# II Design of Destination Recommendation Systems

# 6 Case-based Travel Recommendations

Francesco Ricci, Dario Cavada, Nader Mirzadeh and Adriano Venturini

## 1. Introduction

Searching for travel-related information and services is one of the top web activities and there is a fast-growing number of websites that support a traveller in the selection of a travel destination or a travel service (e.g. flight or hotel). Users search for destination-related information such as point of interest, historical data, weather conditions and for products and services such as travel packages, flights and hotels. The wide spectrum of information currently provided by a number of web actors include: online travel agency, tour operators, cruise operators, destination management organizations (multidestination, regional, city), airlines, hotel chains, convention and visitors' bureau (Werthner and Klein, 1999; Buhalis, 2003).

Basically, the websites maintained by the various tourism organizations offer search tools and content browsing. In the first case, mostly exploited to select a product or service, the user is required to input some product constraints or preferences that are matched by the system in its electronic catalogue. In the second case, the user is offered to navigate the website and browse the structured multimedia content. Hence, the technology largely exploited in the above-mentioned websites is not much different from those found in any other e-commerce site.

Trip planning is a complex problem-solving activity (Moutinho, 1987; Ankomah *et al*., 1996; Fesenmaier and Jeng, 2000; Hwang *et al*., 2002). The terms 'travel plan' and 'destination' lack a precise definition; indeed, even the destination spatial extension is known to be a function of the travellers' distance from, and knowledge about, the destination. Importantly, travel plans may vary greatly in structure and content. For instance, some search for pre-packaged solutions (all included), while other 'free riders' want to select each single travel detail independently. Because of this, the straightforward implementation of general decision aid and recommendation technologies often fail when applied to travel planning and destination choice (Ricci, 2002). Case-based reasoning (CBR), a problem-solving methodology that has been recently exploited to build a number of product recommender systems (Cunningham *et al*., 2001; Shimazu, 2001; Bridge and Ferguson, 2002; McGinty and Smyth, 2002, 2003; McSherry, 2002, 2003) must be reshaped to fit the requirements of the travel domain. The CBR recommender is a knowledge-based system that exploites a 'search and reuse' approach. The search is performed on the catalogue of items (to be suggested), and the reuse of retrieved items could be implemented in different ways, from a simple display of the retrieved items to a more complex adaptation of the items to fit to the peculiar preferences of the user.

We describe Trip@dvice, a travel recommendation methodology that supports the selection of travel products (e.g. a hotel or a visit to a museum or a climbing school) and building a *travel plan*, which is a coherent (from the user point of view) bundling of products. In this approach the case base is composed of travel plans built by a community of users. A case is structured hierarchically (Smyth and Keane, 1996; Stahl and Bergman, 2000) including components that represent the search and/or decision problem definition, i.e. the travel's and the travellers' characteristics, and the problem solution. Trip@dvice extends case-based reasoning with interactive query management (Gaasterland *et al*., 1992). This system attempts to understand the gist of a user request in order to suggest or answer related questions, to infer an answer from data that are accessible or to give an approximate response. Trip@dvice tries first to cope with user needs satisfying the logical conditions expressed in the user's query and, if this is not possible, it suggests query changes (relaxation and tightening) that will produce acceptable results. In Trip@dvice failures to satisfy all user needs are not solved relying on similarity-based retrieval, as is usual in CBR. Instead, (case) similarity is exploited, first, to retrieve relevant old recommendation sessions and, second, to rank the items in the result set of the user's given logical query.

## 2. Case-based Reasoning

CBR is a multidisciplinary subject that focuses on the reuse of experience, which is modelled as a case (Aamodt and Plaza, 1994; Aha, 1998). There are at least a couple of interpretations of CBR: a plausible high-level model for cognitive processing (Schank, 1982; Kolodner, 1993) and as a computational paradigm for problem solving (Aamodt and Plaza, 1994). We shall focus on the second interpretation only. Aamodt and Plaza refer to CBR as a problem-solving paradigm that uses the *specific* knowledge gathered solving concrete problem situations. This is in opposition with more classical approaches based on *general* knowledge about the problem domain (domain theory), which can be expressed using a knowledge representation language such as those based on rules, frames, semantic networks and first-order logic.

The first fundamental issue in CBR is the case representation language (case model) and therefore the scope of the case concept itself. In any CBR application the designer must decide: (i) what to store in a case (content); (ii) the appropriate structure for describing the case contents; and (iii) deciding how the case memory must be organized. What to store in a case is typically application-dependent and therefore we shall not comment on this now (see Section 5 for details on the Trip@dvice model). There are three major ways to basically represent (implement) a case: as a linear vector of features (or more in general a set of features); as a text, eventually semistructured and with mixed content; and as a complex structured object such as a labelled graph or a pattern of objects in an Object-Oriented language. There are also 'mixed' approaches that merge, for instance, text-based and vector-based representations but as a first classification this is quite adherent with the current reality of CBR applications.

In a vector-based representation a case is described as a fixed list of heterogeneous features (nominal-string, real numbers, integer numbers, Boolean, etc.). The CBR systems that adopt this representation language are usually derived from, or strongly influenced by, Machine Learning and Pattern Recognition, and are defined as exemplar-, instance- or memory-based (Aha *et al.*, 1991). Often, these approaches view problem solving as automatic classification or function approximation tasks (Witten and Frank, 2000). In a text-based CBR system the major input for case content is considered to be the information contained in a text. The text itself is typically processed in order to come up with the final case object. In this 'compilation' step, indexes are built, part of the text is tagged with meta-data information and possibly the text is summarized. This approach usually integrates CBR with Information Retrieval (Börner *et al.*, 1996; Burke *et al.*, 1997; Daniels and Rissland, 1997; Lenz and Burkhard, 1997) and is now again raising a lot of interest because of the Web and semi-structured languages like hypertext mark-up language (HTML) and extensible mark-up language (XML) (Shimazu, 1998).

In the complex structured approaches, cases are modelled as combinations of the previous approaches or using graph-based data models (Bunke and Messmer, 1994; Plaza, 1995; Gebhardt *et al.*, 1997; Macedo and Cardoso, 1998; Ricci and Senter, 1998; Stahl and Bergman, 2000). Complex structured case representation languages are often used in planning and design applications where the structure of the case reflects the task–subtask or component–subcomponent hierarchy. The approach, described in this report, belongs to this last category and exploits XML as target implementation data model (see Arslan *et al.*, 2002) for a detailed description of the XML model used in the DieToRecs prototype, shown in Section 6.2).

