Conversational Query Revision with a Finite User Profiles Model

Henry Blanco[1,2], Francesco Ricci[1], and Derek Bridge[3]

[1] Faculty of Computer Science
Free University of Bozen-Bolzano
Bolzano, Italy
fricci@unibz.it
[2] Center of Medical Biophysics
Universidad de Oriente
Santiago de Cuba, Cuba
[3] Department of Computer Science
University College Cork
Cork, Ireland

Abstract. Information Recommendation is a conversational approach aimed at suggesting to the user how to reformulate his queries to a product catalogue in order to find the products that maximize his utility. In previous work, it was shown that, by observing the queries selected by the user among those suggested, the system can make inferences on the true user utility function and eliminate from the set of suggested queries those retrieving products with an inferior utility (dominated queries). The computation of the dominated queries was based on the solution of several linear programming problems, which represented a major computational bottleneck for the efficiency of the proposed solution. In this paper we propose a new technique for the computation of the dominated queries. It relies on the assumption that the set of possible user utility functions is finite. We show that under this assumption the computation of the query suggestions is simplified and the number of query suggestions is strongly reduced.

Keywords: Recommender system, conversational system, user preference model.

1 Introduction

Recommender Systems (RS) are intelligent tools and applications designed to support users in finding information, products or services that suit their needs and preferences [8]. Recommender system technologies are rooted in Machine Learning and Information Retrieval [4]. The core computational problem of a RS is to predict the user's preferences, e.g. expressed as ratings for items, and recommend the items with maximal predicted preference [8]. Classical RS techniques, such as collaborative and content-based filtering, collect the user's preferences in the form of ratings for items to meet the goals mentioned before. Their major limitation is that they present the recommendations in a single shot, and the user can either accept one of these recommendations or enter new preferences and restart the process. Conversely, Conversational Recommender Systems (CRS) [2, 5, 6] not only rank and suggest products to users, but also guide them during the human-computer interaction to finally select the products that they may like. This guidance process is composed of several actions that depend on the underlying conversational technology (e.g., critiquing [7, 6]).

In [1, 9] the authors first introduce and then extend a new conversational technique relying on the idea of "Information Recommendation". In this approach the user is supposed to query a product catalogue by issuing simple queries, such as "I want an hotel with AC and parking". The system, rather than recommending immediately the products that satisfy this query, assumes that the user may have also other needs and suggests some query revisions. These new queries, for instance, may add an additional feature 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 the user utility. But not all features are equally important for the user. So the goal of the system is to make "informed" suggestions, i.e., to suggest features that are likely to increase more the user utility. In fact, the system, observing the user queries, can deduce that certain features are more important than others, i.e., can infer constraints on the definition of the user utility function, even without knowing it. Hence, using this knowledge, it can suggest that the user try a query from a well-selected and small set of candidate queries . A similar idea, i.e., using a utility function estimation to select the more user relevant critiques (new queries), is described in [10].

In [1][9] it is shown that this approach is effective and provides good query suggestions and final recommendations. It guides the user to the query that selects the products with maximal utility in a short number of query revision interactions. The quoted papers describe the details of the approach: the query language, the possible preference models of the user, the inferences made by the system on observing the user's query revisions, and the computation of the query suggestions for the user. Nevertheless some questions mostly related to the efficiency of the query suggestions computation and the size of the advice set are still open and require further investigation. In fact, the computational cost of query suggestion is playing a critical role in this approach. In [1] linear programming techniques were used for computing the query suggestions. Even if the computational complexity of that algorithm is polynomial, it must be invoked numerous times (to compare each pair of candidate queries), and in practice it takes too much time for a real online application. Moreover, the average size of the advice set, i.e., the queries suggested by the system to the user at each interaction step, remains large in many cases (more than 20). This is a critical issue for implementing a real application based on the proposed technique.

