Intelligent Query Management for Travel Products Selection

Francesco Ricci[a],
Dennis Blaas[a],
Nader Mirzadeh[a],
Adriano Venturini[a], and
Hannes Werthner[a b]

[a] eCommerce and Tourism Research Laboratory
ITC-irst, Italy
{ricci, blaas, mirzadeh,venture,werthner}@itc.it

[b] University of Trento, Italy

Abstract

This paper presents a web based recommendation system aimed at supporting a user in building a personalized travel plan. A travel plan bundles together elementary travel components that are selected by the user in a mixed initiative way: the user poses queries and the recommender helps the user to reformulate the query and finally ranks the selected items. This paper concentrates on the methodological aspects that enable the interactive query refinement of the user queries, and in particular the intelligent relaxation of query constraints in case the query returns no items.

Keywords: recommendation systems; case-based reasoning; interactive query management.

1 Introduction

There is a fast growing number of web sites that support a traveler during the process of selecting a travel destination or booking a travel service as a flight or a room in a hotel. These web sites ask the user constraints or preferences related to the products, and then they search in a catalogue of products for those items that match the user's request. All the major eCommerce web sites dedicated to tourism, such as expedia, priceline, travelocity, tiscover, etc, implements this simple and quite effective pattern.

But there are some inherent limitations to this approach. First of all, to maintain simple and not tedious the man/machine interaction, the web form includes only a limited number of product features to be constrained by the user selection. But at the same time the form must try to catch all the user relevant preferences, that may depend on the specific characteristics of the user or on the very nature of the product. For instance in a commercial hotel reservation system the hotel and room descriptions may easily count hundred of features. This kind of problem is normally managed by leaving to the user to choose between at least two forms, one typically quite simple, and another one that shows the complete list of product features.

Second, when the results of a user request are searched in the catalogue two limit situations could happen: there could be too many results or, on the contrary, no result. These situations occur quite often, even if sometimes the user does not realize it. In fact, for instance when too many results match the user constraints a largely adopted "trick" consists of presenting only a few of them, by automatically filtering some results. The filtering technique is typically not personalized, and this will tamper with the goodness of the recommendation.

In our view the available solutions to these problems are not completely satisfactory. The tradeoff between maintaining the user interface simple and at the same time catching all the user needs is resolved in favor of the simplicity. This is correct but it leaves the space open to more sophisticated solutions.

In this paper we present a solution that relies on the observation of how this kind of problems are managed in a "real" shop with a "real" human to human interaction (customer/clerk). The major difference in this case relies on the fact that human to human interactions may be extremely complex and may unfold according to completely different patterns in relation to the customer inquires, clerk's reply and request for further specification.

The simulation of this behaviour has been the goal of many "intelligent" systems, which are based on Artificial Intelligent, initially in the form of expert systems and now as recommender systems (Resnick and Varian, 1997,Schafer et al., 2001,Burke, 2000). Along this line of research we propose to build a recommender system that extends a classical query manager with a set of "intelligent" functions that will help the user in the process of selecting a travel related product from a catalogue.

More in general, the proposed recommender can help a user in the task of bundling a travel plan starting from elementary travel "items", like activities, accommodation and events (Hwang and Fesenmaier, 2001, Klicek, 2001, Mitsche, 2001). It helps the user to reformulate the query exploiting both summary knowledge about the distribution of the data (Chaudhuri and Dayal, 1997) and a simple "decision model" that determines feature's product importance during the selection task. The prototype described in this paper partially implements and extends the concepts described in a previous paper (Ricci and Werthner, 2001). Moreover, this prototype exploits a mediator component, adapted to the needs of case-based reasoning methodology (Aamodt and Plaza, 1994), that is fully described in (Ricci et al., 2001). In this paper we concentrate on the prototype functions devoted to support the interactive query management and in particular we illustrate how the system deals with queries that do not provide any results.