Independent from these approaches, a case is usually decomposed into three subcomponents: the problem description, the solution and the outcome. The first refers to the part that is matched when a new problem arises. This must include all the information needed to first guess that a case can be fruitfully reused for solving a similar problem. Considering problem solving as function approximation, the problem description becomes the domain of the function, where the co-domain is given by the solution and outcome.

The solution models the chunk of information that is searched for, e.g. the diagnosis of a malfunction or the plan to reach a destination. Finally, the outcome provides an evaluation of the applicability or goodness of the solution to the given problem. Our case model further extends this model, as there is no sharp separation between problem and solution components. In a stage of the decision process a case component must be defined or selected (e.g. the destination) and in a successive stage this can be used as part of the problem description when, for instance, the user is searching for attractions. The CBR problem-solving cycle is universally recognized as the basic common denominator of all CBR approaches and is summarized in Fig. 6.1 and discussed more fully below.

1. **Retrieve.** Given a problem description, retrieve a set of cases stored in the case base, whose problems are evaluated as similar. A similarity metric is used to compare the problem component of the new case being built with the problem description of the cases in the base. Indexes, case base partitions, case clusters or other similar tools can be used to speed up this stage.
2. **Reuse.** The retrieved cases are reused to build the solution. This stage could be very simple, e.g. only extract the 'solution' component from one retrieved case, or much more complex, e.g. to integrate all the solutions extracted from the retrieved cases to build a new candidate solution.
3. **Revise.** The solution is then adapted to fit the specific constraint of the current situation. For instance, a reused therapy for a patient suffering for similar disease must be adapted to the current patient (e.g. considering differences in the weight or age of the two patients).
4. **Review.** The constructed solution is evaluated applying it (or simulating the application) to the current problem. If a failure is detected, backtracking to a previous stage might be necessary. The 'reuse', 'revise' and 'review' stages are also called case adaptation.

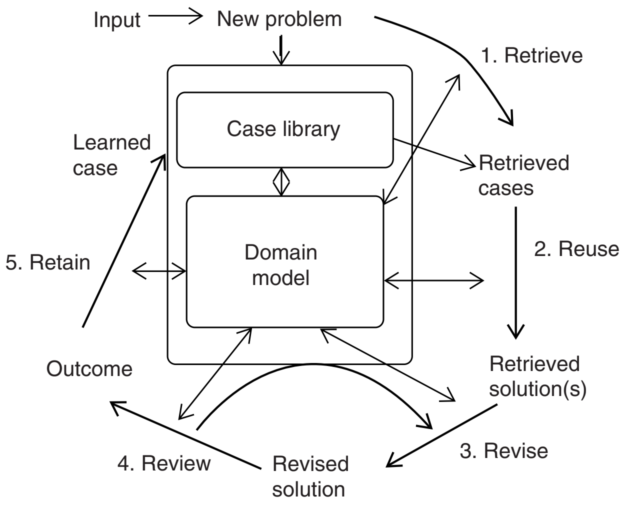


**Fig. 6.1.** The CBR problem-solving process (from Aha and Breslow, 1997).

5. **Retain.** The new case is possibly added to the case base. Not all the cases built following this process must be added to the case base. There could be poorly evaluated cases or cases too similar to previous situations, and therefore not bringing new knowledge.

## 3. CBR and Travel Planning

CBR may be used to build recommender systems, and a number of prototypes have proved the effectiveness of this methodology (Aimeur and Vézeau, 2000; Burke, 2000a,b; Doyle and Cunningham, 2000; Göker and Thomson, 2000; Smyth and Cotter, 2000; Bridge, 2001; Cunningham *et al*., 2001). When CBR is applied to recommender systems, the user needs and wants basically define the problem to be solved and a product suggested is considered as the solution. Hence, CBR recommender systems typically provide suggestions for a product, first asking the user to specify some personal data and preferences related to the product for which a suggestion is searched, and then retrieving from the case base a subset of cases that best match the input description. A case in the memory, in order to adhere to the CBR problem-solving loop described in Section 2, should represent a problem together with its solution; hence the product recommended (solution) together with the motivations for such a recommendation, i.e. a description of the situation in which the user asks for a recommendation. Actually, almost all the CBR recommender systems take a simpler approach: they simply define the case base as the full list of available products. In other words, they assume that the product description, i.e. basically a set of attributes of the product, can even play the role of problem description. We shall comment on this issue by referring to an example.

AQ1

Lenz was among the first to apply CBR to travel and tourism in the CABATA system (Lenz, 1996, 1999). In these studies, CBR was exploited as a tool to issue similarity-based queries to a catalogue. The user is supposed to enter the partial specification of a hotel and the system responds with the most similar ones in the catalogue. So, for instance, if the user enters a partial description of a hotel such as 'cost = 100 and location = Rome', the system would retrieve all the hotels that satisfy those conditions (if any) and also those that do not match all these requirements but are similar, e.g. a hotel that costs 110 and whose location is Rome. In this example, the user needs and wants are modelled by two attributes of the hotel (cost and location). One first observation relates to the retain stage of the CBR problem-solving loop. If, for instance, the user selects the hotel 'Gladiator', among those shown by CABATA, the system does not store in the case base that a given problem, i.e. 'cost = 100 and location = Rome' was solved by the hotel 'Gladiator'. In that respect CABATA, and similarly many other CBR recommender systems, do not close the CBR learning loop, retaining the newly acquired experiences. Another major limitation of all the CBR recommenders that identify a product with a case is that users can query the case base only referring to attributes of the product. Hence, if, for instance, the user would like a hotel with ‘cost = 100 and location = Rome’ and ‘suited for a family with children’, and this ‘suited for’ attribute is not part of the hotel description, the system will never learn the association of some hotels to this attribute. Conversely, if the system would store the full list of user needs and wants, even those that are not explicitly represented as attribute of the product, together with the product chosen at the end of the problem-solving process, the case base could be mined to discover this kind of implicit associations.

Another consequence of the limitation of modelling a case as a product to be recommended is that it is impossible to apply CBR for those products or services that are sold, only aggregating more elementary components. A travel plan is a typical example of a bundled product, it generally comprises some transportation services (flight, train or car), accommodations, attractions to visit and activities to do at the destination. Sometimes this is prepackaged, but one of the main motivations for going online and searching for travel information is to build a tailor-made travel selecting from multiple suppliers and catalogues. To make possible such a bundling the user should be able to search in a range of catalogues but at the same time to compose a single plan where these elementary products fit well together. Once again, it is clear that in travel planning, the problem (user needs and wants) and the solution (tailor-made travel plan) cannot be described as predefined products in a catalogue. These issues motivate the methodology we shall describe in the following sections.