In this paper we refine the proposed model by making the assumption that the user utility function is drawn from a set of finite possibilities. This set of "user profiles" represents the possible "different" users that the system may interact with. We will show that this assumption has a strong effect: it simplifies the search process for the query suggestions and reduces the average number of query suggestions made at each interaction step. This finite model assumption is realistic, as users tend to cluster in groups with similar preferences. Moreover, considering an increasingly large number of user profiles one can approximate all the possible ones.

2 Query Language

In our model a product p is represented by an n-dimensional Boolean feature vector $p = (p_1, \dots, p_n)$. $p_i = 1$ 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 $\{p^{(1)}, \dots, p^{(k)}\}$. 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 [3].

Queries are represented similarly as Boolean vectors: $q = (q_1, \dots, q_n)$. $q_i = 1$ 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 he has not yet declared his 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 may be interested in refining an initial query. Moreover, we assume that the user is not likely to radically modify this query. This may also be a constraint imposed by the GUI of the query system, where the user can be offered with only a small number of easily understood editing operations. In the following we list the query editing operations that we assume the user can make when revising the current query:

- $add(q, i)$, where $i \in idx0(q)$
- $trade(q, i, j, k)$, where $i \in idx1(q)$ and $j, k \in idx0(q)$

where $idx0(q)$ and $idx1(q)$ are the set of indexes with value 0 and 1 in q respectively. The first operation generates a new query by requesting one additional feature. For example, $(1, 1, 0, 0, 1) = add((1, 1, 0, 0, 0), 5)$ is extending a query where only the first two features were requested by adding also the fifth feature to the set of requested ones. The second operation 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, $(0, 1, 0, 1, 1) = trade((1, 1, 0, 0, 0), 1, 4, 5)$

Using the above-mentioned operators the system can generate a set of next queries and ask the user to select the preferred one. In our approach, the goal of the system is not to suggest all these possible next queries, as a standard "query by example" interface may implement, but rather only queries that could retrieve products with the largest utility. Hence, first of all, the unsatisfiable queries must not be suggested. This can be easily implemented with standard query processing techniques. But, as it will be shown later, also other types of queries can be discarded: those that can be proved to retrieve products with a smaller utility than those retrieved by another query in the suggestion list (dominated queries).

3 User Utility Function

User preferences for products are represented here as a vector of weights:

$$w = (w_1, \ldots, w_n), 0 \leq w_i \leq 1 \tag{1}$$

w_i is the importance that a particular user, one having that set of preference weights, 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 utility for a particular product $p = (p_1, \ldots, p_n)$ is given by the following:

$$Utility_w(p) = \sum_{i=1}^{n} w_i \times p_i \tag{2}$$

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. We also define the potential utility of a query $q = (q_1, \ldots, q_n)$ for the user as: $Utility_w(q) = \sum_{i=1}^{n} w_i \times q_i$. We call this utility "potential" if we do not know wether a product with the features specified in the query does exist, i.e., if the query is satisfiable. In case such a product exists, this potential utility is also a true utility.

A user accessing the system may have any of the possible utility functions that can be defined by varying the feature weights w_i. So, in principle, the set of all possible utility functions is infinite. But observing the queries selected by the user among those that he can make (i.e., those suggested by the system), the system can infer constraints on the definition of his utility function. Generally speaking, features present in the selected query can be considered as more desired by the user than features that are present in the alternative queries. The constraints deduced by the system on the user utility function $w = (w_1, \ldots, w_n)$ are illustrated below.

Initial query. If the current query q is the initial query, then the advisor may infer that $w_i \geq w_j$, $\forall i \in idx1(q)$ and $\forall j \in idx0(q)$, unless q, with the i-th feature set to 0 and the j-th feature set to 1, is unsatisfiable. This means that if the user issued a query that requests the presence of a feature then the potential utility of this query is assumed to be larger than or equal to that of another query where this feature is not requested. But only if this "alternative query" is satisfiable.

Adding a feature. If the current query q' results from an $add()$ operation on the previous query, that is, $q' = add(q, i)$, then the advisor infers $w_i \geq w_j$, $\forall j \in idx0(q)$, $i \neq j$, unless $add(q, j)$ results in an unsatisfiable query. The rationale of this deduction is similar to the previous one. We assume that the user has extended the query by selecting a new query that includes an additional feature that brings a larger increase of his utility, compared to the other possible features that he may have included.