The paper is organized as follows. Section 2 describes a typical scenario of usage and presents the implementation of this prototype: the main software components and in particular the graphical user interface. Section 3 describes the technical details of the intelligent component that supports interactive query management

2 The Recommendation System

Let us make an example that illustrate the type of decision support that is implemented in the system. A user is accessing the recommender to build a travel plan. At a certain stage of the travel plan composition process he is looking for a hotel in an already selected destination. A dialogue as described in the following is managed by the recommender. This does not proceed using a Natural Language interface but with a set of standard web forms. We use here an informal description to focus on the information content exchange rather than on the format.

- *Customer:* I'd like to find an hotel with room-price < 100, with parking and Internet connection
- *Recommender:* I'm sorry there is no match. BUT: there are several hotels with room-price < 150, with parking and Internet connection OR one hotel with room-price = 96, with Internet connection but with no parking
- *Customer:* let me see this cheap option [he sees it]
- *Custome :* now let me see the other more expensive solutions
- *Recommender:* would you like to have a restaurant in the hotel?
- *Customer:* yes [he gets 10 ranked suggestions]

What emerges from this description is that the recommender not only passively replies to the user, but also suggests modifications to the user's query, driving actively the dialogue. This is an important feature of the recommender that put it in the class of "conversational" systems (Aha and Breslow, 1997,Göker and Thomson, 2000).

We have developed a prototype that implements the above quoted dialogue style. The system is a web application that is connected to the same rather complex Oracle database (more that 100 tables) that stores all the information content of the Local Tourism Organization of Trentino, that is currently accessible at www.trentino.to. The user can issue four types of queries to select: destination, accommodation, event, and sport or leisure service. The user is then guided by the system through a set of interactions whose goal is to determine a small number of items (destination or accommodation or event or leisure service) that could be recommended to the user. In case an initial query returns no item the system suggests a set of alternative queries that would give some results (see Section 3). Conversely, when the user's query produces a very long list of results the system asks additional information in order to limit the number of suggestions. At the end of this first stage of the interaction, whose goal is to select a small set of candidate items, these selected items are ranked according to the similarity to items already selected by the user (in his past travel plans) or by other similar users.

In that way, the system enables the user to build up his own personalized travel by aggregating elementary items (locations, services and activities). Case-Based Reasoning techniques (Aamodt and Plaza, 1994) enable the user to browse a repository of past travels or a catalogue of services and rank elementary items included in a recommendation.

The central building block of this recommendation system is the TravelItem resource (TI in short). This can be a location, a tourism service or an activity. The second modeling assumption refers to a TravelPlan i.e. a coherent aggregation of TravelItems chosen by the traveler to be ``consumed'' during the travel. The set of all TravelPlan objects composed by a user using the system plays the role of the main case base. But case-based reasoning techniques (retrieval) are used to search and select a single TravelItem as well.

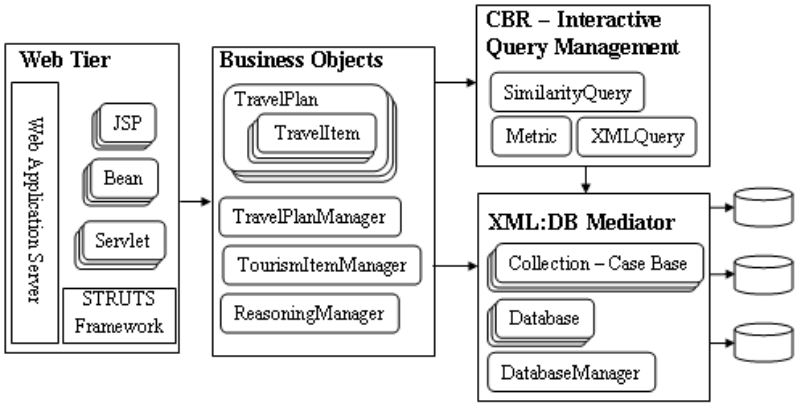

Fig. 1. Architecture of the Recommendation System

Figure 1 depicts the general architecture of the recommendation system. The reference application model is the Java 2 Enterprise Edition Architecture (www.java.sun.com/j2ee/). In the Web Tier we use Struts, an open source framework useful for building web applications with Java Servlet and JavaServer Pages (JSP) technology. Struts encourages application architectures based on the Model-View-Controller (MVC) design paradigm (http://jakarta.apache.org/struts/). The Business Object tier contains the TravelPlan and TravelItem implementations as well as three manager objects that implement the main system functions (management of the plans and recommendation support). The "CBR - Interactive Query Management" and the XML:DB Mediator components implement the mediator architecture (Wiederhold, 1992, Florescu et al., 1998), that retrieves data from legacy databases and models this information as case bases (see (Ricci et al., 2001) for additional details on the Mediator component). The XML:DB Mediator is our custom implementation of the homonymous Application Program Interface (www.xmldb.org) that is designed to enable a common access mechanism to XML databases. The API enables the application to store, retrieve, modify and query data that is stored either in an XML database, or in the Oracle database.

