

Supporting Product Selection with Query Editing Recommendations

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

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

ABSTRACT

Consider a conversational product recommender system in which a user repeatedly edits and resubmits a query until she finds a product that she wants. We show how an advisor can: observe the user's actions; infer constraints on the user's utility function and add them to a user model; use the constraints to deduce which queries the user is likely to try next; and advise the user to avoid those that are unsatisfiable. We call this *information recommendation*. We give a detailed formulation of information recommendation for the case of products that are described by a set of Boolean features. Our experimental results show that if the user is given advice, the number of queries she needs to try before finding the product of highest utility is greatly reduced. We also show that an advisor that confines its advice to queries that the user model predicts are likely to be tried next will give shorter advice than one whose advice is unconstrained by the user model.

Categories and Subject Descriptors

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

General Terms

Human Factors

Keywords

recommender systems, user models

1. INTRODUCTION

Recommender systems are intelligent e-commerce applications that suggest products or services which best suit a user's needs and preferences, in a given situation and context [1, 2]. They have been successfully exploited for recommending travel services, books, CDs, financial services, insurance plans, news, and in many other application markets. From a technical point of view, recommender systems

emerged as supervised learning approaches using a data set of numerical ratings on products (e.g., from 1=bad to 5=excellent), expressed by a collection of users on a catalogue of products, to make a prediction for products not yet rated by the target user, i.e., the user for whom a recommendation is sought. Prediction algorithms include collaborative-filtering, content-based filtering and case-based reasoning [1, 2, 3]. However, these classical approaches typically support a simple human-computer interaction model, which basically first collects user related information (ratings or product preferences) and then exploits the background knowledge to make ratings' predictions and derive product recommendations.

More recently, a number of *conversational* approaches have been proposed. In conversational recommender systems the advisor not only suggests and ranks products, it also guides the user by asking for more information about her preferences or providing information about the product and the search process, e.g., explaining the rationale of the ranking or explaining the failure of a search initiated by the user [8, 5, 3, 9, 10, 11, 7]. The user may not know, or may not be aware, of all her preferences at the start of the interaction. Preferences are revealed, or even constructed, during the interaction as the product space is explored. By contrast, the user of a non-conversational ('single-shot') recommender is expected to be able to articulate all preferences up-front.

In this paper, we introduce the idea of *information recommendation*, bringing the concept of conversational recommender systems to its more radical interpretation. A conversational recommender can use information recommendation to suggest actions that help the user to efficiently search for products, as well as using product recommendation to suggest products that the user may like.

Consider a conversational product recommender system in which a user repeatedly edits and resubmits a query until she finds a product that she wants [9, 5]. An advisor can infer constraints on the user's utility function by observing the user's actions. For instance, the system might infer that the features constrained by the user query are more important, for the user, than the features not yet used in the query. The advisor can add these constraints to a particular kind of dynamic user model [4, 8]. The advisor can then use the constraints to deduce new preference relations between product features and ultimately to rank the next possible search actions of the user, rather than, or as well as, using the preference relations to select and rank products in the catalogue. In other words, in information recommendation, the advisor uses the user model to determine the *feasibility*

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

RecSys '07, October 19–20, 2007, Minneapolis, Minnesota, USA.
Copyright 2007 ACM 978-1-59593-730-8/07/0010 ...\$5.00.

of actions (the *satisfiability* of queries in this case) that it thinks the user is *more likely* to try. It can then advise the user of which to try or which to avoid.

In the rest of this section we compare the use that information recommendation makes of its user model with the use that product recommendation makes. In the Adaptive Place Advisor, for example, the user model is used for product selection [11]. The advisor infers feature weights and defaults, and uses them in product retrieval. Similarly, in the work of Pu *et al.*, the system uses its knowledge of users and of the product space to select sets of products, albeit sets that it hopes will provoke the user into volunteering further preferences [7]. In collaborative filters also, the user model (the ratings profile) is for product retrieval and ranking.

But in information recommendation, the user model is used to guide the user's search rather than to retrieve or rank products. Work on question selection in dynamic dialogues can be seen as an example of information recommendation. The system dynamically selects questions to elicit user preferences. Its goal is to choose a sequence of questions that most effectively homes in on desirable products. In most such work, questions are selected based on the user's partial query and the product distribution. But, Schmitt's *simVar* system also builds and uses simple user models too [10]. Reilly *et al.*'s use of the query history to dynamically recommend compound critiques can also be regarded as information recommendation [8]. In their work, the system shows the user some products which the user can critique, but it also advises the user by displaying dynamically-computed critiques that are known to be satisfiable, which the user can select.

Section 2 describes our assumptions about how products and queries are represented; it describes users' utility functions; and it describes the idea of a conversational product recommender in which the user repeatedly edits and resubmits her query in a search for the products of highest utility. The section also explains the assumptions we make about user rationality. Section 3 explains how the advisor can infer the constraints that it adds to the user model and it explains different strategies for giving advice to the user. Section 4 presents our experimental methodology and results.

2. SEARCHING FOR PRODUCTS

2.1 Products and Queries

We assume a very simple product data model. Products are described by a fixed number of n Boolean features. For instance, hotels may have: a sauna, a pool, parking, etc. Hence, each product can be represented by a fixed-length string of bits, $p = p_1, \dots, p_n$, where $p_i = 1$ means that the product has the i th feature and $p_i = 0$ means that it does not have the feature.

