Inferring User Utility for Query Revision Recommendation

Henry Blanco
Faculty of Computer Science
Free University of Bolzano
Bolzano, Italy
Center of Medical Biophysics
University of Oriente
Santiago de Cuba, Cuba

Francesco Ricci
Faculty of Computer Science
Free University of Bolzano
Bolzano, Italy
fricci@unibz.it

ABSTRACT

A recommender system (RS) can infer constraints on the user utility function by observing the queries selected by a user among those it has suggested. Reasoning on these constraints it can avoid suggesting queries that retrieve products with an inferior utility, i.e., dominated queries. In this paper we propose a new efficient technique for the computation of dominated queries. It relies on the system's assumption that the number of possible profiles (utility functions) of the users it may interact with is finite. Under this assumption query suggestions can be efficiently computed and their number can be kept small. Moreover, we show that even if the system is not contemplating all the possible user profiles its performance is very close to the optimal one.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Query formulation

General Terms

Algorithms, Experimentation, Design

Keywords

Conversational recommender system, preference elicitation

1. INTRODUCTION

Conversational recommender systems offer flexible support to users as they browse a product catalogue, and help them to better understand and elicit their preferences. Instead of requiring users to specify their preferences at the outset, these are acquired and revised over a series of interaction steps. At each step the system makes some recommendations to the user, or invites her to indicate further preferences, e.g., by critiquing recommendations [8].

In [4, 11] it is introduced and evaluated a new conversational technique for helping the users to select items of largest utility to them. Applying that technique, in response to a user query to a product catalogue, the system suggests new queries that: a) extend the user current query, and b) retrieve products that the system deems to have the largest utility for the user. For example, if the user has submitted the following query: "I want an hotel with parking". The system, rather than retrieving immediately the products that satisfy this query, hypothesizes that the user may have other needs not yet expressed and therefore it suggests some query revisions. For instance, the system may say: "are you interested also in sauna?". Products with more features, if available, will surely not decrease the user utility. But not all features are equally important for the user. So, the system's goal is to make "informed" suggestions, i.e., to suggest queries selecting products with the features that are more likely to produce the largest increase to the user utility.

The major limitations of the above mentioned approach were stated to be: a) a limited number of query editing operators, i.e., the system could suggest only two types of new queries to the user (add a feature and trade one feature for two), b) a computationally expensive method for computing the next best queries (undominated queries), c) a potentially long list of query suggestions, making it hard for the user to evaluate them and select the preferred one, d) a product description based only on Boolean features.

In this paper we address those limitations by proposing a new technique for the computation of dominated queries, i.e., queries that should not be suggested because the system can deduce that they have a lower utility than others. The proposed technique relies on the system assumption that the set of profiles (utility functions) of the users it may interact with is finite. This is a meaningful assumption as not all the possible profiles are likely to be observed in practice, and users tend to have similar profiles. We show that the computation of the query suggestions is simplified, and more importantly, the number of queries that are suggested at each interaction step is greatly reduced.

Moreover, the proposed approach has also another advantage, it enables a system designer to freely select the types of query editing operations for generating new query suggestions to the user. We also show that even if the system, when interacting with a user, is not contemplating her true user profile, among the above mentioned finite set of profiles, its performance is still very close to the optimal one, i.e., at the end of the interaction the system suggests a query that selects the best products, given the true user profile and the available products in the data set. In fact, we show that progressively expanding the number of profiles contemplated by

SAC'13 March 18-22, 2013, Coimbra, Portugal.

the system one can increase the utility of the final recommended products and with a large number of contemplated profiles the recommended products have a utility that is not practically distinguishable from that of the best products.

Finally, since cost/price issues typically play an important role in decision making processes we have extended the model to consider a discrete numerical feature, such as the cost of an item, and we have analyzed the impact of this feature on the proposed query recommendation technique.

The rest of the paper is structured as follows. The query language used in this approach is described in Section 2. Section 3 describes our model for representing the user utility. Section 4 explains the concept of "dominated query" and our query suggestion method. The experimental design is shown in Section 5. Results and discussion are reported in Section 6. Finally the related work and conclusions are given in Sections 7 and 8 respectively.

2. QUERY LANGUAGE

We consider products that can be described by their features, e.g., the cost of a room in a hotel or the presence of air conditioning (AC). Then we model a product by an n-dimensional Boolean vector $p = (p_1, \ldots, p_n)$ with n attributes. Boolean features of the product, such as AC, are modeled as Boolean attributes, where $p_i = 1$ ($p_i = 0$) means that the product has (not) the i-th feature. Using Boolean attributes we also model discrete features. In this paper to simplify the notation we consider products with only one discrete feature, which represents the cost of the product, but our approach can be used also when more discrete features are needed. A discrete valued feature is modeled by as many Boolean attributes as its possible values. For example, if we assume that the cost of a product is discrete, and takes the values in $\{v_1, \ldots, v_m\}$, then we model it with m mutually exclusive Boolean attributes. Hence the final model of a product with $f + 1$ features, f Boolean and one discrete, is a Boolean vector of dimensionality $f + m$, $p = (p_1, \ldots, p_f, p_{f+1}, \ldots, p_{f+m})$. Here $p_{f+i} = 1$ means that the product's cost is v_i. A catalogue is a set of products $\{p^{(1)}, \ldots, p^{(k)}\}$ and the considered Boolean features could be keywords or tags used in the product description, and searching for products with these features can be viewed as kind of facet search.