## 4. Trip@dvice Methodology

We have designed a novel hybrid recommendation methodology called Trip@dvice that integrates CBR, interactive query management and collaborative-based filtering. Trip@dvice is motivated by some basic requirements:

- Tourism products and services typically have **complex structures**, where the final recommended item is an aggregation of more elementary components. For instance, a trip may bundle a flight, an accommodation and a rental car. Similarly, in other application domains, such as computers, a desktop computer may be sold together with a printer, a monitor and a scanner.
- The recommender systems based on Trip@dvice must allow the user to **bundle a mix-and-match travel plan**. This can either comprise a prepackaged offer or can be obtained by selecting travel components (items) such as locations to visit, accommodations, attractions and services.
- The recommendation methodology must support the implementation of advanced search functions that are still perceived by the user as conventional, and simple to use, as in **form-based information search engines**. This would make the methodology simple to integrate into existing systems. A recommender system, exploiting the methodology, must provide a range of query-forms: for elementary products and services and for predefined combinations (e.g. a complete travel package).

- The recommendation methodology must support **dialogues between the user and the recommender system**. A user should be allowed to criticize, or comment on, a query result or to refine the query definition. On the other hand, the system should actively support this query-refinement process by suggesting the most reasonable and minimal changes either for relaxing or tightening the user preferences. The final goal is to present the user with a manageable set of options in few interactions.
- Both **short-term (goal-oriented) and long-term (stable) preferences** must influence the recommendation. Short-term preferences are highly situation-dependent, tend to be hard constraints and should dominate long-term preferences. For instance, if the user searches for a business flight, the system must shade the influence of a previous history of 'no frills' flights bought by the user for a leisure travel.
- **The system should bootstrap without an initial memory of user interactions**, i.e. the Trip@dvice methodology should support the user even when not enough cases are collected. If the system has not learned 'enough', then more straightforward search functions should be available.
- **Unregistered users should be allowed to get recommendations** without being identified, if they do not want to. The methodology must exploit in this case only knowledge acquired during the current recommendation session.
- The methodology should **support a large number of user typologies with their preferred decision styles**. Hence, users should be allowed to provide in whatever order and amount they like general and detailed travel needs and wants. Users with a clear picture of what they are looking for should find immediately the searched product or understand why this is not attainable and what compromises they must accept. Conversely, users with a less clear goal should be supported in a more explorative browsing of the options.
- **The system should not assume that products and users' needs and wants completely overlap in their definition.** The methodology must exploit the characterization of the traveller needs and wants that are known, according to the literature on travel decision choice, to influence or determine their choices.

Trip@dvice bases its recommendations on a case model that capture a unique human–machine interaction session. A case collects information provided by the user during the session, the products selected and some stable user-related preferences and demographic data if it is registered. Recommendation sessions are stored as cases in a repository (case base). In addition to the case base, catalogues of products (databases) described according to the supplier view are also exploited.

Input information provided by the user during an interaction session fall into two categories: content queries and collaborative features. Content queries constrain attributes of the products in the catalogues. For instance, 'cost = 100 and location = Rome' is a content query on the Hotel catalogue. Collaborative features may not be descriptors of the products, are acquired from the user and are meant to describe the recommendation problem. For instance, the nationality of the user or the travel purpose could be used to describe a travel, and specify the context in which a travel is built. These in general are not part of the description of the products found in the catalogue. Collaborative features, as the name suggests, are exploited in Trip@dvice to identify similar past recommendation sessions, e.g. human–machine interactions took place with similar user with similar needs and wants.

The recommendation process is basically initiated by the system with a request for some collaborative features. Users can either input some of these or completely skip this stage and ask for suggestions. In the first case, users are forwarded to a search step where they can query the catalogues (for elementary and already bundled products) and get some ranked recommendations with their corresponding rationale. In the second case, users are immediately presented with a limited set of alternative travel options and they are only requested to provide a feedback ('I like it') about the shown options. This initiates a 'conversation' that after some iteration is supposed to terminate with a selection.

In both situations the collaborative features provided by the user during the interaction and the current case (the products and services selected in the interaction) are exploited to retrieve similar cases from the case base. This means that the similarity measure uses only the collaborative features and the cart composition to estimate what the other cases in the case base are that could provide useful knowledge for personalizing the interaction and the results. The exact definition of the recommendation procedure and in particular of the similarity function is described in the following sections. The idea is that what has been selected by other users, and was put in the carts, can provide useful knowledge to personalize the interaction. But to be effective this mechanism requires that only those cases that are pertinent in the current recommendation session be reused; hence, there is an evaluation of similarity that takes into account the traveller and travel characteristics (collaborative features) and products selected (those in the cart).

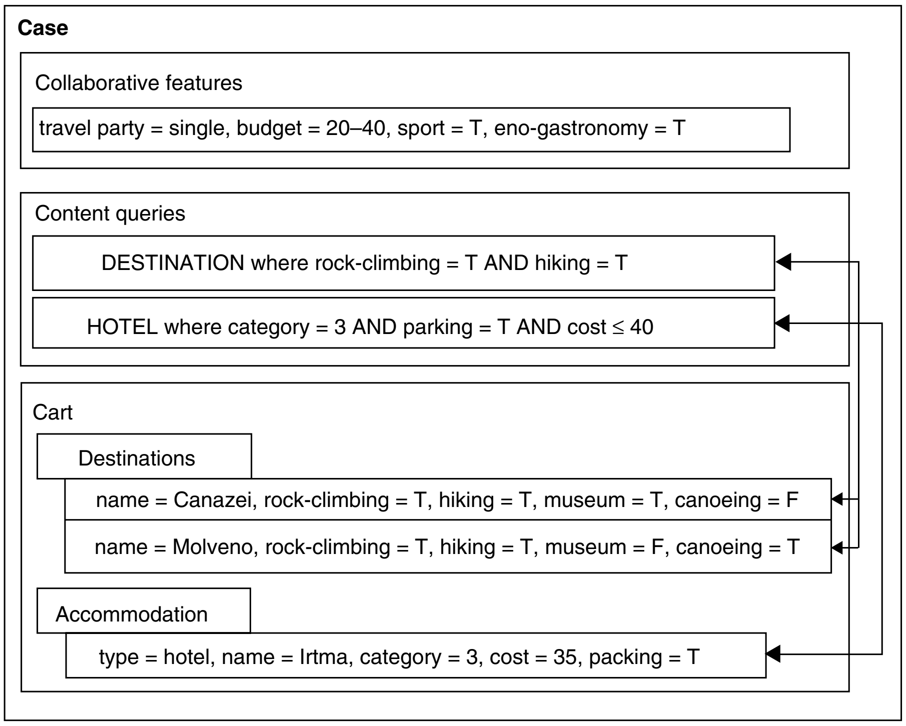
## 5. Case Model

In Trip@dvice, a case represents a user interaction with the system, and therefore is built incrementally during the recommendation session. A case comprises the following main components:

- **Collaborative Features (clf)** are features that describe general users' characteristics, wishes, constraints or goals (e.g. desire to relax or to practice sports). They capture preferences relevant to the users' decision-making process, which cannot generally be mapped into the features of products in the electronic catalogues. These features are used to measure case (session) similarity. Knowledge of the domain and the decision process is essential to select the right collaborative features (Ricci *et al*., 2002a).