User queries are defined by a product pattern $q = q_1, \dots, q_n$, where q_i is either 1 or 0, $i = 1, \dots, n$. If $q_i = 1$, the user is interested in products that have the i th feature; if $q_i = 0$, the user has not (yet) declared any special interest in the i th feature. It must be kept in mind that, e.g., $q = 1010$ does not mean that the user wants to see products that lack the second and fourth features; it simply means that the user wants a product that has the first and the third features.

Given query q , the query engine retrieves products P where if $q_i = 1$ then $p_i = 1$, for all $p_i \in P$. In other words, each retrieved product must possess at least all the features

that are explicitly requested in the query, but may possess other features too. For example, $q = 1010$ is matched by products such as 1011 and 1111 as well as 1010; it is not matched by products such as 0010. We describe a query as *satisfiable* if it is matched by at least one product; otherwise, we call it *unsatisfiable*. Given query q , we expect the query engine to at least tell the user whether q is satisfiable or not.

2.2 Utility

We assume the user has a fixed utility function. The utility of product $p = p_1, \dots, p_n$ is defined as follows:

$$U(p) = w_1 p_1 + \dots + w_n p_n$$

where (w_1, \dots, w_n) is a vector of weights. We assume only that the weights are non-negative and do not exceed 1 ($0 \leq w_i \leq 1$) and that there is at least one non-zero weight ($\sum w_i > 0$). The weight of a feature says how strong the desire of the user for that feature is. If a weight w_i is zero, 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; if $w_i = w_j$ ($i \neq j$), then the user is indifferent between the i th and j th features. We assume that the goal of the user is to find a product that maximises the utility function.

We can also define the utility of a query $q = q_1, \dots, q_n$. In fact, we define two types of query utility. The *potential utility* of query q is given by

$$U(q) = w_1 q_1 + \dots + w_n q_n$$

Hence, the potential utility of a query is the utility to the user of a product that offers *exactly* the features requested in the query, whether such products exist or not.

The *actual utility* of query q is given by

$$V(q) = \begin{cases} 0 & \text{if } q \text{ is unsatisfiable} \\ U(q) & \text{otherwise.} \end{cases}$$

Hence, the actual utility of a query is zero if no product matches the query; otherwise, the actual utility equals the potential utility.

2.3 Utility Gain

In the kind of conversational recommender that we are considering in this work, the user is engaged in an interactive, incremental search process. The states of the search space are different queries. The successors of state (or query) q , $\text{succ}(q)$, will be the queries that the user obtains by editing q . $\text{succ}(q)$ will depend on the actions that the user interface makes available, e.g., it might allow the user to add a feature to a query, to delete a feature from a query, to restart, etc. The user's goal in query editing is to move to a state with higher actual utility.

We assume a certain rationality in user behaviour. A minimal requirement is that a user will not search for products that are less preferable than those already selected:

AXIOM 1. *If q is the user's current query and q' is a successor query, $q' \in \text{succ}(q)$, then the user will try q' if and only if $U(q') \geq U(q)$, and may choose to accept q' if and only if $V(q') \neq 0$.*

In other words, we assume that the user will only *try* q' (i.e. issue the query to the query engine) if its *potential* utility is greater than or equal to q 's; and then the user may choose to *accept* q' (i.e. move to this state) if q' has higher

or equal *actual* utility (i.e. if the query engine reports that q' is satisfiable). A rational user will satisfy a second axiom:

AXIOM 2. *If q is the user's current query and the user is contemplating a set $Q \subseteq \text{succ}(q)$ of successor queries each of whose potential utility is greater than or equal to q 's ($U(q') \geq U(q)$ for $q' \in Q$), then the user will try a member of $Q^* \subseteq Q$, where Q^* contains those members of Q that have maximal potential utility:*

$$Q^* = \{q' \in Q : \forall q'' \in Q, U(q') \geq U(q'')\}$$

In this axiom we are saying that the user will choose to try one of the queries that has maximal potential utility from those successor queries Q that the user is contemplating. Of course, this leaves open the question of which successor queries are in the 'contemplation set', Q , at any point in the user's search. This will depend on the user's background knowledge, what she has learned so far during the interaction and her cognitive resources (e.g. her memory and reasoning capabilities).

In fact, an advisory system may not know exactly how the user computes her contemplation set at any point in the interaction. But, it can infer user preferences by observing the queries the user tries. If the user tries query q' , the system may infer that $U(q') \geq U(q'')$ for all $q'' \in Q'$, where Q' is the set that the system assumes the user is contemplating. But if the set the user actually contemplated is a proper subset of Q' then the system may make wrong inferences: it may incorrectly infer $U(q') \geq U(q'')$ for all $q'' \in Q'$. Hence, in the following we will make some assumptions about the nature of the user's contemplation set and, from the queries the user issues, we will derive constraints on the definition of the user's utility function.

2.4 Query Editing

As we noted above the contemplation set is a subset of the successor queries that the user interface make available. Hence, following the above discussion, one way to keep the contemplation set small, thereby making it less likely that the system makes incorrect assumptions, is to keep also $\text{succ}(q)$ as small as possible by offering only a small number of easily-understood editing operations. Since real user behaviour in query editing processes tends to proceed with minimal modifications, restricting moves to ones of limited 'reach' will not impose a true limitation on real users.

Hence, here we assume that just three editing operations are available:

Op1 — Add a feature This means changing a 0 in q to a 1. This corresponds to requesting a feature that is additional to the ones in q . We will write $Op1(q, i)$ to mean that the user adds feature i to query q .

Op2 — Switch a feature for another This means simultaneously changing a 1 in q to 0 and a 0 to 1. This operation corresponds to switching one feature for another. We will write $Op2(q, i, j)$ to mean that the user discards feature i and introduces feature j .