Regarding the recommendation functions, the user has two main modalities to obtain a destination recommendation. First, he can query a repository of TravelItem objects (as mentioned above). This is supported by a classical search (augmented with support for query relaxation and tightening) and similarity based search and scoring. In this modality a TravelItem catalogue is searched for the most appropriate items, and CBR

technology is used for ranking the results. Second, the user can search the repository of past suggested TravelPlan objects for travels made by other "similar" users or in conditions similar to those specified by query constraints or by a tentative (incomplete) probe TravelPlan that the user is composing.

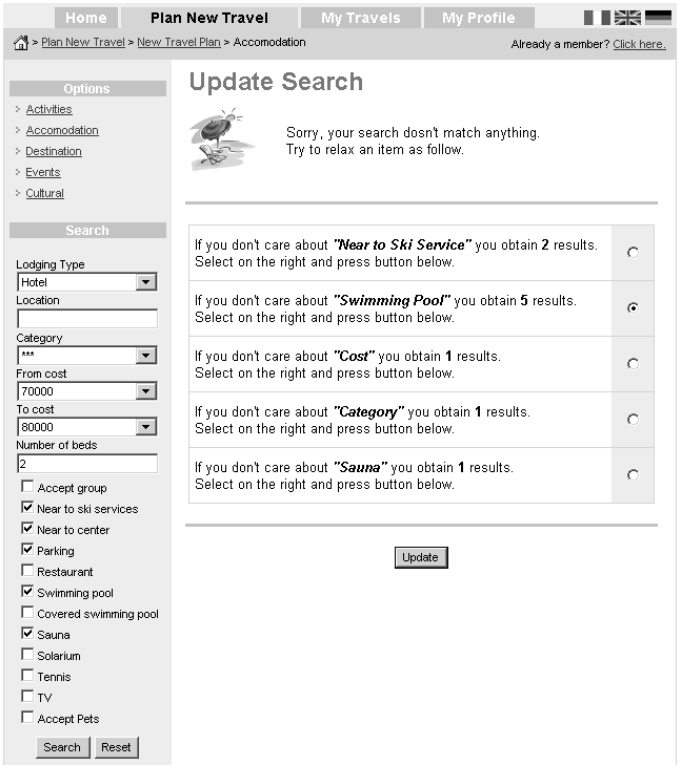

Fig. 2. Snapshot of the user interface

Figure 2 provides a snapshot of the Graphical User Interface. It is shown the result of user query (left side) to a catalogue of accommodations, that has failed to retrieve any hotel with the requested features. The system suggests possible changes to the query (discard some query constraint) that will yield some results (e.g. if the user don't care about the swimming pool, then 5 hotels match the other constraints). After the user has selected one of the proposed changes, a ranked list of hotels, is displayed and the user can add the selected item to the current travel plan or go back to a previous stage of the decision process.

3 The Technical Approach

In this section we present the basic mathematical structures that are implemented in the recommender and the dialogue management algorithm that supports query refining for constraint relaxation.