- **Content Queries (cnq)** are queries posed over the catalogues of products. Content queries are built by constraining (content) features, i.e. descriptors of the products listed in the catalogues. Some content features can also be exploited as collaborative features; hence, in general collaborative features and content features are not void of intersection. For instance, a constraint on the budget for a Hotel search is also exploited as a collaborative feature.
- **Cart** contains the set of products chosen by the user during the recommendation session represented by the case. A cart represents a meaningful (from the user's point of view) bundling of different products. For instance, a travel cart may contain some destinations, some accommodations and some additional attractions. A product in the cart may be rated by the user, and this typically occurs after the travel is done.

Figure 6.2 shows an example in the tourism domain. It represents a user, who is single, has a medium budget (between €20 and 40) and is looking for a vacation (destinations and accommodation) where he can practice some sports and have some eno-gastronomic 'experience'. These are the collaborative features. Then there are a couple of queries (over the product catalogues) that constrain content features of the products. He wants to stay in a three-star hotel, which has a private parking lot and has a cost per night of less than €40. The destination should be a resort suitable for rock climbing and


Case
Collaborative features
travel party = single, budget = 20–40, sport = T, eno-gastronomy = T
Content queries
DESTINATION where rock-climbing = T AND hiking = T
HOTEL where category = 3 AND parking = T AND cost ≤ 40
Cart
Destinations
name = Canazei, rock-climbing = T, hiking = T, museum = T, canoeing = F
name = Molveno, rock-climbing = T, hiking = T, museum = F, canoeing = T
Accommodation
type = hotel, name = Irtma, category = 3, cost = 35, packing = T

AQ2

**Fig. 6.2.** An example of a case.

hiking. Given these preferences, the user is supposed to have selected and added to his cart the Irma Hotel, and Molveno and Canazei resorts. In this example, the user has selected two destinations by querying the destination catalogue.

More formally, the Case Base (CB) is defined as follows: $CB \subseteq CLF \times \wp(CNQ) \times CART$, where $\wp(X)$ is the power set of $X$, i.e. the set of all subsets of $X$. The detailed description of a case will be discussed in Section 7, where we shall see how the similarity between two cases is computed. Here we provide a simpler presentation in order to focus on the recommendation methodology (process) and its main elements.

The **Collaborative Features (CLF)** can be modelled as a vector space $CLF = \prod_{i=1}^{l} CLF_i$, where $CLF_i$ could be a set of symbols, a finite subset of the integers or a real interval. In the example shown in Fig. 6.2, we have $CLF = TravelParty \times MinBudget \times MaxBudget \times Sports \times EnoGastronomy$, where $TravelParty = \{single, family, friends, couple, group\}$, $MinBudget = [0, 80]$, $MaxBudget = [10, \infty]$ and $Sports = Relax = \{T, F\}$. In the example shown, the collaborative features are $clf = (single, 20, 40, T, T)$. Please note that this is a simplified example. In real systems (such as those discussed in Section 8) a case can contain dozens of collaborative features. Moreover, we observe that real systems exploiting Trip@dvice may adopt a structured representation of CLF, i.e. subsets of features in CLF may be grouped together to form a composite feature. For instance, the MaxCost and MinCost may be grouped to define the Cost (structured) feature (cf. also Section 7 where this topic is further described).

**Content Queries (CNQ)** is the space of all the queries that the user can specify over products in the catalogues. We assume that each product can be represented as a vector of features $CNF = \prod_{i=1}^{n} CNF_i$. $CNF_i$ can be (as above for $CLF_i$) a set of symbols, a finite subset of the integers or a real interval. A catalogue $CAT \subset CNF$ is said to be of type $CNF$. A query $q$ over a catalogue $CAT$ is a conjunction of simple constraints, where each constraint involves only one feature. More formally, $q = (c_1, \ldots, c_m)$, where $m \leq n$, and $c_k$ is an equality constraint $(x = v)$ in case it refers to a feature that takes values in symbolic space $CNF_i$, and $c_k$ is a range constraint $(v \leq x \leq u)$ in case it refers to a feature that takes values in numeric space $CNF_j$.

Let $Q(CNF)$ be the space of all the queries over $CNF$. Furthermore, let us assume that there are $N$ product types $CNF^1, \ldots, CNF^N$. Then we denote by $CNQ = \bigcup_{i=1}^{N} Q(CNF^i)$ the space of all the queries on the catalogues. We finally denote with $\wp(CNQ)$ the set of all subsets of queries over the catalogues, i.e.

$$\wp(CNQ) = \{cnq = \{q_1, \ldots, q_k\} \mid q_i = Q(CNF^{j_i})\}$$

In the example shown in Fig. 6.2, cnq = {(rock climbing = T AND hiking = T), (category = 3 AND parking = T AND cost ≤ 40)}. **CART** is defined as $CART = \wp(\bigcup_{i=1}^{N} CNF^i)$, i.e. an element cart ∈ CART is subsets of products: cart = $\{p_1, \ldots, p_k\}$ such that $\mid p_i \in CNF^{j_i}$. In the quoted example, cart = {(Canazei, T, T, T, F), (Molveno, T, T, F, T), (hotel, Irma, 3, 35, T)}.

## 6. Product Recommendation

As discussed previously, Trip@dvice supports a range of decision styles. In this section we describe two of them: the single-item iterative selection and ‘seeking for inspiration’. The first is designed for a user who has some rather precise needs and wants, and who wants to search the available options driven by these requirements. The second is designed for users who would rather browse the options and get inspired by the alternatives before taking some decision and focus on some particular products. A third recommendation functionality, named travel completion, which is aimed at completing a partial travel plan, is not shown here for lack of space.