Op3 — Trade a feature for two features This means simultaneously changing a 1 in q to 0 and two 0s to 1s. This operation corresponds to making a compromise by losing one feature for two others. We will write $Op3(q, i, j, k)$ to mean that the user discards feature i and introduces features j and k .

We now discuss the rationale for considering only these three moves.

The first move is quite obvious: given non-negative weights, an *Op1* move will never decrease the *potential* utility. In fact, if $q' = Op1(q, i)$, then the gain in potential utility is:

$$U(q') - U(q) = w_i$$

We note that *Op1* moves may not change the *actual* utility by this amount since the new query might be unsatisfiable, in which case actual utility will fall from $U(q)$ to 0.

Consider now the second move. If $q' = Op2(q, i, j)$, the effect on utility is

$$U(q') - U(q) = -w_i + w_j$$

In general, this sum can be negative. But Axiom 1 says that a user will try this editing operation only if $-w_i + w_j \geq 0$, i.e. if there is no loss in potential utility.

In the case of the third move, if $q' = Op3(q, i, j, k)$, this has the following effect on the utility:

$$U(q') - U(q) = -w_i + w_j + w_k$$

In general this sum can also be negative. But Axiom 1 says that a user will try this editing operation only if $-w_i + w_j + w_k \geq 0$, i.e. if the gain in potential utility brought by the two features added is greater than or equal to the loss in potential utility due to the feature removed.

We do not consider the move of deleting a feature from the current query since this always has a zero or negative effect on the potential utility. In principle, we could also allow the user to apply more complex transformations, for example discarding two features while adding three new features. But, as explained above, restricting the available editing operations helps both the user and the system.

It is worth noting that there are situations where an *Op3* move can be preferred to an *Op1* move. If for instance the current query q is 100 and the user knows that both $Op1(q, 2) = 110$ and $Op1(q, 3) = 101$ are unsatisfiable but $Op3(q, 1, 2, 3) = 011$ is not known to be unsatisfiable, then the *Op3* move is the best option for the user. It is also worth noting that there are cases where an *Op2* move is useful, even in the presence of the *Op1* and *Op3* operations. For lack of space we shall not describe here an example of this situation.

It might then be thought that we could restrict our attention to a system that offers only *Op1* and *Op2* edits, since an *Op3* edit can be composed of an *Op1* and an *Op2*: e.g. $Op3(q, i, j, k) = Op3(Op2(q, i, j), k)$. This is true but the *Op2* move required to obtain the given *Op3* move may not be rational, i.e. it may decrease the potential utility.

A final observation is that with these three edit operations the user is not guaranteed to reach the global optimum state. For lack of space we shall not describe an example. But we note, happily, that such examples are quite contrived and therefore unlikely to arise in practice for rational users.

2.5 Rationality Implications

Assumptions about the 'rationality' of the user (Axioms 1 and 2) allow an observer to deduce knowledge about that user's utility function. In this section, we will focus on this issue. First, let us introduce some notation. If $q = q_1, \dots, q_n$ is a query and i is an index ($1 \leq i \leq n$) and $q_i = 0$, then we shall denote with $q + i$ the query obtained by setting q_i to 1; if $q_i = 1$, then we shall denote with $q - i$ the query obtained

by setting q_i to 0. Further, let $idx0(q) = \{i : q_i = 0\}$ and $idx1(q) = \{i : q_i = 1\}$, the indexes of the bits in q that are 0 and 1, respectively.

PROPOSITION 1. *Assume that the user tries an initial query q . For all $i \in idx1(q)$ and $j \in idx0(q)$, then $w_i \geq w_j$ unless $V(q - i + j) = 0$.*

Proposition 1 says that the user will not add to her initial query a feature i that is less preferable than another feature j that is not included in the query, unless the query that contains j and not i is unsatisfiable. This proposition derives from Axiom 2 and the observation that the user could have chosen the query $q - i + j$ instead of q , hence it must be the case that $U(q - i + j) \leq U(q)$, and this is true if and only if $w_i \geq w_j$. Here we are ‘playing safe’ by assuming that if $V(q - i + j) = 0$, then the user knows it. Obviously the user may not know all these unsatisfiable queries and, from the initial query, we could derive more inequalities. For instance, we could simply assume that all the features included in the query are preferred to the features not included, i.e. $w_i \geq w_j$ for all $i \in idx1(q), j \in idx0(q)$. But this is not safe because in fact the user may have background knowledge that some feature combinations are not possible and therefore she has not tried these combinations in her initial query.

To illustrate this issue, consider the following simple example. Let us imagine that the user knows that there is no golf course in the centre of the city to which she is travelling. Suppose hotels are described by three features, and in decreasing order of preference for this user they are *centre*, *golf* and *parking*. Then from an initial query in which *centre* = 1, *golf* = 0 and *parking* = 1, an observer would be wrong to infer that *parking* is preferred to *golf*; the observer would be wrong because the switch from *parking* to *golf* is not satisfiable.

PROPOSITION 2. *If the user tries to add a feature i to the current query q , i.e. she applies $Op1(q, i)$, then $w_i \geq w_j$ for all $j \in idx0(q)$ $i \neq j$, unless $V(q + j) = 0$.*

Proposition 2 says that the user will not have added a feature whose additional utility is lower than that of another feature also not yet in the query, unless she knows that adding the feature with greater additional utility would result in an unsatisfiable query. There are similar motivations to that mentioned in the previous case for ‘playing safe’ by assuming that the user knows which are the unsatisfiable queries.