Trading one feature for two. If the current query results from a trade operation on the previous query, i.e., $q' = trade(q, i, j, k)$, the advisor may infer:

1. $w_j + w_k \geq w_i$,
2. $w_j + w_k \geq w_{j'} + w_{k'}, \forall j', k' \in idx0(q), \{j, k\} \neq \{j', k^{'}\}$ unless $trade(q, i, j', k')$ is unsatisfiable.

The first constraint says that the current query does not have a utility inferior to the previous one. While the second constraint says that the selected trade operation must obtain a utility that is not inferior to that of alternative trade operations that the user may have applied (and are satisfiable).

We note that unsatisfiable queries are never suggested, and therefore we never deduce that a query has a potential utility larger than that of a failing query. In the previous work [1], we called this "play safe" because we considered that the user might know that a query will fail and therefore he does not try it, hence we cannot assume that the potential utility of the query that was actually tried is larger than that of a query that the user did not try because he knew it would fail. In the current work we generalize and rephrase it by saying that the system can deduce only that the potential utility of the query that is tried is greater than or equal to the (potential) utility of the other queries that were suggested, or equivalently that the user could have tried (either because the system suggested them or because the user knows they are satisfiable).

4 Advisor

The advisor is the intelligent entity in charge of observing the interaction process, the user movements (queries issued), and making inferences on the user preferences. As mentioned before, the user preferences are not known at the beginning of the interaction between the user and the advisor. The advisor, after the user's first query, will generate a set of next candidates queries and will suggest only those with a utility that cannot be proved to be inferior to one of the other queries (undominated queries).

At each user-system interaction step, the advisor accumulates some constraints on the user utility function (see 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 needs to understand which queries are worth suggesting to the user. These are the queries having a utility not inferior to the utility of another query that may also be suggested. These queries are obtained by removing from C all the dominated queries.

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

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

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 query (that is suggested) for all the possible user utility functions that are compatible with the preferences induced by observing the user behavior. In this paper we solve this problem under the assumption that the user's true utility function is defined by one (unknown) vector among a finite set of weights vectors considered by the system. We call this finite set of all the possible utility function or "user profiles" $P = \{w^{(1)}, \ldots, w^{(m)}\}$. We will consider in the experiments m ranging from some dozens to hundreds.

With this assumption, having the set Φ we can prune from the set P the "incompatible profiles", i.e., those not satisfying the constraints 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 by the considered operators starting from the last issued query of the user. The final set of queries that are recommended are computed using a linear time procedure in the number of queries in C and utility functions in P', as follows:

1. A query $q \in C$, is labelled as dominated if and only if we can find another query $q' \in C$, $q' \neq q$, such that $\forall w \in P'$, $Utility_w(q') > Utility_w(q)$, i.e., $\sum_{i=1}^{n} w_i \times q'_i > \sum_{i=1}^{n} w_i \times q_i$.
2. Build the Advice set - 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 only the profiles $w^{(1)}$ and $w^{(3)}$ satisfy the constraints in Φ, so $w^{(2)}$ is an "incompatible profile", and must be pruned from P'. Table 1 shows the query utilities. $q^{(1)}$ 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 queries are removed from the set C. Notice that the remaining queries $q^{(1)}$ and $q^{(2)}$ do not dominate each other, thus they represent meaningful advice that the advisor can provide to the user.

Finally, the algorithm for query suggestions using a finite set of user profiles is described as follows:

```
1. $\Phi = \emptyset$, $P$ = all possible profiles, AdviceSet = all possible queries
2. Do
3.     Present AdviceSet to the user;
4.     currentQuery = query selected by the user in AdviceSet;
5.     Infer constraints analyzing the currentQuery and add them to $\Phi$;
6.     Remove incompatible profiles from $P$;
7.     Compute candidate queries;
8.     Remove dominated queries from candidate ones and generate AdviceSet;
9. while ((AdviceSet $\neq$ null) and (user wants advice))
```

The advisor presents to the user a possible set of queries. At the beginning these are all the possible ones, i.e., the user is free to enter the first query. Then the advisor infers the constraints 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 are generated from the current query, applying the operators mentioned in Section 2 and those that are not satisfiable are removed. Finally, the advisor identifies the AdviceSet by removing the dominated queries and suggests the remaining ones to the user as potential new moves. If the user selects one from this advice and the AdviceSet is not empty 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 suggest the products that satisfy the last query selected by the user.