### 6.1 Single-item iterative selection

The overall process supported by Trip@dvice, for the single-item iterative selection, is shown in Fig. 6.3. The user interacts with the recommender system by asking for recommendations about a product type (e.g. a destination). To simplify the notation, we will consider just one product space CNF, and *q* will denote the user’s query on this catalogue (1: AskRecommendation(*q*) in Fig. 6.3). The system replies to this query *q* either by recommending some products or, in case the query fails, by suggesting some query refinements. The RecEngine module manages the request. First, it invokes the Evaluate-Query function (2) of the Intelligent Query Manager module (IQM), by passing the query. This function searches the catalogue for products matching the

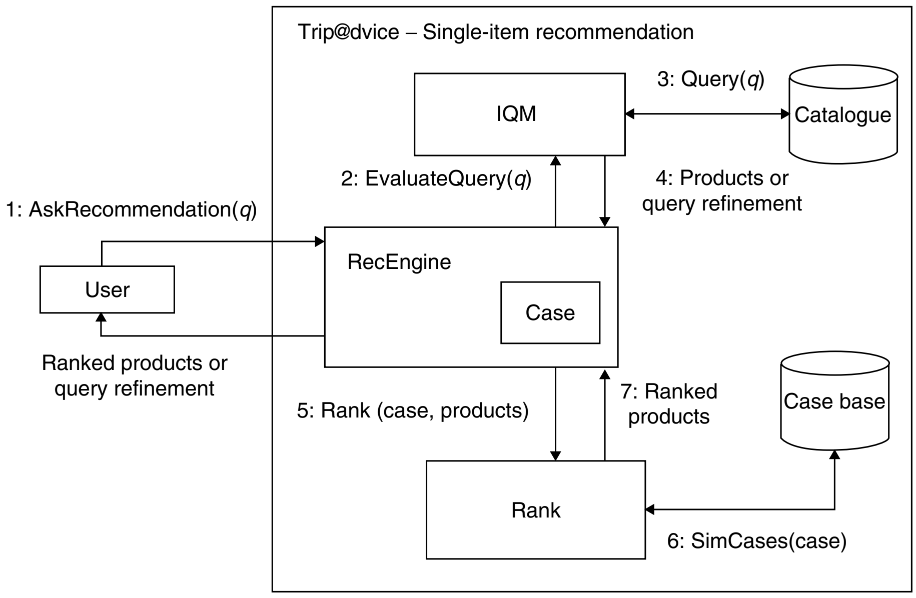


**Fig. 6.3.** Single-item recommendation with Trip@dvice.

```
CAT is the product catalog
q is the user's query
P = {p_1, …, p_k} the products selected by the user query q.
R = {(f_1, c_1, op_1), …, (f_m, c_m, op_m)}, op_j ∈ {add, modify, remove}, c_j is a
  constraint on feature f_j to be: added, modified or removed (op_j)

EvaluateQuery(q, CX)
1 P ← SearchCatalog(q, CX)
2 if Size(P) > threshold
3        R ← TightenQuery(q, CX)
4        return R
5 else if Size(P) = 0
6        R ← RelaxQuery(q, CX)
7        return R
8        else
9        return P
```

**Fig. 6.4.** The EvaluateQuery algorithm.

query. If too many or no product matches the input query *q*, then IQM analyses *q* and determines a set of query refinements for suggestion (Ricci *et al.*, 2002b). If there are too many results, three features, not already constrained in *q*, are selected and the user is asked to provide a value for one of them to narrow down the search result. Conversely, if no result can be found, the system explains to the user the cause of the failure, i.e. it lists those constraints that, if relaxed, would allow the query to return some results.

The EvaluateQuery algorithm is described in Fig. 6.4. It receives as input a query *q*, over a product catalog *CAT*. It returns the products *P* matching the query *q* or a set of query-refinement suggestions *R*. *R* is a set of triples, each containing a feature $f_k$, a constraint $c_k$ over the feature $f_k$ and the suggested operation $op_k$ (add, modify or remove from *q*). In line 1, the SearchCatalog function is invoked, passing the *q* query as parameter. The function searches through the catalogue for products matching *q*, and returns the set of matching products. Line 2 evaluates the size of the result set. If the number of selected products is above a certain threshold, the TightenQuery function is invoked (line 3). This function, using information related to the product catalogue data distribution (entropy and mutual information), returns a set of features (three) suggested to the user to further constrain the search. The user can choose one (or more) and provide a value (symbolic feature) or a range of values (numeric feature). Actually, TightenQuery returns a set of triples $(f_i, null, add)$, where $f_i$ is a feature and $c_i = null$, since the TightenQuery function cannot guess the exact value (or range of values) the user may want to specify for a suggested feature. Line 5 tests the empty result set condition. If the result set is empty, the RelaxQuery function is called (line 6). This function searches for those *q* modifications (constraint relaxation) that will allow *q* to retrieve some results. The suggested modifications are again returned as a set of triples $(f_i, c_i, op_i)$, where $op_i = remove$ for symbolic features and $op_i = modify$ for finite integers or real features, and $c_i$ represents the new (larger) range to be set. If neither relaxation nor tightening is invoked, the result set *P* is returned (line 9).

```
$RC = \{rc_1, \ldots, rc_{10}\}$ retrieved cases
$RP = \{rp_1, \ldots, rp_{10}\}$ products inside the reference cases
$c$ is the current case
$CB$ is the case base
$P = \{p_1, \ldots, p_k\}$ the products selected by the user query

Rank(c, p, CB)
1 $RC \leftarrow FindSimilarCases(c, CB)$
2 $RP \leftarrow ExtractReferenceProducts(RC)$
3 for each $p_i \in \{p_1, \ldots, p_k\} = P$
4         $Score(p_i) \leftarrow max_{j = 1 \ldots 10} \{Sim(c, rc_j)*Sim(p_i, rp_j)\}$
5         $P \leftarrow Sort\{p_1, \ldots, p_k\}$ according to $Score(p_i)$
6 return $P$
```

**Fig. 6.5.** The Rank algorithm.

When the number of items retrieved is satisfactory, the products are ranked by invoking the Rank method (Fig. 6.5). Rank receives the current case and the set of products selected by the EvaluateQuery function (the set *P* in Fig. 6.4). The current case is used to retrieve the set of *K* most similar cases, and the products contained in these retrieved cases are then used to rank the user-selected products. Finally, the ranked products are returned to the user. The RankProducts algorithm ranks the products retrieved by EvaluateQuery. Ranking is computed exploiting two similarity metrics: first, the case base is accessed to retrieve the *K* most similar cases (reference cases) to the current one. This similarity-based retrieval in principle can exploit all the case content, but an empirical evaluation has shown that it is more convenient to focus on the collaborative features and the cart content (cf. Section 7 for more details). Then, the products contained in the carts of the retrieved reference cases (reference products) are used to sort the products selected by the user's query. The basic idea is that among the products in the result set of the query one will get a better score if it is similar or equal to a product (of the same type) bought by a user with similar needs and wants. Figure 6.5 describes the algorithm in detail.