Queries are represented as Boolean vectors as well: $q = (q_1, \ldots, q_{f+m})$. $q_i = 1$ means that the user is interested in products that have the i-th attribute, i.e., they have a Boolean feature or they have a particular value for a discrete one. On the other hand $q_i = 0$ does not mean that the user is not interested in products with that attribute, but simply that she has not yet declared her interest on it. Since the last m entries of this query vector refer to a single discrete feature, only one of them could be set simultaneously to 1. A query is said to be *satisfiable* if there exists a product in the catalogue such that all the attributes expressed in the query as desired ($q_i = 1$) are present in that product. For example, if the products have 2 Boolean features and one discrete feature with three possible values (v_1, v_2, v_3) the vector $p = (1, 1, 0, 1, 0)$ represents a product that has the two Boolean features and the discrete one has value v_2. Moreover, if this product is present in the catalog then the query $q = (0, 1, 0, 1, 0)$ is satisfiable.

We are considering a scenario where the user is advised about how to revise her queries. Moreover, we assume that the system GUI offers to the user a well selected set of query revisions that are generated by the system using simple editing operators (as in critiquing-based approaches). In the following we list the query editing operators that the system uses. However, we observe (as it will be clear in the ensuing description) that the proposed approach is not constrained by this particular choice.

- $add_1(q, i)$, $i \in idx0(q)$;
- $trade_{1,2}(q, i, j, k)$, $i \in idx1(q)$ and $j, k \in idx0(q)$;
- $add_2(q, i, j)$, $i, j \in idx0(q)$;
- $trade_{1,3}(q, i, j, k, t)$, $i \in idx1(q)$ and $j, k, t \in idx0(q)$.

Here $idx0(q)$ and $idx1(q)$ are the sets of indexes with value 0 and 1 in q respectively. The first two operators revise the current query by requesting one additional attribute. For example, still assuming that products have 2 Boolean features and one discrete feature with three possible values, $(1, 1, 0, 0, 1) = add_1((1, 1, 0, 0, 0), 5)$ is the query that extends $(1, 1, 0, 0, 0)$, where only the first two Boolean features are requested, to a new query where also the third discrete feature is requested, and imposes that the value for this is v_3 (which corresponds to the fifth attribute). The second operator (trade one attribute present for two not present) generates a new query by discarding an attribute, the i-th, in favor of two new ones, the j-th and k-th. For example, $(0, 1, 0, 1, 0) = trade_{1,2}((1, 0, 0, 0, 0), 1, 2, 4)$ is discarding the first Boolean feature in favor of the second Boolean feature and the third discrete feature, imposing for it the value v_2.

The last two operators extend a query with two attributes. For example, given the query $q = (1, 0, 0, 0, 0)$, the second (Boolean) and third (discrete) feature are requested in the new query generated by the operator $add_2(q, 2, 5) = (1, 1, 0, 0, 1)$. The second "trade" operator (trade one attribute present for three not present) generates a new query by discarding the i-th attribute in favor of three new ones.

Using the above-mentioned operators the system can iteratively generate a set of next queries, and ask the user to select the preferred one (see Section 6.1 for an example of such an interaction). However, the goal of the proposed system is not to suggest all these possible next query revisions, as a standard "query by example" interface might do, but only those that would retrieve products with the largest utility for the user. Hence, the goal of the proposed system is to make inferences on the true user utility function, and not to suggest those queries that appear to the system to have an inferior utility. This reasoning process is described in the following sections.

3. USER UTILITY FUNCTION

A user utility function, also called user profile, is defined by a vector of weights $w = (w_1, \ldots, w_{f+1})$, $0 \leq w_i \leq 1$. w_i models the importance that the user assigns to the i-th feature of a product. $w_i = 0$ means that the user has no desire for the i-th feature. If $w_i > w_j$ then the i-th feature is more important than the j-th one. If $w_i \geq w_j$ then the i-th feature is at least as important as the j-th one. If $w_i = w_j$ then the user is indifferent between these two features. Here, as described in the previous section, we assume that the first f weights are relative to Boolean features of the products (e.g., AC is either true or false), while the $f + 1$-th weight is relative to a discrete product feature (in our examples it is the cost) that is mapped to m Boolean attributes. The

user utility of product $p = (p_1, \ldots, p_{f+m})$ is given by the following formula:

$$U_w(p) = \sum_{i=1}^{f} w_i \times p_i + w_{f+1} \times \frac{v_m - v_t}{v_m - v_1} \quad (1)$$

The utility is increased by w_i if the Boolean feature $i = 1, \ldots, f$ is present in the product, and is increased by the last term in Equation 1 according to the product cost: this term increases linearly as the cost decreases [5]. Here v_m and v_1 are the largest and smallest possible costs, while v_t is the actual cost of the product. In order to deal with another type of discrete feature a specific scoring function assigning the utility of each possible value is needed, but this would not change the general model here presented (see [5] for other examples). Finally, the utility of a query q for a user with profile w, $U_w(q)$, can be defined as the utility $U_w(p)$ of a product p with the same definition as q, i.e., $q = p$.