5 Experiments

We performed some experiments in order to compare the performance of the proposed approach with the results obtained in [1]. We simulated several interactions between a user and the advisor. We varied the following parameters in the simulations: the product database and the number and format of the user profiles. Three different product databases were used, each one describing real hotels by their amenities expressed as Boolean features. Details of the product databases are given in the Table 2; here an hotel may have the same product description in terms of features as another, which is why the number of distinct products is smaller than the number of hotels.

We considered two kinds of user profiles as typical models of user preferences: "random-shape user profiles" and "exponential-shape user profiles". A "user's profile shape" refers to the distribution of the weights of the features in a user profile. Random-shape user profiles are created by first generating one initial user profile (weights vector) sampling the weights from a uniform distribution in [0,1]. Then the other profiles, in the same set P, are created by a random permutation of the feature weights of the initial user profile. Note that if the weights are sorted into decreasing order, the resulting sequence will decrease near linearly. This is because there is no special 'preference' for any number when you randomly select them. Conversely, the set of exponential-shape user profiles is created by generating first one initial user profile with an exponentially decreasing importance for the weights: $e^{-\alpha i}$, with a selected $\alpha \in [1, 4]$ and $i =$

Table 2. Product databases

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

$1, \ldots, n$. The other user profiles are again obtained with random permutations of the initial user profile. Here we wanted to simulate users with a few important features and many less important ones. For each experiment, we generated three sets of user profiles P: small (24 profiles), medium (120 profiles) and large (720 profiles). We wanted to observe the effect of the assumed variability of the user profiles on the user-advisor interaction length and the size of the advice set.

We assumed that the user is "Optimizing" [1], that is, one who confines his queries to the advice set provided by the advisor and he will always try the query with the highest utility in the advice set. The simulated interaction between a virtual user and the advisor is done considering the algorithm described in the previous section. One element of the set of predefined user profiles is randomly selected and considered as the user's true profile (user's utility). This is not revealed to the advisor, which interacts with the simulated user using the proposed methodology. The advisor deductions about the user's true utility function are based only on the observation of the user queries submitted at each interaction step. The initial query submitted by the simulated user is created in accordance to his true utility function; thus, the initial query includes up to the k most important features for the user.

In total, 18 experiments were performed corresponding to the combination of the variables mentioned before (product database, user profile shape and number of user profiles). In every experiment we ran 50 dialogues between a simulated user and the advisor. The observed measures were: the average number of queries issued per dialogue, the average size of the advice set and the average utility shortfall. The utility shortfall is the difference between the utility of the best query (selecting the product with the highest utility for the user) and the last query suggested by the system to the user. In this way we could measure if the system suggestions are close to the best query according to the user's true utility function.

Table 3 shows the values of the observed measures. We can observe that the average number of queries issued by the virtual user (interaction length) ranges between 3 and 7 almost independently from the "User Profile shape" and "User Profile set size". The interaction length seems to be related to the number of product features and the available products in the data set. The higher the number of product's features the longer will be the interaction. This happens because the user at each query editing step adds one feature to the previous query. In fact, the query suggestions are generated by the *add*() and *trade*() operations that extend the previous query by setting one additional feature to

Table 3. Averaged values of the observed measures for 50 runs in the 18 experiments performed.