The Rank function receives the current case *c*, the set of products *P* retrieved by the function EvaluateQuery and the case base *CB*. It returns the products *P* ranked. First, it retrieves from *CB* the reference cases *RC* (line 1). In line 3, *RP* is defined as the products contained in the reference cases *RC*, having the same type of those in *P*.[2] In line 4, the *Score* of each product $p_i \in P$ is computed as the maximum value of $Sim(c, rc_j)*Sim(p_i, rp_j)$ over all the reference products $rp_j$ (the similarity functions are described in Section 7).

Computing the final product score as the multiplication of cases and products similarity mimics the collaborative-filtering (CF) approach, but there are some notable differences. First, differently from a standard CF approach, only the first nearest neighbour case is considered, i.e. the case

[2] For sake of simplicity we assume that each reference case $rc_i$ contains just one product $rp_i$ of the same type of the products to be ranked, but the same approach applies also when more products are contained in a case.

that yields the maximum value for *Sim*(*c*, $rc_j$)**Sim*($p_i$, $rp_j$). The rationale for this choice is that we can use the retrieved case to explain the score value to the user. Conversely, if we had used the classical CF approach (Breese *et al*., 1998), we would have described a weighted average score without any possibility to refer to the 'collaborating' cases. Second, we do not consider the votes given by other users to the product to be scored, as is common in CF. Conversely, we use the similarity of the product to be scored with products selected in a similar case (user session) as a sort of implicit vote: if another user in a similar session has chosen that product or a similar one, this is an indication that this product fits the needs that are shared by the current user and the previous user.

Let us consider the following simple example. Let us assume that the collaborative features in *CLF* are *TravelParty* (symbolic), *MinBudget* (numeric), *MaxBudget* (numeric), *Sports* (Boolean), *Relax* (Boolean), that the product features in *CNF* are *DestName* (symbolic), *RockClimbing* (Boolean), *Hiking* (Boolean) and that the collaborative feature values of the current case *c* are: *clf* = (*single*, 10, 40, *T*, *T*). Assume that a user query *q* has retrieved the products: $p_1$ = (*Predazzo*, *T*, *T*) and $p_2$ = (*Cavalese*, *T*, *T*). Then FindSimilarCases retrieves two cases $rc_1$ and $rc_2$, whose similarities with the current case *c* are *Sim*(*c*, $rc_1$) = 0.75 and *Sim*(*c*, $rc_2$) = 1. Let us further assume that $rc_1$ and $rc_2$ contain the product $rp_1$ = (*Campiglio*, *T*, *T*) and $rp_2$ = (*Cavalese*, *T*, *T*), respectively. Moreover, the product similarities are (see Section 7 for the similarity definition): *Sim*($p_1$, $rp_1$) = 0.66, *Sim*($p_1$, $rp_2$) = 0.66, *Sim*($p_2$, $rp_1$) = 0.66, *Sim*($p_2$, $rp_2$) = 1. The score of each $p_i$ is computed as the maximum of *Sim*(*cc*, $c_j$)**Sim*($p_i$, $rp_j$); thus: *Score*($p_1$) = *max*{0.75*0.66, 1*0.66} = 0.66, and *Score*($p_2$) = *max*{0.75*0.66, 1*1} = 1. Finally, $p_2$ is scored higher than $p_1$. The user, after having selected a first item (e.g. a destination) and added it to the cart, can iterate the process selecting another destination or another product or service among those offered in the supplier catalogues. It is worth noting that in these next iterations even the content of the cart (the products selected at that point) are used to compute the similarity with past cases. In this way the decisions taken by the user, at a certain point, impact on the computation of similar cases and in turn on the ranking of the products recommended in the next stages.

## 6.2 Seeking for inspiration

The second decision style supported by Trip@dvice is seeking for inspiration. This is designed for users who would rather 'look at' the options and get inspired by the alternatives before taking some decision and focus on some particular products. The recommendation proceeds according to the following loop (see Fig. 6.6):

- **First Retrieval.** The process starts with a seed case *c*, which could be either the current case or a random case taken from the case base. It is the current case if the user has already created one case in the previous stages of the interaction, for instance by selecting a destination and adding it

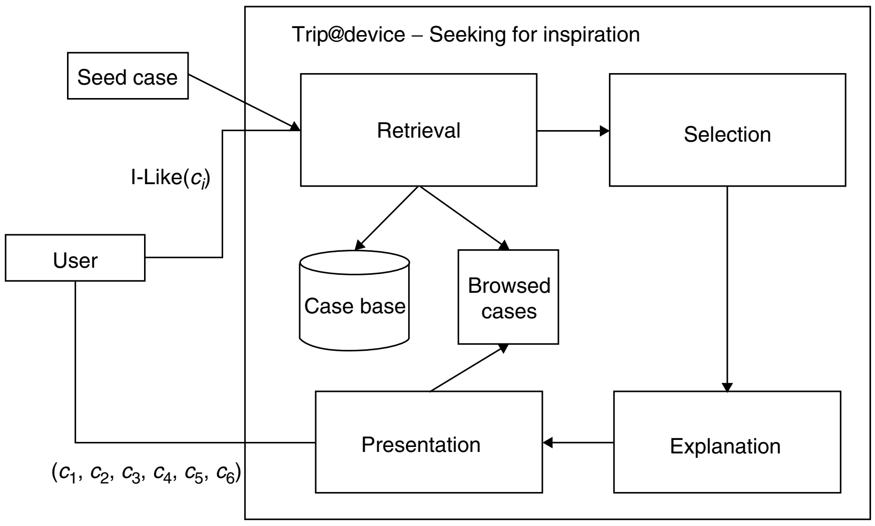


**Fig. 6.6.** Seeking for inspiration recommendation cycle.

to the cart using the single-item recommendation function. Conversely, if the interaction with the recommender starts with this first retrieval, the recommender does not know anything about the user and can only make an initial random guess of what could be a good travel for the user. The retrieval module searches for the *M* most similar cases in the case base and passes this set to the next module, the Selection. We call *Ret*(1, *M*) this retrieval set to stress that this is the first retrieval set and its cardinality is *M*. The parameter *M* is typically a large number, possibly equal to the cardinality of the underlying case base. The Retrieval module is aimed at selecting past cases (travel plans) that are similar to the current one, hence potentially applicable even in the current situation.