PROPOSITION 3. *If the user tries to switch feature i to feature j , i.e. she applies $Op2(q, i, j)$, then $w_i \leq w_j$; and $w_j \geq w_k$ for all $k \in idx0(q)$ $k \neq j$ unless $V(q - i + k) = 0$.*

Proposition 3 says that the user will not switch a feature for one that has inferior preference; and the feature to which the user is switching must not have utility inferior to any other feature not constrained in the query, unless this alternative query is unsatisfiable.

PROPOSITION 4. *If the user tries to trade feature i for features j and k , i.e. she applies $Op3(q, i, j, k)$, then $w_i \leq w_j + w_k$; and for all $j', k' \in idx0(q)$ such that $\{j, k\} \neq \{j', k'\}$, then $w_j + w_k \geq w_{j'} + w_{k'}$ unless $V(q - i + j' + k') = 0$.*

Finally, Proposition 4 says that the user will not discard a feature for two features having inferior sum of preference;

and the two features added must not have sum of utility inferior to two other features not constrained in the query, unless this alternative query is unsatisfiable.

Note how several of these propositions assume only a limited amount of reasoning capability of the user. In fact, if the user is more fully rational, we could deduce more. For example, in the case of Proposition 4, we could also deduce that the selected $Op3(q, i, j, k)$ is better than all the other satisfiable $Op3(q, i, j', k')$ moves, i.e. if the user applies $Op3(q, i, j, k)$, then for all $i' \in idx1(q)$ and $j', k' \in idx0(q)$ such that $\{i, j, k\} \neq \{i', j', k'\}$, $-w_i + w_j + w_k \geq -w_{i'} + w_{j'} + w_{k'}$, unless $V(q - i' + j' + k') = 0$.

Using the same reasoning we can also infer that a selected $Op2$ move is better than or equal to all the satisfiable $Op1$ moves, and a selected $Op3$ move is better than or equal to all the satisfiable $Op1$ and $Op2$ moves.

These are what we might infer if we attribute ever greater rationality to the user. In this paper, we are not going to attribute these higher levels of rationality to the user, and hence we will infer only what is stated in Propositions 1–4. An important observation is that it is not a problem to our proposed methods if we assume that users are less rational than they really are. The reason this does not pose a problem to our proposed methods is that assuming users are less rational than they really are will result only in us making fewer deductions when observing their moves; it will not result in us drawing incorrect inferences. In fact, it is more dangerous to assume a fully rational user, who can really take the best move, since this will cause us to draw inferences that may be incorrect.

3. INFORMATION RECOMMENDATION

3.1 The Advisor

We now describe the help that an advisory system can provide to the user to let her more effectively find the products that have maximal actual utility.

If the system could somehow know the user’s utility function then it could use a simple search procedure to identify the product(s) with highest actual utility. But in our work, we are assuming that *the system does not know the user’s utility function and it is not going to ask the user about feature preferences (weights)*. Instead, the advisor *infers* constraints on the user’s utility function by observing the user’s moves, and adds these constraints to a user model.

In addition to telling the user the findings of the query engine, i.e. how many products, or which products if any, match the current query, the advisor is also capable of telling the user about the satisfiability of related queries (especially those that it thinks that the user is likely to try). We note that in contrast with *query relaxation*, e.g., [5, 6], where the goal is to *repair* an unsatisfiable query (especially in non-conversational recommenders), our focus is on methods that support the user in *avoiding* the generation of unsatisfiable queries in a conversational setting.

Imagine for instance a user whose current query requests hotels that have air-con and golf, and the advisor knows that, while the user’s query is satisfiable, there are no such hotels located in the city centre but there are some that also have swimming pools. In this situation the system could help the user in the search process by supplying exactly this information: this will dissuade the user from fruitlessly trying to add to her query a request for a city centre location,

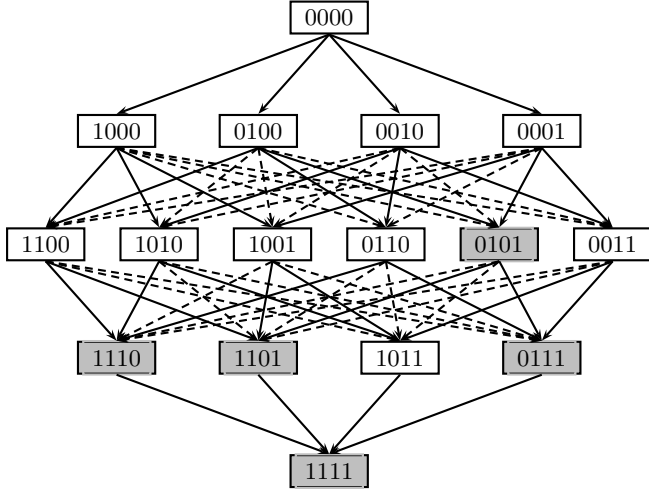


Figure 1: Example of all queries on four Boolean attributes; shaded queries are ones we are taking to be unsatisfiable in this example. *Op1* moves are shown as solid lines; *Op3* moves are shown as dashed lines. *Op2* moves are not allowed in this example.

and it will enable her to realise that she can satisfy a desire for a pool. Moreover, if it is satisfiable to have air-con, golf and a sauna but the advisor has inferred from earlier actions that a sauna is less preferable than a pool, then it is better to tell the user that there are no hotels with air-con and golf in the centre but there are some that have a pool rather than to tell her that there are no hotels with air-con and golf in the centre but there are some that have a sauna.