Product D.B.	**User Prof. shape**	**User Prof. set size**	**Queries issued**	**Queries in Adv. set**	**Utility Shortfall**
Cork	Random	24	5.58	1.21	0.00281
		120	5.21	2.83	0.00491
		720	5.64	4.71	0.00622
	Exponential	24	5.66	1.28	0.0082
		120	5.52	3.05	0.0062
		720	5.31	3.37	0.0051
Marriott	Random	24	4	1.59	0
		120	4	3.41	0.00031
		720	4	4.33	0.00083
	Exponential	24	3.67	1.67	0.00636
		120	4	3.01	0.00097
		720	4	5.73	0
Trentino	Random	24	6.62	1.08	0.00315
		120	6.58	2.06	0.00319
		720	6.32	2.93	0.00757
	Exponential	24	6.38	1.14	0.00019
		120	6.18	1.72	0.00197
		720	6.48	2.89	0.00833

1. Hence, assuming that the best query has a certain number of features set to 1, then the user needs to pass through that number of steps (minus the number of features set to 1 in the initial query) in order to reach it, or to reach another query that does not provide the maximal utility but still cannot be further extended without reaching a failing query. Another factor to take into account is the number of products in the database. The smaller the number of products is, the more likely the process is to stop, because the current query cannot be further extended without building a failing query. The most important aspect of these values is that the interaction length is typically low and quite reasonable for real online applications.

The "Average size of the advice set" is sub-linearly correlated to the profile set size, that is, to the number of predefined user profiles. The higher the number of predefined user profiles, the (slightly) higher is the number of query suggestions in the advice set on average. In fact, if there are more user profiles, the more difficult it is to find dominated queries, thus the set of undominated queries (the advice set) is more likely to be larger. In general the average advice set size ranges between 1 and 6. This number of query suggestions represents an acceptable value for real applications. The "Average size of the advice set" doesn't seem to be related to the variables "User Profile shape" and "Product database". In

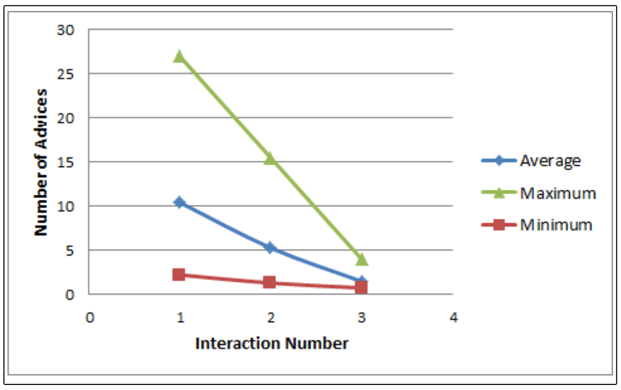

Fig. 1. Size of the Advice Set at different interaction steps.

general the “User profile shape” doesn’t seem to influence either the “Interaction length” or the “Average size of the advice set”.

The utility shortfall is very close to 0 on average. This cannot be 0 because the query suggestions are searched in a greedy way (always expanding the previous query), hence the advisor can fall into local maxima paths while searching for the best query suggestion [1]. Thus we cannot assure that the Advice Set will always contains the query with the largest utility that can be obtained by using the current query editing operations. Hence, limiting the query editing to the $add(\ldots)$ and $trade(\ldots)$ operators does not assure the user to reach the best query. Nevertheless at the end of the process the final query is very close to the best attainable given the user preferences.

Figure 1 shows the evolution of the Advice Set size (averaged over 50 dialogues) in the experiment that produces the highest number of average advices per suggestion (5.73 queries in table 3). That experiment corresponds to: Product Database = Marriott NY, Profile Shape = Exponential, and User Profile set size = 720. The curve labeled as “average” shows the average number of advices given to the user at the first three interaction steps. At the first step, the number of queries suggested is on average $10.4 \pm 8.2 (avg. \pm stdv.)$; at the next interaction step, it is 5.3 ± 3.9; and finally the system suggests only 1 ± 0.7 queries (the best). The curves labeled as “Maximum” and ”Minimum” correspond to the maximum and minimum number of queries suggested at each interaction step to the user. In general we can see that the number of advices falls quickly in a short number of interactions. Still, it is clear that there are certain dialogues with a rather large number of advices, and this is an issue to consider in the application of this technique.

