Recommending Personalized Query Revisions

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

Derek Bridge
Department of Computer Science
University College Cork
Cork, Ireland.
d.bridge@cs.ucc.ie

ABSTRACT

Observing the queries selected by a user, among those suggested by a recommender system, one can infer constraints on the user's utility function, and 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 (or utility functions), of the users it may interact with, is finite. We show that making query suggestions is simplified, and the number of suggestions is strongly reduced. We also found that even if the system is not contemplating the true user profile, among the above mentioned finite set of profiles, its performance is still very close to the optimal one.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*information filtering*

General Terms

Experimentation, Theory.

Keywords

Recommender system, conversational system, user model.

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 a recommendation [6].

In [3, 9] the authors introduce and evaluate a new conversational technique for helping the users to select items of largest utility to the user. In order to accomplish this goal, when a user is querying a product catalogue the proposed technique suggests to the user new queries that: a) extend the user's current query, and b) retrieve products with higher utility. For example the user of a hotel catalogue may have submitted the following query: "I want an hotel with AC and parking". The system, rather than retrieving immediately the products that satisfy this query, hypothesizes that the user may have also other needs and makes recommendations by suggesting queries that are revisions of the original query. These new queries may add one or more additional features to the query, e.g., the system may say: "are you interested also in a sauna?". Products with more features, if available, will surely increase (or, at least, 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 those features that are likely to produce the largest increase to the user's utility. In fact, observing the user's previously submitted queries, the system can deduce that certain features are more important than others, i.e., it can infer constraints on the user's utility function, even without knowing that function.

The major limitations of the previous work on this proposed technique were: a) a limited number of query editing operations, 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 long list of query suggestions could be possibly presented to the user, making it hard for her to evaluate them and select her preferred one. In this paper we propose a new effective technique for the computation of the dominated queries, i.e., the queries that should not be suggested to the user because the system can deduce that they have a lower utility. The proposed technique relies on the system assumption that the set of profiles (or 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 ever 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 conversational step is greatly reduced. We also show that the query suggestions can be further filtered by estimating the utility of each query suggestion using those profiles that are compatible with the queries previously selected by the user. The proposed approach has also another advantage, it enables a system designer to freely select the types

Paper presented at the 2012 Decisions@RecSys workshop in conjunction with the 6th ACM conference on Recommender Systems.

of query editing operations that he or she would like to use to generate new query suggestions to the user.

We also show that even if the system is not contemplating the 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 dialogue the user can select the best products, given her true profile and the available products in the data set. Hence the finally recommended items are close to the optimal ones. In fact, we show in this paper that progressively expanding the number of profiles contemplated by 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.

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 user preferences. 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

In our model a product p is represented by an n-dimensional Boolean feature vector $p = (p_1, \ldots, p_n)$. $p_i = 1$ means that the i-th feature (e.g., Air Conditioning) is present in the product, whereas $p_i = 0$ means that p does not have feature i. A catalogue is a set of products $\{p^{(1)}, \ldots, p^{(k)}\}$. The Boolean features could be keywords or tags found in the product description, and searching for products with these features can be viewed as kind of facet search.

Queries are represented similarly as Boolean vectors: $q = (q_1, \ldots, q_n)$. $q_i = 1$ means that the user is interested in products that have the i-th feature. On the other hand $q_i = 0$ does not mean that the user is not interested in products with that feature, but simply that she has not yet declared her interest on it. A query is said to be *satisfiable* if there exists a product in the catalogue such that all the features expressed in the query as desired ($q_i = 1$) are present in that product. For example if the product $p = (1, 1, 0, 1, 0)$ is present in the catalog then query $q = (0, 1, 0, 1, 0)$ is satisfiable.

We are considering a scenario where the user is advised about how to refine her queries. Moreover, we assume that the system GUI offers to the user a limited number of easily understood editing operations (as in critiquing-based approaches). In the following we list the query editing operations that, in this paper, we assume the user can make when revising the current query. But we observe (as 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 set of indexes with value 0 and 1 in q respectively. The first two operations generate a new query by adding to the current query a request for one additional feature. For example, in $(1, 1, 0, 0, 1) = add_1((1, 1, 0, 0, 0), 5)$ the query $q = (1, 1, 0, 0, 0)$ (where the first two features are requested) is extended by requesting also the fifth feature. The second operation (trade one feature present for two not present) generates a new query by discarding a feature, the i-th, in favor of two new ones, the j-th and k-th features. For example, $(0, 1, 0, 1, 1) = trade_{1,2}((1, 1, 0, 0, 0), 1, 4, 5)$.