A travel plan is made up of travel items. Travel items belong to many different types, for instance, an accommodation type or a sport activity type. We shall denote with X an item space (of a particular type). An item space X models a single type of items and it is a vector space $X = \prod_{i=1}^{n} X_i$, i.e a travel item of type X is represented as a n-dimensional vector of features (we use the word "feature" as a synonym of an item attribute or item attribute space of values). An item space is a catalogue of product/services accessed through the web application described in Section 2. For instance the hotel "Majestic, located in Pza. Navona, in Rome, whose single costs 125 Euro per night, has category 3, has lift and AC" can be described as the vector x = *(Rome, PzaNavona, Majestic, 3, 125, True,True,)*, and it is a member of $Location \times Address \times Name \times Category \times Cost \times AC \times Lift$, where, e.g "Rome" is the feature value of the "Location" feature. There are three general types of features:

Finite Integer: A finite integer feature contains a finite set of elements among which a relation of total ordering is defined (denoted with <). Examples of finite integer features are : *Level = { beginner, intermediate, advanced }*, *Category = {0, 1, 2, 3 }*. The order relation states that, $0 < 1 < 2 < 3$, or *beginner < intermediate < advanced*.

Real: A real feature takes values in an interval of the real numbers (e.g. Cost). We will assume that is always normalized to [0,1] (possibly without the endpoints). Real features have a total order relation as well.

Symbolic: A symbolic feature contains a finite number of elements. Examples of symbolic spaces are: *AC = {True, False}* the boolean space; or *Location = {Rome, Milan, Venice}*. In principle no relation between the values of a symbolic feature is given. So we will assume that they are only different symbols[1]

Given an item space $X = \prod_{i=1}^{n} X_i$, we shall say that X contains the items $x^1, \ldots, x^N$, and $N = |X|$. Moreover, we shall denote with $x^j = (x_1^j, \ldots, x_n^j)$ the components of a vector $x^j \in X$.

3.1 The Query Language

The query language that we are considering is quite simple. Let $X = \prod_{i=1}^{n} X_i$ be an item space, then a query Q is obtained by the conjunction of simple constraints, where each constraint involves only one feature. More formally, $Q = C_1 \wedge \ldots \wedge C_m$, where $m \leq n$, constraint C_k involves feature x_k[2], and:

[1] It is evident that relations can be defined in many cases, for instance, the "closeness" relation for the "Location" feature.

[2] For sake of simplicity, we assume that the features are ordered in such a way that the first k features are involved in the query.

$$C_k = \begin{cases} x_k = v_k, & \text{if } X_k \text{ is symbolic} \\ l_k \le x_k \le u_k, & \text{if } X_k \text{ is finite integer or real} \end{cases}$$

Example. Let $X = AC \times Category \times Cost = \{True, False\} \times \{1, 2, 3\} \times [0, 1]$, and $Q = (x_1 = T) \wedge (0.2 \le x_3 \le 0.5)$. Then the query Q selects all the hotels that have air conditioning and room cost between 0.2 and 0.5. So if $X = \{ (True, 3, 0.6), (True, 3, 0.5), (False, 3, 0.4), (False, 2, 0.4) \}$, the query will retrieve the items: $(True, 3, 0.5), (True, 2, 0.4)$.

3.2 Query Relaxation

Query relaxation changes a query definition in such a way that the number of items returned by the query is increased. For instance $Q' = (0.2 \le x_3 \le 0.5)$ is a relaxation of the query $Q = (x_1 = True) \wedge (0.2 \le x_3 \le 0.5)$, where the first constraint is relaxed. A query is typically relaxed when the result set retrieved from the item space X is void, or when the user is interested in having more examples to evaluate.

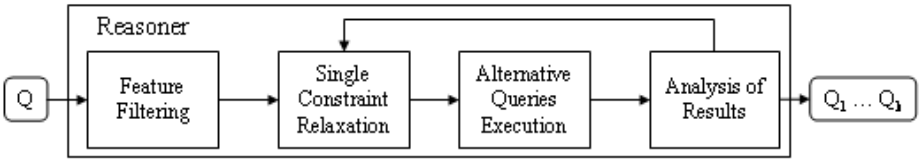

Fig. 3. The relaxation process

The query relaxation suggestion process is implemented in the Reasoner component (Figure 3). This component takes in input a query $Q = C_1 \wedge \ldots \wedge C_m$ and builds a new set of relaxed queries $Q_1, \ldots Q_k$, such that each Q_i relaxes only one constraint of the original query. In the following discussion we shall use the query $Q = ((x_1 = ski) \wedge (x_2 = cavalese) \wedge (x_3 = T) \wedge (x_4 \le 1000))$, where we are searching in a database of sport activities/services, and the features are *SportType*, *Location*, *School*, *Cost*. The Reasoner has some modules that process the initial query according to the following procedure:

1. **Features Filtering.** When a query must be relaxed, there are some feature constraints that cannot be changed without changing the information goal of the user. Referring to the example above, the user might well be interested in some school a bit more expensive or maybe not in Cavalese but he is interested in skiing, not in fishing. This module identify those features' constraints that cannot be relaxed. In the example above it is found that only the Cost, Location and Scool features can be relaxed.