We assume that the user is rational and prefers products with larger utility. Moreover, we assume that a user may have any one of the possible utility functions that can be defined by varying the feature weights w_i, $i = 1, \ldots, f + 1$. So, the full set of all possible utility functions is in principle infinite. But observing the queries selected by the user among those suggested the system can infer constraints on the definition of the true user utility function. Generally speaking, the features present in the selected query should be considered more desirable those present in the alternative queries that the user could have tried but did not select.

More precisely, let us assume that the system suggests to the user a set of new query revisions, which we call the *AdviceSet*. The queries in the *AdviceSet* are a subset of the queries that can be generated by applying the query editing operators (see previous section) to the query that was selected by the user at the previous interaction step. When the user selects one out of these recommended queries, $q_s \in$ *AdviceSet*, as the new best query, the system can deduce that the utility of q_s is greater than or equal to the utility of the other queries that were present in the *AdviceSet*:

$$U_w(q_s) \geq U_w(q), \ \forall q \in AdviceSet. \quad (2)$$

Hence, at each interaction step the system can acquires a set of inferred inequalities on the values of the importance weights w_i, $i = 1, \ldots, f + 1$ of the true user utility function.

Example 1. If products have 4 Boolean features and one discrete feature with two alternative values representing the product's cost (v_1 and v_2). Let us assume that the last query selected by the user is $q = (1, 0, 0, 0, 1, 0)$. To make the description simpler let us describe a query by listing the indexes of the attributes that are set to 1, i.e., $q = \{1, 5\}$. This query requests products having the first Boolean feature and the cost equal to v_1. Imagine that the system suggests the following four query revisions *AdviceSet* = {{1,2,5}, {3,4,5}, {1,3,6}}, and the user selects the query {3,4,5}. Then the constraints inferred by the system are: $w_3 + w_4 \geq w_1 + w_2$, $w_4 + w_5 \frac{v_2 - v_1}{v_2 - v_1} \geq w_1 + w_5 \frac{v_2 - v_2}{v_2 - v_1}$, which is $w_4 + w_5 \geq w_1$.

Constraints on the true user profile w can also be deduced when the user issues the very first query of the interaction. In this case, the advisor will infer that all the features that are present in the query have greater or equal importance than those that are not present. In the following section we will describe exactly how the system generates the *AdviceSet*.

4. THE QUERY ADVISOR

The advisor is the recommender system component in charge of suggesting to the user how to revise the current query in order to select products with larger utility; the *AdviceSet* is composed by these query revisions. We stress that the true user profile is not known by the advisor and the advisor does not explicitly ask the user for the importance of the features (the weights w_i).

At each interaction step the advisor accumulates constraints on the true user utility function (as described in Section 3). We denote this set of constraints by Φ. Moreover, given a set of next possible queries $C = \{q^{(1)}, \ldots, q^{(k)}\}$, i.e., those that can be generated by applying the operators described in Section 2, and that are satisfiable, the advisor will not suggest queries that (in its opinion) have a lower utility than another one: these queries are called here "dominated".

A query $q \in C$ is considered by the system as *dominated* if there exists another query $q' \in C$ such that for all the possible weight vectors w that are compatible with the set of constraints Φ we have $U_w(q') > U_w(q)$. A weight vector w is said to be *compatible* with the set of constraints in Φ if and only if all the constraints in Φ are satisfied when the variables $w_1, \ldots, w_{f+1}$ take the values specified in w.

Removing the dominated queries is meaningful because their utility is lower than the utility of another candidate query for all the possible user utility functions that are compatible with the preferences that have been inferred by observing the user behavior.

In [4], the problem of finding dominated queries was cast in a set of linear programming problems, allowing an infinite number of user profiles to be considered. In this paper we assume that the set of user profiles contemplated by the system is finite $P = \{w^{(1)}, \ldots, w^{(M)}\}$. We will consider in the experiments sets of user profiles ranging from some dozens to thousands.

With this finite assumption, at each interaction step, using the set of constraints Φ we can prune from P the "incompatible profiles", i.e., those not satisfying the constraints in Φ. The computation of the *AdviceSet* proceeds as follow. Let us assume that the set of user profiles compatible with the accumulated constraints is $P' = \{w^{(1)}, \ldots, w^{(t)}\} \subset P$ and $C = \{q^{(1)}, \ldots, q^{(k)}\}$ is the set of next possible queries, i.e., the queries that are satisfiable and are generated from the last issued query of the user by applying the query editing operators, then:

1. A query $q \in C$ is labelled as dominated if and only if there exists another query $q' \in C$, $q' \neq q$, such that $\forall w \in P'$, $U_w(q') > U_w(q)$.

2. The *AdviceSet* (undominated queries) is built by removing from C the dominated queries.

Example 2. Let us assume that products have 2 Boolean features and one discrete feature with three possible values, which is modeled by three additional Boolean attributes (3, 4 and 5), indicating the product has cost either 11, or 21 or 30 respectively. Let us further assume that $P' = \{w^{(1)}, w^{(2)}, w^{(3)}\}$ is the set of profiles compatible with the acquired constraints Φ and: $w^{(1)} = (0.15, 0.25, 0.6), w^{(2)} = (0.25, 0.15, 0.6), w^{(3)} = (0.6, 0.15, 0.25)$. Let us imagine that observing the current user's query selection the system has acquired some new constraints and $\Phi = \{w_3 \geq w_1, w_3 \geq$