The last two operations extend a query with two additional features. For example, given the query $q = (1, 1, 0, 0, 0)$, the fourth and fifth features can be requested in the new query generated by the operation $add_2(q, 4, 5) = (1, 1, 0, 1, 1)$. The second "trade" operation (trade one feature present for three not present) generates a new query by discarding the i-th feature, but now in favor of three new ones, the j-th, the k-th and t-th features. For example, $(0, 1, 1, 1, 1) = trade_{1,3}((1, 1, 0, 0, 0), 1, 3, 4, 5)$.

Using the above-mentioned operators the system can generate a set of next queries and ask the user to select her preferred one, and this step can be repeated several times (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 queries, as a standard "query by example" interface might, 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 remove from the suggestions it makes at every step those queries that appear to the system to have an inferior utility. This reasoning process is clarified in the following sections.

3. USER UTILITY FUNCTION

A user's utility function, also called her user profile, is represented here as a vector of weights: $w = (w_1, \ldots, w_n)$, $0 \leq w_i \leq 1$. w_i is the importance that the user assigns to the i-th feature of a product. So if $w_i = 0$, then the user has no desire for the i-th feature. If $w_i > w_j$, then the i-th feature is preferred to the j-th one. If $w_i \geq w_j$ then the i-th feature is at least as desired as the j-th one. If $w_i = w_j$, $i \neq j$ then the user is indifferent between these two features. The user's utility for a particular product $p = (p_1, \ldots, p_n)$ is given by the following:

$$Utility_w(p) = \sum_{i=1}^{n} w_i p_i \quad (1)$$

A product p with a higher utility than another product p' is always assumed to be preferred by the user, i.e., we assume that users are rational. A user may have any of the possible utility functions that can be defined by varying the feature weights w_i. So, the set of all possible utility functions is infinite. But observing the queries selected by the user among those suggested by the system the system can infer constraints on the definition of the true user utility function. Generally speaking, features that are present in a query that the user selects can be inferred to be more desirable for that user than features that are present in the alternative queries that the user could have tried but did not select.

More precisely let us assume that the system recommends to the user a set of new queries, which we will call the AdviceSet. The queries in the AdviceSet will, in general, be a subset of the queries that can be generated by applying the query editing operations described in the previous section to the query that was selected by the user at the previous

interaction step. When the user selects one of these recommended new queries, as the new best query for her, the system can deduce that the utility of the one she selects is greater than or equal to the utility of the other queries that were included in the AdviceSet. If we define $Utility_w(q)$, the utility of query q for a user with profile w, as the utility of a product p with the same definition as q, i.e., $q = p$, then, if the user selects $q_s \in AdviceSet$, we can infer that:

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

For example: Let's assume that the previous query selected by the user is $q_0 = (0, 0, 1, 1, 0, 0, 0)$, i.e., there are seven features in this data set and the user would like to retrieve products having the third and fourth feature. Assume that the system suggests that the user edits the current query and specifically recommends that she select one of the following four queries:

$$\begin{aligned} AdviceSet = \ & \{(1, 0, 1, 1, 0, 0, 0), \\ & (0, 1, 1, 1, 0, 0, 1), \\ & (0, 0, 1, 1, 1, 0, 0), \\ & (0, 0, 1, 0, 0, 1, 1)\} \end{aligned}$$

Let us further assume that the query that the user selects from these recommended ones is $q_s = (0, 0, 1, 1, 1, 0, 0)$, i.e., she is interested in products that additionally have the 5-th feature. Then, the inferred constraints, based on not choosing other members of the AdviceSet, are:

1. $w_5 >= w_1$
2. $w_5 >= w_2 + w_7$
3. $w_4 + w_5 >= w_6 + w_7$

We must also explain what constraints on the true user profile w can be deduced when the user issues the very first query in any interaction. In this case, if q is the initial query, the advisor will infer that $w_i \geq w_j$, $\forall i \in idx1(q)$ and $\forall j \in idx0(q)$, unless q_s, which is identical to q except that its i-th feature is set to 0 and its j-th feature is set to 1, is unsatisfiable. This means that features requested in the initial query are at least as desired as features not initially requested. But, the system must "play safe". In the case where q_s, identical to q but with the i-th feature set to 0 and the j-th feature set to 1, is unsatisfiable, it should not deduce a constraint of the type $w_i \geq w_j$. This is because there is the possibility that the user already knew this query to be unsatisfiable and for this reason she did not try it as her initial query, even though she preferred it. A longer discussion of this "play safe" rule is given in [3].