2. **Single Constraint Relaxation.** Whenever the relaxable constraints have been identified, for each of them a "relaxed" version (C_i') must be identified. Two different approaches are exploited according to the feature type:

- **Symbolic feature constraint relaxation.** In this case relaxation means that the constraint is discarded.

- **Finite integer and real feature constraint relaxation.** In this case the range of allowed values is expanded by a certain percentage that varies according to the feature type.

After this stage the relaxed constraints in Q are: $x_2 = ALL$, $x_3 = ALL$ and $x_4 \leq 1100$. The *ALL* keyword means that all the values are allowed for that feature.

3. **Alternative Queries Execution.** At this stage, for each "relaxable" constraint a new relaxed version of the original user's query is built, with only that constraint relaxed. In our example we have three new queries: $Q_1 = ((x_1 = ski) \wedge (x_2 = ALL) \wedge (x_3 = True) \wedge (x_4 \leq 1000))$, $Q_2 = ((x_1 = ski) \wedge (x_2 = cavalese) \wedge (x_3 = ALL) \wedge (x_4 \leq 1000))$, and $Q_3 = ((x_1 = ski) \wedge (x_2 = cavalese) \wedge (x_3 = True) \wedge (x_4 \leq 1100))$. These relaxed queries are executed and the number of items retrieved by each single query is determined (counts)[3]

4. **Analysis of Results.** When the results (counts) are obtained they must be analysed. The goal is to understand if the relaxed queries have produced an improvement, i.e., if the new set of results (for each query) is not still void or some of them has become too large. Among those queries that still return a void (or too large) set two situations may arise: if the constraint relaxed involves a symbolic feature, then the query cannot be further relaxed (or tightened); if the feature is integer or real then a new relaxation (or tightening) is tried by sending this information to the Single Constraint Relaxation module. Let us assume, for sake of simplicity, that in our example: Q_1 returns 10 items, Q_2 returns none, and Q_3 returns 5 items. Then two new queries are suggested to the user Q_1 and Q_3, i.e., the systems tells to the user that there are ski schools: 1) that are not in Cavalese but cost less than 1000; 2) or that are in Cavalese but cost a bit more but less than 1100.

Some additional comments are in order. First, the list of features constraints that cannot be relaxed (in the Feature Filtering module) is defined statically for each query type. The system manages a number of query types (currently five) and for each of them a meta-data list those features that cannot be discarded in a query. Second, when relaxing a finite integer or real feature constraint the reasoner uses again a meta-data information (for each feature type) to determine the percentage of increase in the range. For instance, when relaxing a "Cost" feature constraint, the user input range is increased by 10%.

3.3 Discussion

The above presented procedure to relax user queries is still limited in a number of ways. We now briefly mention a set of improvements that are under development.

[3] Actually the computation of these counters is faster than the complete execution of the query, that is the retrieval of items from the data base.

First of all, we are going to implement methods that exploit session information for determining, in the Feature Filtering module, those constraints that cannot be relaxed. For instance, if the user has already selected a destination, then the "Location" feature in a service search is not proposed for relaxation.

Secondly, the constraints on symbolic features should not be completely discarded in the Single Constraint Relaxation module. According to information related to the subtype of the feature the equality constraint can be changed to a different equality constraint or to a range constraint. For instance if the constraint is *Location = cavalese* this may be relaxed to *Location = cavalese OR molina* or even better, introducing specific relaxation operators, e.g. *Location = Close(cavalese)*.