Obviously the advisor could tell the user the satisfiability of all the possible moves, but such a list would not be of much use, as it could potentially be very long. Hence the advisor must put in place methods to minimise the quantity of information that is passed to the user during their interaction. In other words, the advisor must pass to the user the information that has the greatest value for the user.

The information that has the greatest value is here considered to be that which minimises the total quantity of information exchanged and the interaction length, while still finding the product(s) having maximal actual utility.

3.2 Example

In this section we illustrate with an example the typical human-computer interaction we want to support. Figure 1 depicts all possible queries on four Boolean features, i.e. all 16 possible feature combinations. The shaded boxes represent queries we will take to be unsatisfiable in this example, i.e. no products match those queries. The arrows represent *Op1* and *Op3* editing operations. To simplify the example and to keep it relatively concise, we are going to assume that the query engine does not support *Op2* editing operations.

Let us assume that the user has issued an initial query $q = 1100$. This means that she is looking for a product that has the first two features, and we can see that this is satisfiable.

From this query the advisor infers that $w_1 \geq w_3$, $w_2 \geq w_3$,

and $w_2 \geq w_4$, but it does not infer that $w_1 \geq w_4$ because switching the first and fourth features produces an unsatisfiable query. As we have discussed previously, the advisor ‘plays it safe’, in case the user somehow already knew that 0101 was unsatisfiable. It also cannot deduce anything about the relationship between w_1 and w_2 because they have been introduced simultaneously in the same initial query.

The advisor knows that 1110, 1101 and 1111 have a higher utility for the user (if the weight of the added feature is non-zero) or the same utility (if the weight of the added feature is zero) but they are all unsatisfiable. Conversely, the advisor is still uncertain whether 1011 and 0111 have a higher, equal or lower utility than that of 1100, although it does know that 0111 is unsatisfiable.

In this situation, the advisor can, for instance, take a conservative approach and tell the user about the satisfiability or unsatisfiability of each of the queries that can be obtained by a single *Op1* or *Op3* operation applied to 1100. In this case, it would tell the user that 1110, 1101 and 0111 are unsatisfiable but 1011 is satisfiable.

Then there follow three possibilities. If for this user in fact $w_2 < w_3 + w_4$, then the user at the next step can immediately find the optimal solution, i.e. she can decide to edit using *Op3* and retrieve the products satisfying 1011; if for this user in fact $w_2 > w_3 + w_4$, then the user will know she is already at an optimal solution; and if for this user $w_2 = w_3 + w_4$, then she must try the *Op3* edit in case subsequent edits are available that can increase utility. Hence in the best case she will do zero moves and in the worst case one. Unfortunately the advisor, at this point, cannot know whether $w_2 < w_3 + w_4$: for some users it will be true and for others it will not.

On the other hand, the advisor could tell the user only the unsatisfiable next queries (1110, 1101 and 0111).

For the user, the same reasoning applies. If for the user $w_2 < w_3 + w_4$, then she will use *Op3* to move to 1011; if $w_2 > w_3 + w_4$, she can stay in 1100; if $w_2 = w_3 + w_4$, she must try the edit to see whether it brings opportunities to subsequently improve utility. Hence, this is a better behaviour for the advisor: it reveals information about only three queries (one fewer than before), yet there is no extension of the interaction in both the worst and best cases.

Suppose instead the advisor tells only the unsatisfiability of the *Op1* moves (i.e. that 1110 and 1101 are unsatisfiable). Then the user may require (in the worst case) two moves, i.e. a move to 0111 (with an *Op3* move), only to learn that it is unsatisfiable, and then the alternative move to 1011 (also with *Op3*). The best case is always 0 moves.

Finally, suppose the advisor (with an even more incomplete strategy) were to tell of the unsatisfiability of just the second *Op1* move (i.e. that 1101 is unsatisfiable, without mentioning the unsatisfiability of 1110). Then the user will always try 1110 since, without the knowledge that 1110 is unsatisfiable, it has a utility no worse than the two *Op3* moves for the user. In the worst case in this scenario, the user will need three moves (1110, 0111 and 1011) to get the optimal query. A similar situation occurs if the system reveals only that 1110 is unsatisfiable.

In this simple example, we have seen that there is a trade-off between the quantity of information revealed by the advisor and the length of the interaction. Moreover, we have seen that there are actions made by the advisor that are not minimal, in the sense that revealing less information will result in the same interaction length.

3.3 User Models and Advice

Every user model can contain the facts that weights are assumed to be non-negative and to not exceed 1: $0 \leq w_i \leq 1$. And every user model can contain the fact that at least one weight is non-zero: $\sum w_i > 0$. Beyond this, the advisor will infer the constraints that it can deduce according to Propositions 1–4. Let us denote the contents of a user model, i.e. the set of constraints the advisor has collected at a certain point in the interaction, by C .

We assume that, after each query submitted by the user, the query engine tells the user whether the query is satisfiable or not. And, after each satisfiable query, the advisor tells the user whether certain (related) queries that the user has not yet tried but is likely to try would be satisfiable or not. Note that we assume that the advisor only needs to give advice when the user’s query is satisfiable as only in this case can it be further extended (unsatisfiable queries will not change the current query).

The goal is to identify, at each step of the interaction, the queries whose (un)satisfiability the advisor should report to the user such that the user will find the products that maximise her utility function in a process with minimal cost, where the cost of the process is a monotone function of the number of queries tried and the number of query (un)satisfiabilities revealed by the advisor to the user.