4. THE QUERY ADVISOR

The advisor is the recommender system in charge of suggesting to the user how to extend the current query to obtain better products, i.e., it generates the AdviceSet. The true user's preferences, in her profile, are not known by the advisor. Moreover, we assume that the advisor does not explicitly ask the user for her preferences. Nevertheless, right after the user's first query, the advisor will generate a set of candidate queries and will recommend only the undominated candidates, i.e. those with a utility that cannot be proved to be inferior to one of the other candidates. Each time the user chooses one of the recommended queries, the advisor makes new recommendations. It does this repeatedly until the user is happy with her current query or no additional suggestions can be made by the system.

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 operations described in Section 2, and that are satisfiable, the advisor will not suggest queries that have a lower utility than another one: these queries are called here "dominated". A query $q \in C$ is *dominated* if there exists another query $q' \in C$ such that for all the possible weight vectors that are compatible with the set of constraints Φ this relation holds: $Utility_w(q') > Utility_w(q)$. 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 $w_1, \ldots, w_n$ 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 induced by observing the user's behavior.

In our previous work, the problem of finding dominated queries was cast as a linear programming problem, allowing an infinite number of user profiles to be considered. The problems with this are discussed in Section 7. In this paper we assume that the set of user profiles contemplated by the system is finite. Initially, at the beginning of the interaction with a user, the set of all the possible utility functions or user profiles is $P = \{w^{(1)}, \ldots, w^{(m)}\}$. We will consider in the experiments sets of user profiles ranging from some dozens to thousands.

With this finite assumption, having the set of deduced constraints Φ we can prune from the set P the "incompatible profiles", i.e., those not satisfying the constraints in Φ. Then, the computation of the undominated queries proceeds as follow. Let's 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., queries that are satisfiable and are generated from the last issued query of the user by the query editing operations. Then the AdviceSet is given by the following method:

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 $\forall w \in P'$, $Utility_w(q') > Utility_w(q)$. , i.e., $\sum_{i=1}^{n} w_i q'_i > \sum_{i=1}^{n} w_i q_i$.
2. Build the AdviceSet (undominated queries) by removing from C the dominated queries.

Example. Assume that $\Phi = \{w_1 \geq w_3, w_2 + w_3 \geq w_4\}$, $P' = \{w^{(1)}, w^{(2)}, w^{(3)}\}$ and $C = \{q^{(1)}, q^{(2)}, q^{(3)}, q^{(4)}\}$, $w^{(1)} = (0.35, 0.1, 0.25, 0.3)$, $w^{(2)} = (0.1, 0.35, 0.3, 0.25)$, $w^{(3)} = (0.3, 0.35, 0.1, 0.25)$, $q^{(1)} = (1, 1, 0, 1)$, $q^{(2)} = (1, 0, 1, 1)$, $q^{(3)} = (0, 1, 1, 1)$, $q^{(4)} = (1, 1, 1, 0)$.

In this example the profiles $w^{(1)}$ and $w^{(3)}$ satisfy the constraints in Φ. While, $w^{(2)}$ is an "incompatible profile", since $w_1^{(2)} < w_3^{(2)}$, and must be pruned from P'. Table 1 shows the query utilities for these two compatible profiles. $q^{(1)}$ has a higher utility than $q^{(3)}$ and $q^{(4)}$ for every profile in P', thus $q^{(3)}$ and $q^{(4)}$ are dominated by $q^{(1)}$. These dominated

Table 1: Query utilities for the profiles $w^{(1)}$ and $w^{(3)}$.