Thirdly, the method used to generate relaxed queries is very simple, i.e., only one constraint is changed and this is repeated for all those constraints that can be relaxed. This choice is intended to limit the combinatorial explosion of such changes. In fact if a query has m constraints the number of relaxed queries that one can generate modifying an arbitrary number of constraints is 2^m. Hence a method for limiting the nature of the changes is to be adopted. We are implementing a refined generation method that limits the number of relaxed queries proposed by the system but can propose relaxations that simultaneously operate on more that one feature.

4 Conclusions

This paper describes a work in progress at a newly established research center on eCommerce and Tourism (http://ectrl.itc.it). The proposed approach has been implemented for the Azienda di Promozione Turistica of Trentino (Trentino Destination Management Organization), and will be shortly validated within a use group. A more comprehensive version will be developed as main result of an European IST Project (DIETORECS, in collaboration with: TIScover AG - Travel Information Systems (A), Institute for Tourism and Leisure Studies - Vienna University of Economics and Business Administration (A), National Laboratory for Tourism and eCommerce - University of Illinois at Urbana-Champaign (USA) and Azienda per la Promozione Turistica del Trentino (I)).
The current result of our work is a methodology to ease the selection of items from a catalogue by suggesting relevant changes to the users' queries, avoiding a typical problem that occur in these situations: the query constraints are too strict and no suggestion can be made. More in general, this is only a partial implementation of a comprehensive middleware for issuing personalized recommendation to a leisure traveler in identifying and aggregating elementary services or activities to be consumed. It is based on the principle that effective suggestions are based on a combination of factors like: appropriate destination modeling, data retrieval and filtering with both precise and approximate matching, scoring using personal preferences that can be derived from a base of previous cases.
There are still many open issues, and some of them, which are related to the interactive query management component, have been already quoted in Section 3.3. In addi-

tion, we are now focussing on the problem of suggesting modifications of the query that will reduce the number of items returned, and in implementing the similarity metric needed to rank the items and retrieve similar travel plans.

5 Acknowledgement

This work has been partially funded by CARITRO foundation (under contract "eCommerce e Turismo") and by the European Union's Fifth RTD Framework Programme (under contract DIETORECS IST-2000-29474).

References

Aamodt, A. and Plaza, E. (1994). Case-based reasoning: foundational issues, methodological variations, and system approaches. *AI Communications*, 7(1):39-59.

Aha, D. and Breslow, L. (1997). Refining conversational case libraries. In *Case-Based Reasoning Research and Development, Proceedings of the 2nd International Conference on Case-Based Reasoning (ICCBR-97)*, pages 267-278. Springer.

Burke, R. (2000). *Encyclopedia of Library and Information Science*, chapter Knowledge-based Recommender Systems. To appear.

Chaudhuri, S. and Dayal, U. (1997). An overview of data warehousing and olap technology. *SIGMOD Record*, 26(1):65-74.

Florescu, D., Levy, A., and Mendelzon, A. (1998). Database techniques for the world-wide web:a survey. *SIGMOD Record*, 27(3):59-74.

Göker, M. H. and Thomson, C. A. (2000). 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.

Hwang, Y. H. and Fesenmaier, D. R. (2001). Collaborative filtering: strategies for travel destination bundling. In *Enter2001*, Montreal, Canada.

Klicek, B. (2001). Tourist's decision making process assisted by the web and multimedia intelligent advisory system. In *Enter2001*, Montreal, Canada.

Mitsche, N. (2001). Personalised travel counselling system: providing decision support features for travellers. In *Enter2001*, Montreal, Canada.

Resnick, P. and Varian, H. R. (1997). Recommender systems. *Communications of the ACM*, 40(3):56-58.

Ricci, F., Mirzadeh, N., and Werthner, A. V. H. (2001). Case-based reasoning and legacy data reuse for web-based recommendation architectures. In *Proceedings of the Third International Conference on Information Integration and Web-based Applications & Services*, pages 229-241, Linz, Austria.

Ricci, F. and Werthner, H. (2001). Case-based destination recommendations over an xml data repository. In *Enter2001*, Montreal, Canada.

Schafer, J. B., Konstan, J. A., and Riedl, J. (2001). E-commerce recommendation applications. *Data Mining and Knowledge Discovery*, 5(1/2):115-153.

Wiederhold, G. (1992). Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38-49.