. Funk and P. A. González-Calero, editors, *ECCBR*, volume 3155 of *Lecture Notes in Computer Science*, pages 763–777. Springer, 2004.

[21] H. Shimazu. Expertclerk: Navigating shoppers buying process with the combination of asking and proposing. In *Proceedings of the 17 International Joint Conference on Artificial Intelligence (IJCAI'01)*, volume 2, pages 1443–1448, 2001.

[22] F. N. Tou, M. D. Williams, R. Fikes, D. A. H. Jr., and T. W. Malone. Rabbit: An intelligent database assistant. In *AAAI*, pages 314–318, 1982.

[23] A. Tversky. Judgement under uncertainity: Heuristics and biases, 1974.

[24] P. Viappiani, B. Faltings, V. Schickel-Zuber, and P. Pu. Stimulating preference expression using suggestions. In *Mixed-Initiative Problem-Solving Assistants*, volume FSS07-05 of *AAAI Fall Symposium Serie*, pages 128–133. AAAI, 2005.