	$\mathbf{q^{(1)}}$	$\mathbf{q^{(2)}}$	$\mathbf{q^{(3)}}$	$\mathbf{q^{(4)}}$
$\mathbf{w^{(1)}}$	0.75	0.9	0.65	0.7
$\mathbf{w^{(3)}}$	0.9	0.65	0.7	0.75

1. $\Phi = \emptyset$, P =set of profiles, AdviceSet = empty set
2. **do** {
3. Present the AdviceSet to the user.
4. sq = 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;
8. Remove dominated queries from candidate ones and generate AdviceSet;
9. *(optional)* Filter the AdviceSet;
10. } **while** ((AdviceSet $\neq$ null) and (user wants advice))

Figure 1: Interaction process

queries must be removed from the set C and not included in the AdviceSet. Note that the remaining queries $q^{(1)}$ and $q^{(2)}$ do not dominate each other, thus they represent meaningful next queries that the advisor can recommend to the user.

The full algorithm for query suggestions is described in Figure 1. At the first step there are no query suggestions, and the user is free to enter the first query. 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. Afterwards, the set of candidate queries is generated from the current query, by applying the operators mentioned in Section 2 and discarding any queries that are not satisfiable. Subsequently, the advisor builds the AdviceSet by removing the dominated queries, and optionally filters the AdviceSet to keep it small. The filtering strategy that we have applied 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, and the user selects one from this advice set, then the selected query becomes the current query and the process is repeated. If the user does not want further advice then the system will display the products that satisfy the last query selected by the user.

5. EXPERIMENTS DESIGN

We performed several experiments by simulating interactions between a virtual user and the advisor according to the algorithm described in the previous section. For each experiment we varied the following independent variables: product database, number of predefined user profiles, and whether the undominated queries were filtered or not in order to reduce the number of suggestions in the AdviceSet (step 9 of the algorithm). We measured: the average number of queries issued per dialogue (interaction length), the average size of the AdviceSet(number of queries suggestions at each step), the utility shortfall, and the Jaccard similarity between the last selected query and the optimal one. The utility shortfall (or "regret") is the difference between the utility of the best product available in the data set, i.e., the one with the highest utility for the user, and the utility of the products satisfying the last query selected by the user. This measure indicates if the advisor's suggestions do converge on the best product according to the true utility function, hence if the final product recommendations are optimal. Moreover, in order to understand how many features differ between the user's 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 in their union. In practice, the utility shortfall can be very small (if the features that differ in the best product and in the last query have small weights in the user's utility function), but the Jaccard similarity could still be far from 1.0.

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

Name	Features	Hotels	Dist. Hotels
Marriot-NY	9	81	36
Cork	10	21	15
Trentino-10	10	4056	133

Three different product databases were used, each one describing real hotels by their amenities expressed as Boolean features. Details of the data set are given in the Table 2; here a hotel may have the same description in terms of features as another; that's why the number of distinct hotels is smaller. Moreover, we considered for each experiment four different sizes of the set of predefined user profiles: small (25 profiles), medium (250 profiles), large (2500 profiles) and very large (25000 profiles). We wanted to measure the effect of the size of the profiles set on the user-advisor interaction length, and on the size of the advice set.

In each experiment a set of predefined user profiles is created by first generating one totally random initial 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 the initial user profile. Note that with 10 features there are $3.6 \times 10^6 \sim 10!$ possible user profiles.

For step 9 of the algorithm, i.e., the optional filtering of the query suggestions in the advice set to produce an AdviceSet that has at most a small number of suggested queries (5 in our case), we used one strategy. We considered the strategy that selects the top K queries in the AdviceSet, with the largest expected utility. The expected utility of each query in the AdviceSet is computed 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 addition to the user profiles contemplated by the advisor, in each simulated interaction we randomly generated the true profile of the virtual user and it was not revealed to the advisor. Note that the true virtual user profile is very unlikely to be among the predefined set of advisor user profiles. Moreover, the initial query submitted by the virtual user is created in accordance with her true utility func-

tion; thus, the initial query includes the t most important features for the user ($t = 2$ in our experiments). The advisor's deductions about the true user utility function are based only on the observation of the queries submitted by the user at each interaction step. We also assumed that the virtual user is "Optimizing" [3], that is, one who confines her queries to the advice set provided by the advisor and always tries the query with the highest utility. Twenty-four experiments were performed corresponding to all the combinations of the variables mentioned before (product database, number of user profiles, filtering strategy). In every experiment we ran 100 dialogues 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. In this example we are considering the Marriott catalogue, and the system is using the utility-based filtering strategy, hence no more than 5 queries will be recommended at each step. Some of the details are in Table 3.