- **Case Selection.** In the second step the *M* cases retrieved from the memory are analysed to select a very small subset of candidates to be presented to the user. In the DieToRecs system, for instance (see Section 8), six cases are selected. The selection is accomplished by a greedy algorithm that iteratively chooses the six cases starting from the seed case and puts them in the result set. The case added at each iteration is the one that minimizes the sum of the similarities between itself and the cases already in the result set. For instance, the second case added to the result set is $c_2 = argmin_{c_2 \in Ret(1, M)} \{Sim(c_1, c_2)\}$. Similarly, the third case is defined as $c_3 = argmin_{c_3 \in Ret(1, M)} \{Sim(c_1, c_3) + Sim(c_2, c_3)\}$. The selected cases $\{c_1, \ldots, c_6\}$ are then passed to the explanation module.
- **Explanation.** This module is aimed at providing a rationale for the selected cases. The explanation computed in this module stresses the differences of the selected cases, i.e. identifies the attributes that are peculiar to one case and are not common among the six selected cases. The algorithm considers each case and computes, for each feature *f* in a subset of all the

case features (the most relevant for the explanation), a diversity count value $dc(f, c)$. This is the number of times the feature $f$ has a different value in the other five cases. For instance, if $f_1$ = *accommodation-type* feature, has value 'hotel' in $c_1$ and 'apartment' in the others, then $dc(f_1, c_1) = 5$, and $dc(f_1, c_i) = 1$ for $i = 2, 3, 4, 5, 6$. Having repeated this for all the explanation-relevant features, for each case the three most peculiar features, i.e. those with higher diversity count, are selected. The explanation module then passes the six cases and the selected features to the presentation module.

- **Presentation.** The goal of this module is to present the six selected cases along with an explanation of their peculiarities. The explanation basically argues that a travel case is potentially interesting because it possesses some characteristics (features) that are not shared by other cases. Referring to the example above, the first case will be described, together with other descriptors, as a travel with a Hotel accommodation. Moreover, the presentation component selects some images, taken from the products or services included in the travel, to illustrate pictorially the case content. At this point the user is supposed to browse the offers, and eventually provide a positive feedback on one of these cases. This feedback, i.e. the selected case, is given as input for a possible successive retrieval.
- **Retrieval.** The second time the retrieval is called, the procedure described above is repeated, with some notable changes. The seed case is now the case that received positive feedback from the user, and the number of cases retrieved from the case base, $M$ in the first retrieval, is decreased by a factor $0 < \lambda < 1$, i.e. the new retrieval set is $Ret(2, \lambda M)$. The rationale for decreasing the number of retrieved cases is to better focus the retrieval around the input case, since at this new stage we must count on the positive evaluation of the user on the selected case. This $\lambda$ parameter is tuned in such a way that $M\lambda^{l+2\sigma} < 6$, where $l$ is the average interaction length (as observed in the experiments with real users) and $\sigma$ is the standard deviation. In this way there is no further diversity selection after a typically maximal number of iterations $(l + 2\sigma)$.

## 7. Similarity Measures

The recommendation functions described above heavily exploit similarity-based computation and similarity-based retrieval. The first refers to the computation of a similarity measure for two objects, either cases or products. The second refers to the extraction from the case base of a set of cases similar to a given one. In this section we shall focus on the similarity computation and will describe the general similarity metric defined in Trip@dvice. In this chapter we do not discuss the particular implementation of the similarity-based retrieval. The reader can assume that a standard linear scan of the case base is implemented (see Arslan *et al*., 2002) for further details on this topic). As we observed in Section 5, a case is a structured object, i.e. a case $c$ can be decomposed into three subcomponents: the collaborative features, the content queries and the cart. In this section, to make the discussion more general, we shall simply assume that a case $c$ is a rooted and ordered tree $G_c(V_c, E_c)$, which we denote with $V_c$, the nodes, i.e. the set of all subcomponents of $c$, and with $E_c$, the set of edges. An edge $(p, q)$ is in $E_c$ if the component $p$ contains the component $q$. In this case, we say that $q$ is an out-neighbour of $p$ and we shall denote with $O(v)$ the set of out-neighbours. The out-neighbours of a node are ordered, hence we shall indicate with $v = (v_1, \ldots, v_n)$ the out-neighbours of $v$, instead of using the notation $O(v) = \{v_1, \ldots, v_n\}$. For instance, considering the case $c = (clf, cnq, cart)$, we have $(c, clf) \in E_c$ and $clf$ is an out-neighbour of $c$. Moreover, each node $v$ is labelled by a type information $\tau(v)$, which describes the type (with a name) of the case component represented by the node. For instance, in the case $c$ as above, $\tau(clf) = collaborative\ features$.

In addition to the type $\tau(v)$ of a node $v$, it is defined by the metric-type $\mu(v)$ of the node. The metric-type determines how the metric is computed at the node and depends on: (i) the type of the node; and (ii) the metric-type of the out-neighbours. First of all, let us define a leaf node as a node that cannot have any out-neighbour; then we have the following cases:

- $\mu(v) = vector$ means that if $O(v)$ is not empty, then it is an ordered list of nodes with fixed length.
- $\mu(v) = set$ means that if $O(v)$ is not empty, then it is an unordered set of nodes.
- $\mu(v) = heterogeneous$ if and only if $O(v)$ contains leaf nodes (i.e. nodes that have no out-neighbours).

Leaf nodes can have the following metric-type: hierarchical, numeric, jaccard, modular, symbol, range.

- $\mu(v) = hierarchical$ means that $v$ is a vector of features with a precise hierarchical relationship. For instance, a node of type 'geo-area' is a vector of four symbols (country, region, area, village).
- $\mu(v) = numeric$ means that $v$ is a numeric feature (float or integer).
- $\mu(v) = jaccard$ means that $v$ is a vector of Boolean features.
- $\mu(v) = modular$ means that $v$ is a modulo. For instance, the hours are integer modulo 24 (e.g. $23 + 3 = 1$).
- $\mu(v) = symbol$ means that the value of $v$ is an arbitrary symbol in a given symbol set (e.g. a Boolean feature).
- $\mu(v) = range$ means that $v$ is a range of numbers, e.g. (minPrice, maxPrice) constraints in a hotel query constraint.

Before explaining how the distance metric is computed, let us extend our case example $c = (clf, cnq, cart)$ to describe the above-mentioned node metric-types. For sake of simplicity let us describe a simpler case $c = (clf, cart)$, i.e. where the content queries are not considered. Then let us assume that $clf = (single, 1, \quad [20, 40], \quad [0, 1, 0], \quad [italy, trentino, fassa, null]), \quad cart = (ds, as)$, $ds = (de_1, de_2), \ as = null \ \ de_1 = vigo, de_2 = pozza$. Then $c$ has type *case* and metric-type *vector*. *clf* has type *collaborative features* and metric-type *vector*. *cart* has type *cart* and metric-type *vector*, *ds* and *as* have type *destinations* and *accommodations* and metric-type *set*, $de_1$ and $de_2$ have type *destination* and metric type *symbol*. The first out-neighbour of *clf* (*single*) has type *travel party* and metric-type *symbol*; the second (1, i.e. January) has type *month period* and metric-type *modulo*; the third ([20, 40]) has type *budget* and metric-type *range*; the fourth ([0, 1, 0]) has type *interests* and metric-type *jaccard* (represents the presence or absence of three interests, e.g. *sport*, *culture*, *wellness*); the fifth ([*italy*, *trentino*, *fassa*, *?*]) has type *geo-area* and metric type *hierarchical*.