Table 1: Query utilities for profiles $w^{(1)}$ and $w^{(2)}$

	$\mathbf{q^{(1)}}$	$\mathbf{q^{(2)}}$	$\mathbf{q^{(3)}}$
$\mathbf{w^{(1)}}$	1	0.68	0.4
$\mathbf{w^{(2)}}$	1	0.68	0.4

$w_1 + w_2\}$. Now $w^{(1)}$ and $w^{(2)}$ still satisfy the constraints in Φ, but $w^{(3)}$ is an "incompatible profile" since $w_3^{(3)} < w_1^{(3)}$ and therefore it must be removed from P'. Let us further assume that the candidate queries are $C = \{q^{(1)}, q^{(2)}, q^{(3)}\}$, $q^{(1)} = \{1, 2, 3\}, q^{(2)} = \{1, 2, 4\}, q^{(3)} = \{1, 2, 5\}$. Table 1 shows their utilities for the two compatible profiles. With $w^{(1)}$ the utilities of $q^{(1)}$, $q^{(2)}$ and $q^{(3)}$ are $0.15 + 0.25 + 0.6 \times \frac{30-11}{30-11}$, $0.15 + 0.25 + 0.6 \times \frac{30-21}{30-11}$ and $0.15 + 0.25 + 0.6 \times \frac{30-30}{30-11}$ respectively.

Since $q^{(1)}$ has a higher utility than $q^{(2)}$ and $q^{(3)}$, for every compatible profile in $P' = \{w^{(1)}, w^{(2)}\}$, then $q^{(2)}$ and $q^{(3)}$ are dominated by $q^{(1)}$. Dominated queries must be removed from the set C and the *AdviceSet* must only contain $q^{(1)}$.

The full algorithm for query suggestions is described in Figure 1. At the first interaction step there are no query suggestions and the user is free to enter the first query q_s (step 4). The user selects the query with the t most important features, e.g., she may initially specify only two features. Then, the advisor infers the constraints to be added to Φ according to the rules mentioned in Section 3. The advisor then removes the user profiles that do not satisfy these constraints. At the next step the set of candidate queries is generated from the current query by applying the operators mentioned in Section 2 and discarding any query that is not satisfiable. Next, the advisor builds the *AdviceSet* by removing the dominated queries (step 8), and optionally filters the *AdviceSet* to reduce its potentially long size (step 9). Filtering is performed by discarding some more queries, and the filtering strategy that we have considered will be presented in the next section. Finally, the advisor recommends the remaining queries to the user as potential next moves.

If the *AdviceSet* is not empty, the user is supposed to select the query revision from this set with the largest utility, according to her true utility function, and the selected query becomes the current query. At this point the process is repeated (new interaction step) and new query revisions are suggested to the user. If the user does not want any further advice the system displays the products that satisfy the last query selected by the user.

5. EXPERIMENTS DESIGN

We have performed several experiments by simulating interactions between virtual users and the advisor according to the algorithm described in the previous section. In each simulated interaction the virtual user is represented with a randomly generated vector of weights (the true user profile) which is not revealed to the system. Moreover, the system contemplates a predefined set of randomly generated user profiles P that are the profiles of the users the system believes it will interact with. The larger the number of profiles the higher is the probability that the system contemplates profiles "close" to the that of the users it will interacting with. But in our simulations there is zero probability that

1. $\Phi = \emptyset$, P =set of profiles, $AdviceSet = \emptyset$
2. **do** {
3. Present the *AdviceSet* to the user.
4. q_s = initial query or one in the *AdviceSet*;
5. Infer constraints analyzing sq, and add them to Φ;
6. Remove incompatible profiles from P;
7. Compute candidate queries C;
8. Obtain *AdviceSet* by discarding from C the dominated queries;
9. *(optional)* Filter the *AdviceSet*;
10. } **while** ((*AdviceSet* $\neq$ null) and (user wants advice))

Figure 1: Interaction process management algorithm

P contains exactly the profile of the (random) user that is querying the database.

We must observe that the assumption that the user profiles are randomly distributed represents the worst situation the system may face. In practice true user profiles tend to cluster into groups of users with similar preferences. In this paper we wanted to face this worst case. We will consider milder scenarios in a future work.

The initial query submitted by the virtual user is created in accordance with her true utility function; thus, the initial query includes the t most important features for the user ($t = 2$ in our experiments). When the cost is among the two most important features the initial query requests a product with the lowest cost. This is because in our model the component of the utility determined by the cost feature (see Equation 1) increases as the cost decreases.

We measured the quality of the query suggestions. In particular we measured the utility shortfall, which is the difference in utility between the last query in the *AdviceSet*, which was selected by the user, and the product in the database with the largest utility for her. This measure indicates if the advisor's suggestions do converge to the best product according to the true user utility function, hence if the final product recommendations are optimal. Moreover, in order to understand how many features differ between the user best product and the products satisfying the last query considered by the user, which in a real scenario would be the products actually shown to the user, we measured their Jaccard similarity. This is the ratio of the number of features common to the best product and the last query over the number of features set to 1 in their union. We also measured the averaged number of queries suggested (advice set size) by the system at each interaction step, and the averaged number of user-system interaction steps, that is, the averaged number of queries issued to the system by the user.