The features, numbered from 0 to 8 are: 0=*Internet access point*, 1=*Restaurant on site*, 2=*Room service*, 3=*Pets allowed*, 4=*Meeting room*, 5=*Airport shuttle*, 6=*Swimming Pool*, 7=*Golf camp*, 8=*Tennis camp*. The five most important features for the simulated user in this example are {0,1,2,5,8}, but there is no hotel with exactly these features in the dataset, and the best available hotel is {0,1,2,3,5}.

The user starts the interaction with the query which, according to her preferences, contains the two most important features: {1,2}. The system infers some initial constraints from the initial query, i.e., that these features are more important that the others not requested (see Section 3), and discards the profiles that do not satisfy these constraints. In this case it discovers that 10 out of 250 initial predefined profiles satisfy the inferred constraints (compatible profiles). These profiles are considered by the system as those potentially containing the true user profile and thus will be examined to make new query suggestions. The system computes the next candidate queries and discards those that are not satisfiable: (58 are candidates) Then the advisor removes those that are dominated, the remaining queries (10 undominated) are ranked by computing their expected utility, and the top 5 are suggested. Note that the queries suggested extend the previous one with extra features. The query selected by the user is {1,2,5,6}, since it is the one that maximizes her utility. At this point the utility shortfall is 0.136 and the Jaccard similarity with the best hotel, {0,1,2,3,5}, is 0.5 (3 common features out of 6 in the union). The system infers 4 new constraints: these constraints state that the utility of the selected query is greater than or equal to the utility of the other queries that were suggested. The number of compatible profiles is now 2, and only 1, that is {0,1,2,5,6}, out of the 7 satisfiable queries, is undominated, and thus suggested to the user. It is interesting to note that the best (and satisfiable) query $\{0,1,2,3,5\} = trade(\{1,2,5,6\},6,1,3)$ was (erroneously) considered by the system to be dominated by the suggested query {0,1,2,5,6}, and therefore not included in the advice set. This results from the fact that the dominated queries are computed using the compatible profiles (2 in this case) not the true user model, which is unknown to the system. These two compatible profiles (erroneously) assign a higher weight to feature 6 (*Swimming Pool*) instead of feature 3 (*Pets allowed*) as it is stated in the true user profile. In the third interaction step the user is forced to select the unique query that is suggested. At this point it is not possible to extend the current query with a satisfiable one any further, the system cannot make new query recommendations, and the interaction ends. The utility shortfall and the Jaccard similarity are 0.0018 and 0.67 respectively. In this example it is clear that reasoning with a finite set of profiles causes some loss in recommendation accuracy, which is compensated by a speed up in system performance and a reduction in the sizes of the AdviceSet compared with the approach introduced in [3] (see discussion later).

6.2 Interaction Length

Table 4 shows the results of our experiments. The query suggestion strategy based on the utility filtering, as well as the baseline approach (not filtering the query suggestions), produce interaction sessions with average length ranging between 2 and 4.

When the size of the user profiles set is small (25 profiles), the interaction length is even shorter, ranging between 2 and 2.6; this is because it is more likely to fall into a situation where no user profile is compatible with the inferred constraints and the system cannot suggest a new query.

In general, the interaction length is dependent on the number of product features and the available products in the data set. Firstly, the higher the number of product features, the longer will be the interaction. This is because the user, at each interaction step, when she is selecting one of the query editing operations, extends the previous query by one or two additional features. Secondly, the smaller the number of products, the more likely the process is to stop, because the current query cannot be further extended 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 AdviceSet Size

The average size of the advice set ranges between 0 and 12 when no filtering is applied. In this case, inspecting the experiments' log data, we detected 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 20 (when the system is contemplating a large number of profiles). On the other hand, when the system is filtering the AdviceSet, obviously, the size of the advice set is never greater than $K = 5$. Table 4 shows the average number of queries suggested and, as expected, the filtering strategy (utility-based) produces smaller AdviceSets compared to the not-filtered case. In general, when the size of the set of predefined user profiles is small (25 profiles), the number of query suggestions ranges between 0 and 1.5; this is caused (as we discussed above for the interaction length) by the lack of compatible user profiles, resulting in the difficulty of identifying queries to suggest to the user.