Given a user model (a set of inequalities) C , we can compute (using linear programming techniques) a partial order \geq_C on a set of queries Q , where for each $q, q' \in Q$, $q \geq_C q'$ iff we can derive from C that $U(q) \geq U(q')$. Let us denote by $Best(C, Q)$ the set of queries that are maximal in Q according to the partial order defined by C . We have the following:

PROPOSITION 5. *If the current query is q and the user model at this point is C , the next query that the user will try is a member of $Best(C, succ(q))$*

where $succ(q)$ as before is the set of queries that can be generated by modifying q with a single *Op1*, *Op2* or *Op3* operation. This proposition says that the partial order that the advisor builds from the user model is compatible with the user’s utility function, and that the best move, i.e. the one maximising the potential utility, is among those that are reachable by *Op1*, *Op2* or *Op3* and that are maximal according to this partial order.

Hence, we expect the user to try a move that belongs to $Best(C, succ(q))$. Actually this is a superset of the moves that the user will consider since the information that the system has derived from the interaction can only partially reveal the user’s utility function. This is one reason why the system can only deduce a partial order on the queries, whereas the user will have a set of one or more equally good best next actions.

We can decompose $Best(C, succ(q))$ according to two orthogonal distinctions. First, there are three types of queries in $Best(C, succ(q))$, i.e. *Op1*, *Op2* and *Op3* moves. We will denote these by $Best(C, succ_1(q))$, $Best(C, succ_2(q))$ and $Best(C, succ_3(q))$, respectively. Second, the advisor will know that some of the queries are satisfiable and some are unsatisfiable. We will denote these by $Best(C, succ(q))^+$ and $Best(C, succ(q))^-$, respectively.

Hence, the advisor has $3 \times 2 = 6$ types of information that it can reveal, and it must decide what to tell the user given a query history. If the advisor provides only a subset

Table 1: Three databases of hotels

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

of the best next moves, then the user may still try one of the moves that she was not told about and it may prove to be unsatisfiable.

One rational strategy, which we call the *Complete Recommendation on Maxima Strategy*, is to reveal either $Best(C, succ(q))^-$, i.e. all moves that are unsatisfiable (and advise the user to avoid them), or $Best(C, succ(q))^+$, i.e. all moves that are satisfiable (and advise the user to confine her attention to them). To minimise the amount of advice which is given we assume this strategy involves telling the user about the smaller of the two sets.

Another strategy, which we call the *Incomplete Recommendation on Maxima Strategy*, is to tell the user about *Op1* moves (again choosing to tell the user either about the satisfiability of $Best(C, succ_1(q))^+$ or the unsatisfiability of $Best(C, succ_1(q))^-$, whichever is the smaller set) and only if there is no satisfiable *Op1* move to provide advice about the *Op2* and *Op3* moves (again, the satisfiable queries or unsatisfiable queries, whichever is the smaller set). This is clearly an incomplete strategy since there might be an *Op3* move with a higher utility gain than the *Op1* moves suggested.

4. EXPERIMENTS

4.1 Strategies, Products and Users

We have evaluated five information recommendation strategies in a simulation. Two of the five are the Complete and Incomplete Recommendation on Maxima Strategies described in the previous section. We also have three ‘baseline’ strategies. The first is Null Recommendation, i.e. providing no advice at all. The other two are based on the Complete and Incomplete Strategies but they use $succ(q)$ in place of $Best(C, succ(q))$, i.e. they form their advice from all next possible moves, hence they *generate longer advice*: their advice is not confined to queries that, according to the user model, will be *favoured* by the user. By comparing the two strategies that use the maxima with the ones that use all next moves, we can see how much shorter the advice is when the advisor exploits a user model.

We use three separate product databases, each describing hotels by their amenities expressed as Boolean features such as *airport shuttle*, *pets permitted*, *restaurant on-site*, etc. Details of the product databases are given in Table 1. Many hotels offer the same amenities, which explains the difference between the number of (physical) hotels and, from an amenities point of view, the number of (*distinct*) products.

Our evaluation uses three different types of simulated user: *optimizing*, *prioritizing* and *random*. They have several things in common. They do not try queries that they have tried before; they only try *Op1*, *Op2* and *Op3* moves that do not lessen potential utility; and they take heed of all advice given, i.e. if the advisor tells them to confine their queries to a certain set or to avoid queries in a certain set, then they do so. They differ in the way they select from moves

that remain. An *optimising* user chooses the best possible move from the set, as per Axiom 2. In the event of ties (more than one equally good best move), the user uses random tie-breaking. A *prioritising* user has a preference for *Op1* moves. It will first try to use *Op1* to add the most useful feature to its current query (again with random tie-breaking). If no satisfiable move results from adding each most useful feature, its next moves will be to try to add each second most preferred additional feature (with random tie-breaking). Only when all *Op1* moves have been exhausted (in decreasing order of usefulness) will it try *Op2* or *Op3* moves, in descending order of potential utility gain. A *random* user chooses its next move randomly.

In the experiments, we pair each of the three user types with each of the five advisors. In pairings which include random users with advisors that use user models, the advisor will assume greater rationality of the user than the user actually exhibits. In such pairings, the inferences that the advisor draws may be incorrect. Worse, the user model may become inconsistent, at which point we take $Best(C, succ(q))$ to be equal to $succ(q)$, i.e. the advice degenerates to that which would be given with no user model at all.

4.2 Results

For each product database, we randomly generate 50 weight vectors (where weights are randomly-chosen real numbers to one decimal place in $[0, 1]$) and a satisfiable initial query compatible with each weight vector. The use of a set of random queries gives us fair coverage of the query space, but it would be useful in the future to incorporate a bias towards the kinds of queries and weight vectors that real users exhibit.