In our experiments we have used three hotel databases (Trentino, Marriott and Cork) with products described only by Boolean features. In addition we considered the Trentino-C hotel database having the same products of Trentino but including in their description the cost feature. This is represented by 4 additional Boolean attributes, i.e., we represent the cost with four different price levels: bargain $v_1 = 11$, cheap $v_2 = 21$, medium $v_3 = 30$, and expensive $v_4 = 41$. Each database describes real hotels by their amenities. Details of the data sets are given in the Table 2. In the experiments where the cost was considered we observed the effect

Table 2: Product databases (Dist. Hotels = Distinct Hotels)

Name	Attributes	Hotels	Dist. Hotels
Marriot	9	81	36
Cork	10	21	15
Trentino	10	4056	133
Trentino-C	14	4056	133

produced by this additional feature on the system performance (utility shortfall, interaction length, advice set size). For each experiment we considered four different sizes of the predefined set of user profiles P: small (25 profiles), medium (250 profiles), large (2,500 profiles) and very large (25,000 profiles).

In each experiment a set of predefined user profiles is created by first generating one totally random user profile (weights vector), sampling each random feature weight from a uniform distribution in [0,1], and then normalizing the user profile vector so that the sum of the weights is 1. Then the other profiles are created by random permutations of the feature weights of that initial user profile.

We have also considered an approach to reduce the size of the *AdviceSet* that we have called *Filtering*. It consists of finding in the *AdviceSet* the queries with the largest expected utility for the user. The expected utility of a query in the *AdviceSet* is estimated by averaging the utility of the query for all the profiles compatible with the inferred constraints. This approach assumes that the compatible profiles have equal probability to be the true profile of the user. In our experiments we have run simulations where only the top 5 queries are suggested.

Thirty-two experiments were performed corresponding to the possible combinations of the variables mentioned before (product database, number of user profiles and filtering strategy). In every experiment we ran 100 simulated interactions between a virtual user and the advisor, and then averaged the observed measures.

6. RESULTS AND DISCUSSION

6.1 Example of Simulated Interaction

Before describing the results of the system evaluation we want to illustrate with one example a typical user-advisor interaction process. We consider here the Trentino-C dataset and 250 predefined profiles. Moreover, the system is using the utility-based *Filtering* strategy mentioned above, hence only five queries will be recommended at each interaction step. The first two user-system interaction steps are illustrated in Table 3.

In this example a hotel is described by the following 14 Boolean attributes: 1) *CityCenter*, 2) *Shop*, 3) *Ski*, 4) *Restaurant*, 5) *PetsAllowed*, 6) *TwinBeds*, 7) *Kids*, 8) *Handicapped*, 9) *Tv/Sat*, 10) *Phone*, 11) *Price*=11, 12) *Price*=21, 13) *Price*=30, 14) *Price*=41. The first 10 correspond to true Boolean features of the hotel, while the other 4 represent the room price. The best option for this user that is present in the catalogue is $\{1, 2, 4, 5, 6, 7, 8, 9, 10, 11\}$.

The interaction starts when the user enters the initial query {1,5}. This is the best 2-features query because w_1 and w_5 are the two largest weights. Observing this query the system infers 18 constraints: $w_1 \geq w_i$ and $w_5 \geq w_i$, $i = 2 \ldots 11$, and $i \neq 5$. The system then discards the profiles that do not satisfy these constraints and finds that 7 out of the 250 initial predefined profiles satisfy the inferred constraints (compatible profiles). The system then assumes that the true user profile is one of these seven that are still compatible with the inferred constraints. The set of candidate queries is then computed and those queries that are not satisfiable are discarded. In our data set the system constructs 528 satisfiable queries using the operators described in Section 2. Then the dominated queries are discarded, and the remaining ones (28 in our example) are ranked by computing their expected utility and finally, the top 5 are suggested (are shown in Table 3).

At the next step the user selects the query {1, 4, 5, 9}, since it maximizes her utility. Note that the query {1, 5, 6, 9} has a larger utility for the user, because $w_6 > w_4$, but it was not suggested. In fact, the system does not know the true user utility, and (erroneously) assumes that the user utility is one of the compatible profiles. These profiles (not shown here for lack of space) erroneously assign a higher weight to features 4 (*Restaurant*) instead of feature 6 (*TwinBeds*). Next the user selected the query {1, 4, 5, 9}. The system infers 4 new constraints $\{w_9 \geq w_8, w_4 \geq w_8, w_9 \geq w_{11}, w_4 + w_9 \geq w_8 + w_{11}\}$. The number of compatible profiles is now reduced to 3. New candidates queries are computed using the query editing operators. Five of them are undominated and therefore they are suggested. The user selects the query {1, 4, 5, 7, 9, 10} that still differs from the optimal query by 4 features. Then the interaction go through two additional steps (not shown here for lack of space) and finally the system suggests the best query for this user: {1, 2, 4, 5, 6, 7, 8, 9, 10, 11}. The utility shortfall is 0 and the Jaccard similarity is 1.

6.2 Interaction Length

Table 4 shows the results of our experiments. In Marriott (9 features) the interaction length ranges between 2 and 3, in Cork and Trentino (10 features) ranges between 2 and 4, while in the case of Trentino-C (cost feature considered, i.e., 14 features) the interaction length ranges between 2 and 5. Interestingly when the size of the user profiles set is small (25 profiles) the interaction length is shorter, ranging between 2 and 2.6. This occurs because the system is more likely to fall into a situation where no user profile is compatible with the inferred constraints and it cannot suggest a new query.