6.4 Utility Shortfall

We expected to observe a higher utility shortfall when filtering the advice set. In fact, in this case, the system

Table 3: An example of the user-system interaction

hotel features:	0	1	2	3	4	5	6	7	8
true user profile:	0.134	0.264	0.188	0.025	7.0e-4	0.141	0.023	0.06	0.164
best hotel:	{0, 1, 2, 3, 5}		**number of initial profiles:**				250		

User	**Advisor**
* Issues the initial query = {1, 2}	* Number of inferred constraints = 11 * Number of compatible profiles = 10 * Number of satisfiable queries = 58 * Undominated queries = 10. Top K=5 suggested: {{1,2,4,6}, {1,2,5,6}, {1,2,3,6}, {1,2,4,5}, {1,2,3,4}}
* Selects the query: {1, 2, 5, 6}	* Number of inferred constraints = 4 * Number of compatible profiles = 2 * Number of satisfiable queries = 7 * Undominated queries = 1: {{0,1,2,5,6}}
* Selects the query: {0, 1, 2, 5, 6}	* No new constraints inferred. * The same number of satisfiable profiles remains. * Number of satisfiable queries = 0. * The system cannot make more query suggestions.
Utility shortfall = 0.0018, **Jaccard Similarity** = 0.667	

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**				**Utility filtering**			
		IL	**AdvSS**	**USh**	**JSim**	**IL**	**AdvSS**	**USh**	**JSim**
Cork	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
Marriott	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
Trentino	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

may not include in the AdviceSet the best next query, causing, at that step, a loss in the user utility compared with the best query and thus an increase of the utility shortfall. 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, the system, when filtering the suggestions, may not recommend the query using the best of the two missing features at the first step, but it could do it at the next suggestion step.

In general the utility shortfall decreases when the number of user profiles increases. This is true regardless of whether filtering is used or not. When the number of user profiles is small (25 profiles) the utility shortfall values are higher, ranging between 0.2 and 0.3. 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 case there is not a big difference in the utility shortfall whether filtering query suggestions is used or not, because the size of the advice set never exceeds the threshold $K = 5$.

When the system filters the query suggestions and the user profiles set size is medium (250 profiles) or even larger (2500 profiles), the utility shortfall is very close to that of the not-filtered case. Moreover, in some cases (e.g., Trentino and 2500 profiles) the utility-based strategy may even perform better than the not-filtering approach (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 2500 profiles), where the utility-based approach behaves better than the not-filtering one, the interaction length is on average a bit larger (3.96 vs. 3.65).

In the case where the system is contemplating a large number of user profiles (25000 profiles), filtering the query suggestions has a very small effect. The difference in the utility shortfall with the not-filtering approach is still smaller than 0.0041 (e.g., Marriott 25000, 0.0 vs. 0.004), and there is never a gain in utility. In general the Jaccard similarity between the best hotel and the last selected query increases when the number of predefined profiles increases as well. The Jaccard similarity is higher than 89% when the system is contemplating 25000 profiles. Moreover, this value is better (96%) for the smallest data sets (e.g. Cork and Marriott). These results confirm the previous conclusions on the utility shortfall; it is more likely that better system query suggestions are obtained when the number of predefined user profiles is higher. In fact, it is not important to have many profiles, but rather to have an optimal set of predefined user profiles covering the true user profiles of the subjects accessing the system. This is a topic of further research.

6.5 Infinite Profile Set Model

Finally, in Table 5 we compare the interaction length, the advice set size, and the average utility shortfall obtained in our experiments with those measured in our previous work, where an infinite number of profiles was considered [3]. In this comparison we use 25000 profiles and we confine the system to use only the add_1 and $trade_{1,2}$ operators to generate new candidate queries: because in [3] the results were obtained by considering only these two editing operations.

We can observe that the interaction length in the two systems is more or less equivalent. The utility shortfall in the proposed finite profiles model is always a bit larger than in the infinite model. This is what has to be paid for the constraining assumption that the number of possible user profiles is finite. The major beneficial effect of the proposed approach is the significant reduction of the advice set size by more than 10 times. Moreover, computing the advice in our implementation, took just some miliseconds, even if 25000 profiles were used, while with the infinite model it required on average some seconds.