We now compare our results with those presented in [1]. Table 4 shows the values of the variables “Average number of queries issued per Dialogue”, “Average size of the Advice Set”, “Average Utility shortfall” obtained in the previous work, where an infinite number of profiles was considered and the query dom-

Table 4. Comparison between the current (finite model) and previous work (infinite model) on the observed measures.

Database	Averaged measures	Infinite model	Finite model
Marriott-NY	Queries issued	4.67	3.58
	Numb. Advices	45.96	4.33
	Utility shortfall	0	0.0008
Cork	Queries issued	6.09	6.32
	Numb. Advices	69.88	2.93
	Utility shortfall	0	0.0075
Trentino	Queries issued	5.55	5.64
	Numb. Advices	59.02	4.71
	Utility shortfall	0	0.0062

inance relation was computed using linear programming techniques. It is clear that the average number of queries per dialogue is low in both approaches and very similar. This is due to the fact that the actual query editing operations are the same in the two approaches, and the dialogues converge to optimal queries with similar operations. The utility shortfall in the current approach is a bit larger than that measured previously. This is what one has to pay for the limiting assumption that the number of possible user utility functions (profiles) is finite. The major beneficial effect of the proposed approach is the significant reduction in the number of queries suggested by the advisor to the user by more than 10 times. This makes it much more suitable in real applications. Obviously this is again related to the assumption that the variability of the user utility functions is assumed to be smaller. We believe that in real scenarios approximating the set of all possible utility functions with a smaller, 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, making it feasible for the user to browse the advice set and pick up his best query.

6 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 a user searching for products in a catalogue. The products are described by Boolean features. They can be for instance tags or keywords found in the product descriptions. In this paper we assume that the user utility function is one among a finite set of possible functions that are known to the system, but the system does not know which is the true utility function of the user.

The results of our experiments showed that this assumption has a strong effect on the process of finding the best query suggestions that guide the user to the products that maximize his utility. In particular the number of user-advisor interaction steps (number of queries issued by the user) and the utility shortfall are low (as in our previous work where the user profiles were not limited to be finite). But, differently from the previous case, we have now observed a significant reduction in the number of advices provided at each user-advisor interaction step. We have also showed that having a good number of predefined user profiles is an important ingredient for improving the system performance and producing an effective support.

In future work we will consider the case when the true utility function of the user is not one of those assumed by the system. This is the true general situation when a totally unknown user is approaching the system and the system has no knowledge about his preferences. In particular, we will measure how this impacts on the utility shortfall. Additionally we will implement this approach on a real online and mobile application, which will undoubtedly help to give a better understanding of user behavior and the true effectiveness of the proposed approach.

References

1. 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.
2. M. H. Göker and C. A. Thomson. Personalized conversational case-based recommendation. In *Advances in case-based reasoning: 5th European workshop, EWCBR-2000, Trento, Italy, September 6–9, 2000: proceedings*, pages 99–111. Springer, 2000.
3. M. A. Hearst. *Search User Interfaces.* Cambridge University Press, 2009.
4. P. Lops, M. de Gemmis, and G. Semeraro. Content-based recommender systems: State of the art and trends. In F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors, *Recommender Systems Handbook*, pages 73–105. Springer Verlag, 2011.
5. T. Mahmood and F. Ricci. Learning and adaptivity in interactive recommender systems. In *ICEC '07: Proceedings of the ninth international conference on Electronic commerce*, pages 75–84, New York, NY, USA, 2007. ACM.
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. Q. N. Nguyen and F. Ricci. User preferences initialization and integration in critique-based mobile recommender systems. In *Proceedings of the 5th International Workshop on Artificial Intelligence in Mobile Systems, AIMS'04*, pages 71–78, Nottingham, UK, 2004.
8. F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors. *Recommender Systems Handbook.* Springer, 2011.
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. J. Zhang and P. Pu. A comparative study of compound critique generation in conversational recommender systems. In *Proceedings of the 4th International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems, AH 2006*, pages 234–243, 2006.