In general, the higher the number of product features the longer the interaction is. This happens because at each interaction step a revised query adds one or two features to the current query. Moreover, the smaller the number of products the more likely the process is to stop, because the current query cannot be further revised without building a failing query. It is important to note that the interaction length is typically low and fairly acceptable for an online application.

6.3 Advice Set Size

When the *Filtering* strategy is not applied the average advice set size can contain up to 13 queries, in the three data sets that do not include the cost feature, and up to 20 in Trentino-C. This difference is due to the fact that when the system generates a query involving the cost feature, it

Table 3: An example of the user-system interaction

											Price attributes			
Boolean attributes:	1	2	3	4	5	6	7	8	9	10	11	12	13	14
True user profile:	0.153	0.058	0.008	0.128	0.156	0.136	0.059	3.0E-4	0.146	0.044	0.111			
Best hotel:	{1, 2, 4, 5, 6, 7, 8, 9, 10, 11}				**initial profiles:** 250			**Top K:** 5						

User	**Advisor**
Initial query = {1, 5}	a) Number of inferred constraints = 18. b) Number of compatible profiles = 7. c) Number of satisfiable queries = 528. d) Number of undominated queries = 28. e) AdviceSet: {{1,4,5,8}, {1,4,5,9}, {1,5,8,9}, {1,4,5,11}, {1,5,8,11}}
Selected query: {1,4,5,9}	a) Number of inferred constraints = 4. b) Number of compatible profiles = 3. c) Number of satisfiable queries = 524. d) Number of Undominated queries = 5. e) AdviceSet: {{1,2,4,5,9,10}, {1,4,5,7,8,9}, {1,4,5,7,9,10}, {1,4,5,8,9,10}, {1,2,5,7,9,10}}
After two additional interaction steps the final **utility shortfall** = 0, and the **Jaccard similarity** = 1	

Table 4: Averaged values of the observed measures for 100 runs in the 24 experiments performed. (DB = Product Database; # Prof. = Number of predefined user profiles; IL = Interaction Length; AdvSS = AdviceSet Size; USh = Utility Shortfall; JSim = Jaccard Similarity)

DB	**# Prof.**	**Not filtering**				**Filtering**			
		IL	**AdvSS**	**USh**	**JSim**	**IL**	**AdvSS**	**USh**	**JSim**
Marriott (9 features)	25	2.13	1.12	0.167	0.594	2.13	1.12	0.167	0.594
	250	2.61	8.66	0.033	0.857	2.93	3.33	0.037	0.825
	2500	2.98	7.93	0.0	0.994	3.0	4.25	0.003	0.965
	25000	2.99	7.82	0.0	0.996	3.0	4.22	0.004	0.965
Cork (10 features)	25	2.57	0.65	0.177	0.575	2.57	0.61	0.177	0.575
	250	3.09	8.32	0.063	0.778	3.6	2.98	0.031	0.895
	2500	3.69	8.43	0.005	0.968	3.81	3.40	0.0	0.991
	25000	3.81	7.69	0.0	1.0	3.84	3.43	0.0	0.993
Trentino (10 features)	25	2.11	0.51	0.324	0.462	2.11	0.5	0.324	0.462
	250	2.95	11.31	0.163	0.626	3.65	2.95	0.080	0.761
	2500	3.65	12.67	0.060	0.797	3.96	3.66	0.018	0.876
	25000	3.99	11.55	0.015	0.890	4.01	3.62	0.012	0.891
Trentino-C (cost considered)	25	2.81	0.75	0.425	0.459	2.88	0.39	0.417	0.474
	250	3.74	14.25	0.227	0.612	5.06	2.29	0.096	0.832
	2500	4.22	18.34	0.159	0.681	5.28	3.21	0.032	0.867
	25000	4.22	20.53	0.085	0.710	5.76	3.28	0.005	0.97

must also suggest one cost value (among the 4 available). This is true regardless the particular modeling approach, which in our case exploits Boolean attributes. Inspecting the experiments' log data, we also found that in the initial steps of the user-system interaction, i.e., when the system has poor knowledge about the user preferences, the average number of suggested queries could be as high as 45 (when the system is contemplating a large number of profiles).

However, when the system is *Filtering* the *AdviceSet*, obviously, the number of queries suggested is never greater than 5. In general, when the size of the set of predefined user profiles is small (25 profiles), the number of query suggestions is smaller that 1.5. This small average number of suggested queries is caused by the possible lack of compatible user profiles, which ultimately makes it difficult to identify queries to suggest.

We note that using a finite set of user profiles produces a significant reduction of the advice set size by more than 10 times compared to the infinite model described in [4] (see also [3] for more details).

6.4 Utility Shortfall and Jaccard Similarity

We expected to observe a higher utility shortfall when *Filtering* the *AdviceSet*. In fact, in this case the system may not include in the *AdviceSet* the best next query hence producing a loss in the user utility compared with the best query. What mitigates this problem is the fact that the system may still suggest the best query at a subsequent interaction step. For instance, if the current query contains two features and the best query contains two additional features, then the system, when filtering the suggestions, may not recommend this query at the first step, but it could do that at a next suggestion step.