In conclusion, we believe that in real scenarios approximating the set of all possible user's utility functions with a finite set is a reasonable assumption, and the small cost paid in terms of increased utility shortfall is compensated by the strong reduction in the size of the advice set and computational complexity, making it feasible for the user to browse the advice set and pick her best query.

Table 5: Comparison of the system performance between the current finite set of profiles model and the infinite model.

DB	Averaged measures	Infinite model	Finite model
Cork	Interaction length	6.09	5.63
	Advice set size	69.88	4.81
	Utility shortfall	0	0.003
Marriott	Interaction length	4.67	3.98
	Advice set size	45.96	5.08
	Utility shortfall	0	0.001
Trentino	Interaction length	5.55	6.31
	Advice set size	59.02	5.17
	Utility shortfall	0	0.037

7. RELATED WORK

Recommending personalized query revisions was first proposed in [3] and then extended in [9]. This approach has proved to be effective, and provides good query recommendations and final product recommendations. It guides the user to the query that selects the products with maximal utility in a short number of query revision interactions. The cited papers describe the details of this approach: the query language, the user model, and the inferences made by the system, observing the user's query revisions and finally the computation of the query suggestions for the user. [3, 9] left open some questions mostly related to the efficiency of computing the query suggestions and the size of the advice set. That approach uses linear programming extensively and require too much computation time to be exploited in an online application. Moreover, the average number of queries suggested to the user at each interaction step is in many cases too large to be presented to a user.

These problems were initially tackled in a preliminary workshop paper [2] by assuming that the user utility function is not an arbitrary one (i.e., coming from an infinite set) but is drawn from a finite set of user profiles that are known by the system. This set represents the possible different users that the system considers that it may interact with. This assumption simplifies the computation of query suggestions (as was also shown here). Moreover, 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 query suggestion (when the system knowledge about the user preferences is poor), the number of queries suggested can still be high. Moreover, the authors artificially assumed that the true user utility function is included among the finite set of user profiles contemplated by the system. This is a crude simplification since a totally unknown user approaching the system may have an arbitrary profile and the system has no knowledge about that. We have lifted that assumption in this paper and we have also extended the type of query editing operations, showing that this set can be arbitrarily defined by the system designer.

Critiquing is a conversational recommendation approach that is related to our technique [6]. In critiquing the user is offered query revisions in the form of critiques to the current selected product. The main difference with our proposed approach to building conversational recommender systems is that the query processing in critiquing is based on similarity-based retrieval, while here we are using a logic based approach. Interestingly, in [12, 8] 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 [11, 10] the authors consider a conversational recommender system based on example-critiquing that suggest the top K options with highest likelihood to be "not dominated" by others options (Pareto optimality) [4]. The suggestions are based on an analysis of the user's current preference model (adapted in each interaction) and the user's reaction to the suggestions. In our case we take into account the query submitted by the user (user's reaction) in order to generate new queries, and only those that prove 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 [7, 12] 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 new type of conversational recommender system that suggests query revisions to help the user to find products with the largest utility. We assume 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 show that the finite user profiles set assumption has a strong effect on the process of computing the best query suggestions that guide the user to the products that maximize her utility. In particular the number of user-advisor interaction steps (number of queries issued by the user), and the utility shortfall are low. We have observed a significant reduction in the number of pieces of advice (suggested next queries) provided at each user-advisor interaction step. We have also shown that having a relatively large number of predefined user profiles, and exploiting even simple techniques for filtering the suggestions, is an important ingredient for improving the system performance and producing effective support.

In the current model we consider only Boolean features. But, the proposed approach can be extended to ordinal and numerical features (e.g. hotel category and room price). We plan to develop such an extension in the future. It is also important to note that the user's utility function is assumed to be linear. We plan to investigate the use of more general integral aggregation functions such as Choquet and Sugeno, or Ordering Weighted Averaging functions [1]. This will also be useful for modeling 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 perfectly 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 that the system cannot use to produce new query suggestions. We are planning to tackle these issues using relaxation techniques for over-constrained problems [5]. Finally, we must observe that 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 *Procs. of the 3rd Italian Information Retrieval Workshop*. CEUR-WS, 2012.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

[12] 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.