For each user-advisor pairing, we ran 50 dialogues starting from each of the initial queries. The results, in Table 2, show the following, averaged over the 50 interactions:

- A) The number of queries the user tries;
- B) the number of queries the user tries that are successful, i.e. satisfiable (and this is also the number of times advice is given since advice is given when the user moves to a new query and the user moves to a new query when a query it tries is satisfiable);
- C) the total number of queries whose (un)satisfiability is reported by the advisor during the interaction (i.e. individual items within the advice); and
- D) the amount of utility, if any, by which the final query that the user reaches falls short of the product of highest utility that can be reached from the initial query.

We will focus to begin with on Optimising users (Opt) and Complete on Maxima strategies (CoM). We can clearly see the advantage of giving advice. Compare the figures for Null advisors with the rest. The number of queries that the user must try on average (column (A)) is substantially reduced if the user receives advice: from 21.24 to 4.67 (Marriott-NY); from 37.14 to 5.55 (Cork); and from 16.12 to 6.09 (Trentino-10). Furthermore, with Null advisors the majority of queries the user tries are unsatisfiable (column (B)).

We can also clearly see the advantage of using a user model to restrict the advice to the moves that we infer the user will try next. Compare the figures for advisors that base

their advice on all next moves (CoAN) with those that compute the maxima (CoM). The number of queries in the advice (column (C)) is substantially reduced when the maxima is used: from 75.36 to 45.96 (Marriott-NY); from 96.84 to 59.02 (Cork); and from 109.20 to 69.88 (Trentino-10).

A similar story can be told for the incomplete strategies. But, furthermore, the number of queries within the advice given by incomplete strategies is much lower (and more reasonable) than that given by corresponding complete strategies. Consider, for example, the Complete and Incomplete on Maxima strategies (CoM and IoM): from 45.96 to 5.14 (Marriott-NY); from 59.02 to 8.51 (Cork); and from 69.88 to 6.27 (Trentino-10). And this comes at only fairly modest increases in the number of queries that the user must try: from 4.67 to 5.98 (Marriott-NY); from 5.55 to 12.13 (Cork); and from 6.09 to 9.36 (Trentino-10). Of course, some of these extra queries are unsatisfiable (column (B)).

Finally, let's look across the three types of users. Unsurprisingly, Random users suffer longer interactions (column (A)) and may fail to find the product of highest utility, although the average shortfalls in utility (column (D)) are small. But Optimising users do not gain substantially over Prioritising users. The latter prefer *Op1* moves, and this could lead to suboptimal behaviour. But in practise we are not seeing much of this on these datasets. It turns out (observing detailed simulation logs) that rarely are *Op2* and *Op3* moves tried (either because they are not satisfiable or because they worsen utility). This helps explain why, at least for these hotel databases, prioritising users and incomplete strategies fare no worse than optimising users and complete strategies.

We might conclude that the Incomplete on Maxima strategy strikes the best balance: interactions are not much longer than interactions with the corresponding Complete strategy, and the amount of advice it gives is much lower.

5. CONCLUSIONS AND FUTURE WORK

We have described information recommendation, a process by which an advisor observes the actions of the user of, for example, a conversational product recommender system; builds a user model, in this case, by inferring constraints on the user's utility function; and then gives advice about actions it thinks the user is likely to try. We have given a detailed formulation of this in the case where products and queries are described by sets of Boolean features.

Our experimental results show that information recommendation can dramatically reduce interaction length and that the length of the advice given is much shorter when the advice is constrained by the user model than when it is not. For our hotel data, an incomplete advice strategy strikes a good balance between amount of advice and length of interaction.

Efficiency is an issue that requires further investigation. The cost of finding which successor queries are unsatisfiable is polynomial. But the major computational cost in our approach is, given a query q , deciding whether, according to the user model that we have built so far, each successor query q' has larger potential utility than each other successor q'' . We are currently using linear programming techniques in deciding this, but we plan to compare with alternative techniques such as constraint satisfaction.

There remain many other avenues to explore, including: inferring richer user models and further considering how to

handle the case where users are less rational than the advisor assumes; allowing user models to be carried over from one dialogue to another, or from one user to another, with the risk that they incorrectly characterise the user's preferences; and testing the ideas with real users. We also need to extend the approach to product databases where features are multi-valued. Formally, this can be done by considering each feature-value to be a new Boolean feature, and then applying the methods we have described in this paper. In some domains, this might even be a practicable approach; in other domains, it will result in too many Boolean features and unreasonably long advice. For these domains, we may need to develop a more concise formalism, and ways of generalising the advice to make it shorter.

6. REFERENCES

- [1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, 2005.
- [2] S. S. Anand and B. Mobasher. Intelligent techniques for web personalization. In S. S. Anand and B. Mobasher, editors, *Intelligent Techniques for Web Personalization*, pages 1–36. Springer, 2005.
- [3] D. Bridge, M. Göker, L. McGinty, and B. Smyth. Case-based recommender systems. *The Knowledge Engineering review*, 20(3):315–320, 2006.
- [4] G. Fisher. User modeling in human-computer interaction. *User Modeling and User-Adapted Interaction*, 11:65–86, 2001.
- [5] D. McSherry. Retrieval failure and recovery in recommender systems. *Artificial Intelligence Review*, 24(3-4):319–338, 2005.
- [6] N. Mirzadeh, F. Ricci, and M. Bansal. Supporting user query relaxation in a recommender system. In *Procs. of the 5th International Conference on Electronic Commerce and Web Technologies*, pages 31–40. Springer, 2004.
- [7] P. Pu, P. Viappiani, and B. Faltings. Increasing user decision accuracy using suggestions. In *Procs. of the SIGCHI conference on Human Factors in computing systems*, pages 121–130. ACM Press, 2006.
- [8] J. Reilly, K. McCarthy, L. McGinty, and B. Smyth. Dynamic critiquing. In *Procs. of the 7th European Conference on Case-Based Reasoning*, pages 763–777. Springer, 2004.
- [9] F. Ricci, D. Cavada, N. Mirzadeh, and A. Venturini. Case-based travel recommendations. In D. R. Fesenmaier et al., editors, *Destination Recommendation Systems: Behavioural Foundations and Applications*, pages 67–93. CABI, 2006.
- [10] S. Schmitt. *simVar*: A similarity-influenced question selection criterion for e-sales dialogs. *Artificial Intelligence Review*, 18:195–221, 2002.
- [11] C. A. Thompson, M. Göker, and P. Langley. A personalized system for conversational recommendations. *Journal of Artificial Intelligence Research*, 21:393–428, 2004.