In general the utility shortfall decreases as the number of user profiles in P increases. This is true regardless of whether *Filtering* is used or not, or whether the cost feature

is considered. When the number of user profiles is small (25 profiles) the utility shortfall values are higher, ranging between 0.2 and 0.43. This is essentially due to the fact that very often the user profiles do not satisfy the constraints inferred by the system. This causes the interruption of the interaction at an early stage. In this situation there is not a large difference in the utility shortfall whether *Filtering* query suggestions is used or not, because the size of the advice set is always small (smaller than 5).

When the system filters the query suggestions and the user profiles set size is medium (250 profiles) or even larger (2,500 profiles), the utility shortfall of the *Filtering* approach is very close to that of the baseline (no filtering). Moreover, in some experimental conditions (e.g., Trentino and 2,500 profiles) the *Filtering* may even perform better than not filtering (0.018 vs. 0.060). This could happen for a very simple reason. When the system suggests fewer queries, the selection of one of these queries by the simulated user causes the system to infer fewer constraints on the user utility function. In fact the system can only deduce that the selected query does not have a lower utility (for the user) than the other suggested queries. Inferring fewer constraints causes the system to eliminate fewer profiles and hence enables the system to make a larger number of interaction steps before arriving at the possibly failing situation that no profile is compatible with the inferred constraints. This is confirmed by the fact that in these cases (e.g., Trentino and 2,500 profiles) the *Filtering* approach behaves better than not filtering.

When the system is contemplating a large number of user profiles (25,000 profiles) filtering the query suggestions has a very small effect. The increase in the utility shortfall compared with the not-filtering approach is negligible (Marriott 25,000, 0.0 vs. 0.004), and in Trentino-C filtering produces even a smaller utility shortfall. Actually, in this data set filtering produces always lower utility shortfall than not filtering.

Discussing now the Jaccard similarity between the best hotel and the last selected query we must observe that it increases when the number of predefined profiles increases. It is is larger than 89% when the system is contemplating 25,000 profiles. Moreover, this value is larger (96%) in the smaller data sets (e.g. Cork and Marriott). These results confirm the conclusions drawn before about the utility shortfall; it is more likely that better query suggestions are computed when the number of predefined user profiles is larger.

Finally, we note that when the cost feature is considered the system performance is not substantially affected. That is, the utility shortfall is very close to optimal values, especially when the filtering strategy is used. For example when considering 25,000 profiles the utility shortfall is 0.0052, which means that approximately every 200 user-system dialogues there is just 1 non optimal query suggestion. Moreover, the interaction length and the advice set, as mentioned before, are small and acceptable for an effective true user-system interaction.

7. RELATED WORK

Recommending personalized query revisions by inferring constraints on the user utility function by observing the user selection was first proposed in [4] and extended in [11]. This approach was proved to be effective: it was shown that good query suggestions were provided, and the final product recommendations were also optimal. However, [4, 11] left open some questions mostly related to the efficiency of computing the query suggestions and the size of the advice set. In fact, linear programming was extensively used to prove that a potential query suggestion was dominated and hence discarded. Ultimately, the proposed technique for finding the dominated queries required too much computation time to be exploited in an online application. Moreover, on average the set of queries suggested to the user at each interaction step was found to be in many cases too big to be effectively presented to a user.

We initially tackled these issues in a preliminary workshop paper [2] by assuming that the true user utility function is not arbitrary, but it is drawn from a predefined finite set of user profiles known by the system. This set represents the range of possible different users that the system assumes it may interact with. We proved that this assumption simplifies the computation of query suggestions and we showed that the average number of query suggestions made at each interaction step is also dramatically reduced (by a factor of 10). However, it remained the case that, during the initial steps of the query suggestion interaction process, i.e., when the system knowledge about the user preferences is poor, the number of queries suggested could still be large. Moreover, we naively assumed that the true user utility function is included among the finite set of user profiles contemplated by the system. This was a crude simplification since a totally unknown user approaching the system may have an arbitrary profile and the system has no knowledge about that. In another workshop paper [3] we have lifted that assumption and we have also extended the type of query editing operations, showing that this set can be arbitrarily defined by the system designer. In this paper we have included all the mentioned extensions and in addition we have proposed an approach for considering discrete features in the product description.

Critiquing is another query revision approach that is related to our technique [8]. In critiquing the user is offered query revisions in the form of critiques to the current selected product. The main difference with our approach is that the query processing in critiquing is based on similarity-based retrieval, while here we are using a logic based approach. Interestingly, in [14, 10] the authors use a multi-attribute utility-based preference model and critiquing suggestion technique that has similar objectives to our approach. They maintain for each user an estimated profile (utility function). Then, they generate the best critiques using the estimated user utility and update the estimated profile by increasing the importance of a feature (weight) if the selected product has a larger feature value compared to the previously selected one.

Similarly to the "dominated query" concept considered in our work, in [13, 12] the authors consider a conversational recommender system based on example-critiquing that suggest the top K options with the highest likelihood to be "not dominated" by others options (Pareto optimality) [6]. The suggestions are based on an analysis of the user current preference model (adapted in each interaction) and the user reaction to the suggestions. In our case we take into account the query submitted by the user (user reaction) in order to generate new queries, and only those that are proved to have the highest utility according to the user model inferred so far are considered as not dominated, and thus suggested to the user.

Reducing the number of user-system interaction in finding the target products has been approached in critiquing-based systems through the use of compound critiques [9, 14] which enable the user to express her preferences on multiple attributes at the same time, potentially shortening the interaction cycles. In our approach we enable the user to express implicitly her preferences requesting more than one feature at a time, which reduces the number of cycles needed to reach the best product for the user and making inferences on the true user model is kept simple.

8. CONCLUSIONS AND FUTURE WORK

In this paper we have described and analyzed the performance of a conversational recommender system that suggests query revisions to help the user to find the products with the largest utility. We have assumed that the system contemplates only a finite set of possible user profiles, and interacts with a user who has an unknown profile (probably) close to one of those that the system contemplates.

The results of our experiments on three different data sets have shown that the finite user profiles set assumption has a strong effect on the process of computing the best query revisions compared to the infinite models described and analyzed in [4, 11].

First of all, the number of user-advisor interaction steps, which corresponds to the number of queries issued by the user is smaller. Moreover, we have observed a significant reduction in the number of suggested queries at each user-advisor interaction step. This positive results are obtained by paying a negligible increase of the utility shortfall, i.e., in a very small set of interactions the user may end the interaction without selecting the best available product. Additionally we have shown that considering larger and larger sets of predefined user profiles one can mitigate this problem, and filtering the query suggestions can further improve the system performance. We have also shown that discrete features, such as the product cost, can be managed effectively.

We must observe that in the current model the user utility is assumed to be a linear function of the utility produced by each feature. We plan to investigate the use of more general aggregation functions such as Choquet and Sugeno integrals, or Ordering Weighted Averaging functions [1]. This will also be useful for modeling the possible interactions between product features (redundancies, complementarities, contradictions).

Moreover, in this work we have assumed that the user preferences do not change during the interaction with the system and the user is rational (always selects the best option). In fact, the user may change her preferences or not select the best available option (given her current utility function), and this may generate an inconsistent set of inferred constraints preventing the system to produce new query suggestions. We are planning to tackle these issues using relaxation techniques for over-constrained problems [7].

Finally, we must observe that in order to fully evaluate the proposed approach we must perform live user experiments. Therefore, we are implementing a mobile application for hotel recommendation that exploits the proposed technique.

9. REFERENCES

[1] G. Beliakov, T. Calvo, and S. James. Aggregation of preferences in recommender systems. In *Recommender Systems Handbook*, pages 705–734. 2011.

[2] H. Blanco, F. Ricci, and D. Bridge. Conversational query revision with a finite user profiles model. In *IIR 2012, Procs. of the 3rd Italian Information Retrieval Workshop*, volume 835, pages 77–88. CEUR-WS, 2012.

[3] H. Blanco, F. Ricci, and D. Bridge. Recommending personalized query revisions. In *Decisions@Recsys 2012, Proceedings of the 2nd Workshop on Human Decision Making in Recommender Systems*, volume 893, pages 19–26. CEUR-WS, 2012.

[4] D. Bridge and F. Ricci. Supporting product selection with query editing recommendations. In *RecSys '07: Proceedings of the 2007 ACM conference on Recommender systems*, pages 65–72, New York, NY, USA, 2007. ACM Press.

[5] R. Buyya and K. Bubendorfer. *Market-Oriented Grid and Utility Computing*. Wiley Publishing, 2009.

[6] B. Faltings and P. Pu. The lookahead principle for preference elicitation: Experimental results. In *In Seventh International Conference on Flexible Query Answering Systems (FQAS*, pages 378–389, 2006.

[7] U. Junker. Quickxplain: Preferred explanations and relaxations for over-constrained problems. In *Proceedings of the 19th National Conference on Artificial Intelligence*, pages 167–172. AAAI Press / The MIT Press, 2004.

[8] L. McGinty and J. Reilly. On the evolution of critiquing recommenders. In F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors, *Recommender Systems Handbook*, pages 419–453. Springer Verlag, 2011.

[9] J. Reilly, K. McCarthy, L. McGinty, and B. Smyth. Dynamic critiquing. In *Advances in Case-Based Reasoning, 7th European Conference, ECCBR 2004, Madrid, Spain, August 30 - September 2, 2004, Proceedings*, pages 763–777, 2004.

[10] J. Reilly, J. Zhang, L. McGinty, P. Pu, and B. Smyth. Evaluating compound critiquing recommenders: a real-user study. In *EC '07: Proceedings of the 8th ACM conference on Electronic commerce*, pages 114–123, New York, NY, USA, 2007. ACM.

[11] W. Trabelsi, N. Wilson, D. Bridge, and F. Ricci. Comparing approaches to preference dominance for conversational recommender systems. In E. Gregoire, editor, *Procs. of the 22nd International Conference on Tools with Artificial Intelligence*, pages 113–118, 2010.

[12] P. Viappiani, B. Faltings, and P. Pu. Preference-based search using example-critiquing with suggestions. *J. Artif. Intell. Res. (JAIR)*, 27:465–503, 2006.

[13] P. Viappiani, P. Pu, and B. Faltings. Conversational recommenders with adaptive suggestions. In *Proceedings of the 2007 ACM conference on Recommender systems*, pages 89–96. ACM, 2007.

[14] J. Zhang and P. Pu. A comparative study of compound critique generation in conversational recommender systems. In *Procs. of 4th Intl. Conf. on Adaptive Hypermedia & Adaptive Web-Based Systems*, pages 234–243. Springer-Verlag, 2006.