Table 2: Results: averages for 50 dialogues between different users (Opt = Optimising; Pri = Prioritising; Rnd = Random) and different advisors (Null = no advisor; CoM = Complete on Maxima; IoM = Incomplete on Maxima; CoAN = Complete on All Next; IoAN = Incomplete on All Next)

| Database | User | Advisor | Queries tried (A) | Successful queries (B) | Queries in advice (C) | Utility shortfall (D) |
|-------------|------|---------|-------------------|------------------------|-----------------------|-----------------------|
| Marriott-NY | Opt | Null | 21.24 | 5.16 | 0.00 | 0.00 |
| | Opt | CoM | 4.67 | 4.67 | 45.96 | 0.00 |
| | Opt | IoM | 5.98 | 4.70 | 5.14 | 0.00 |
| | Opt | CoAN | 5.16 | 5.16 | 75.36 | 0.00 |
| | Opt | IoAN | 6.42 | 5.16 | 14.12 | 0.00 |
| | Pri | Null | 20.42 | 5.14 | 0.00 | 0.01 |
| | Pri | CoM | 4.70 | 4.70 | 46.80 | 0.01 |
| | Pri | IoM | 4.70 | 4.70 | 5.00 | 0.01 |
| | Pri | CoAN | 5.12 | 5.12 | 75.08 | 0.01 |
| | Pri | IoAN | 5.14 | 5.14 | 13.64 | 0.01 |
| | Rnd | Null | 33.34 | 7.94 | 0.00 | 0.00 |
| | Rnd | CoM | 3.60 | 3.60 | 36.10 | 0.32 |
| Cork | Rnd | IoM | 7.50 | 4.14 | 4.59 | 0.28 |
| | Rnd | CoAN | 7.76 | 7.76 | 110.36 | 0.01 |
| | Rnd | IoAN | 10.90 | 8.08 | 29.08 | 0.00 |
| | Opt | Null | 37.14 | 5.72 | 0.00 | 0.00 |
| | Opt | CoM | 5.55 | 5.55 | 59.02 | 0.00 |
| | Opt | IoM | 12.13 | 5.64 | 8.51 | 0.00 |
| | Opt | CoAN | 5.72 | 5.72 | 96.84 | 0.00 |
| | Opt | IoAN | 12.20 | 5.72 | 10.00 | 0.00 |
| | Pri | Null | 35.20 | 5.66 | 0.00 | 0.02 |
| | Pri | CoM | 5.66 | 5.66 | 60.50 | 0.02 |
| | Pri | IoM | 5.66 | 5.66 | 8.88 | 0.02 |
| | Pri | CoAN | 5.66 | 5.66 | 94.36 | 0.02 |
| Trentino-10 | Pri | IoAN | 5.66 | 5.66 | 11.12 | 0.02 |
| | Rnd | Null | 49.48 | 7.88 | 0.00 | 0.04 |
| | Rnd | CoM | 4.42 | 4.42 | 40.68 | 0.02 |
| | Rnd | IoM | 13.36 | 4.50 | 7.21 | 0.04 |
| | Rnd | CoAN | 7.80 | 7.80 | 139.02 | 0.02 |
| | Rnd | IoAN | 27.76 | 7.46 | 12.64 | 0.02 |
| | Opt | Null | 16.12 | 6.46 | 0.00 | 0.00 |
| | Opt | CoM | 6.09 | 6.09 | 69.88 | 0.00 |
| | Opt | IoM | 9.36 | 6.27 | 6.27 | 0.00 |
| | Opt | CoAN | 6.42 | 6.42 | 109.20 | 0.00 |
| | Opt | IoAN | 9.60 | 6.44 | 9.68 | 0.00 |
| | Pri | Null | 14.66 | 6.44 | 0.00 | 0.00 |
| Trentino-10 | Pri | CoM | 6.32 | 6.32 | 71.42 | 0.01 |
| | Pri | IoM | 6.32 | 6.32 | 6.46 | 0.01 |
| | Pri | CoAN | 6.44 | 6.44 | 108.38 | 0.00 |
| | Pri | IoAN | 6.44 | 6.44 | 10.08 | 0.00 |
| | Rnd | Null | 21.10 | 8.82 | 0.00 | 0.00 |
| | Rnd | CoM | 4.21 | 4.21 | 49.64 | 0.14 |
| | Rnd | IoM | 8.83 | 4.83 | 5.25 | 0.17 |
| | Rnd | CoAN | 8.68 | 8.68 | 151.14 | 0.00 |
| | Rnd | IoAN | 13.92 | 8.84 | 13.28 | 0.00 |