The cases contained in a cases base all share the same structure, i.e. the out-neighbours of a node with given type have always the same type. In more formal way, if $c = G_c(V_c, E_c)$ and $c' = G'_c(V'_c, E'_c)$, then $c = (v_1, \ldots, v_n)$, $c' = (v'_1, \ldots, v'_n)$ and $\tau(v_1) = \tau(v'_1), \ldots, \tau(v_n) = \tau(v'_n)$ and then recursively down to the leaves of the case. Moreover, if $v$ is a node of $c$ and its metric-type is *set* (i.e. $\mu(v) = set$), then $v = (v_1, \ldots, v_{n(v, c)})$, all the $v_i$ have the same type and metric-type and we denote with $n(v, c)$ the fact that the number of out-neighbours of $v$ is case-dependent. A typical example is the node representing the activities contained in a case. All the out-neighbours are activities, represented in the same way (metric-type), but the number of activities varies case by case.

We can now define the case metric $d(c, c')$ by recursively applying the distance computation on the subcomponents of the given nodes. It starts from the two nodes $c$, $c'$ and applies a different computation according to the node metric-type. More formally, if $v = (v_1, \ldots, v_n) \in V_c$ and $v' = (v'_1, \ldots, v'_m) \in V'_c$, then

- If $\mu(v) = \mu(v') = vector$, then $n = m$ and

$$d(v, v') = \frac{1}{\sum_{i=1}^{n} w_i(\tau(v))} \sum_{i=1}^{n} w_i(\tau(v)) d(v_i, v'_i)$$

where $0 \geq w_i(\tau(v)) \geq 1$ are weights depending on the type of node.

- If $\mu(v) = \mu(v') = set$, then

$$d(v, v') = \frac{1}{n * m} \sum_{i=1}^{n} \sum_{j=1}^{m} d(v_i, v'_j)$$

- If $\mu(v) = \mu(v') = heterogeneous$, then $n = m$ and

$$d(v, v') = \frac{1}{\sqrt{\sum_{i=1}^{n} w_i(\tau(v))}} \sqrt{\sum_{i=1}^{n} w_i(\tau(v)) d(v_i, v'_i)^2}$$

- If $\mu(v) = \mu(v') = hierarchical$, then $n = m$ and

$$d(v, v') = 1 - (\max_i \{i : v_i = v'_i\} / n)$$

- If $\mu(v) = \mu(v') = numeric$, then $v, v' \in R$ and

$$d(v, v') = \begin{cases} \dfrac{|v - v'|}{4\sigma} & if \quad \dfrac{|v - v'|}{4\sigma} \leq 1 \\ 1 & else \end{cases}$$

where $\sigma$ is the standard deviation of the values of all the nodes $x$ of type $\tau(v)$ in the case base (or in the catalogue).

- If $\mu(v) = \mu(v') = jaccard$, then $n = m$ and

$$d(v, v') = \frac{N_{01} + N_{10}}{N_{01} + N_{10} + N_{11}}$$

where $N_{01}$ is the number of times $v_i = 0$ and $v'_i = 1$. The other numbers are defined similarly.

- If $\mu(v) = \mu(v') = modular$, then $v, v' \in R$ and
  $d(v, v') = (arg_x\ min\ \{v + x = v'\ (mod\ k), in\ v' + x = v\ (mod\ k)\})/k$

where $k$ is the modulo of the nodes $v, v'$. Hence, for instance, if $k = 12$ (twelve months), then the distance between 11 (November) and 1 (January) is the minimum $x/12$ s.t. $11 + x = 1$ ($mod$ 12) and $1 + x = 11$ ($mod$ 12), therefore $x = 2$ and the distance is 2/12.

- If $\mu(v) = \mu(v') = symbol$, then

$$d(v, v') = \begin{cases} 0 & if\ v = v' \\ 1 & if\ v \neq v' \end{cases}$$

- If $\mu(v) = \mu(v') = range$, then $v = [v_1, v_2]$ and $v' = [v'_1, v'_2]$

$$d(v, v') = \frac{|v_1 - v'_1| + |v_2 - v'_2|}{2 * 4\sigma}$$

where $\sigma$ is the standard deviation of the values of all the nodes $x$ of type $\tau(v)$ in the case base and in the catalogue.

It must be noted that if a node $v$ is null, i.e. it is not defined in the case (e.g. it is unknown), then the distance of that node with any other node is the maximal distance, i.e. 1.

Let us now take an example to illustrate how the case distance is computed. Referring to the previously mentioned example, let us imagine that:

$c = (clf, cart)$
$clf = (single, 1, [20, 40], [0, 1, 0], [italy, trentino, fassa, ?])$
$cart\ (ds, as), ds = (de_1, de_2), as = null, de_1 = vigo, de_2 = pozza$
$c' = (clf', cart')$

$clf' = (couple, 2, [30, 40], [1, 1, 0], [italy, trentino, fassa, campitello])$
$cart'(ds', as'), ds = (de'_1), as = null, de'_1 = mazzin.$

Then we have:

$d(c, c') = [1/(w(clf) + w(cart))]\ (w(clf)\ d(clf, clf') + w(cart)\ d(cart, cart'))$
$d(clf, clf') = [1/(\sqrt{\{\sum_{i=1}^{5} w_i(\tau(clf))\}})]\sqrt{\{\sum_{i=1}^{5} w_i(\tau(clf))\ d(clf_i, clf'_i)^2\}}$
$d(clf_1, clf'_1) = 1$
$d(clf_2, clf'_2) = 1/12$
$d(clf_3, clf'_3) = 10/(8\sigma)$
$d(clf_4, clf'_4) = 1/2$
$d(clf_5, clf'_5) = 1 - 3/4$
$d(cart, cart') = [1/(w(des) + w(acc))]\ (w(des)\ d(ds, ds') + w(acc)\ d(acc, acc'))$
$d(des, des') = 1/2\ (d(de_1, de'_1) + d(de_2, de'_1)) = 1/2\ (1 + 1)$
$d(as, as') = 1$

Using the same approach the metric can compute the distance of two items in a catalogue. Hence, if, for instance, the destination is modelled as a heterogeneous vector of the features (nodes) $de = ([italy, trentino, fassa, campitello], 1448, [1, 1, 0])$, then in computing the distance between this destination and another destination the same rules as above apply. Finally, the similarity of two cases $c$, $c'$ (or two items) is defined as

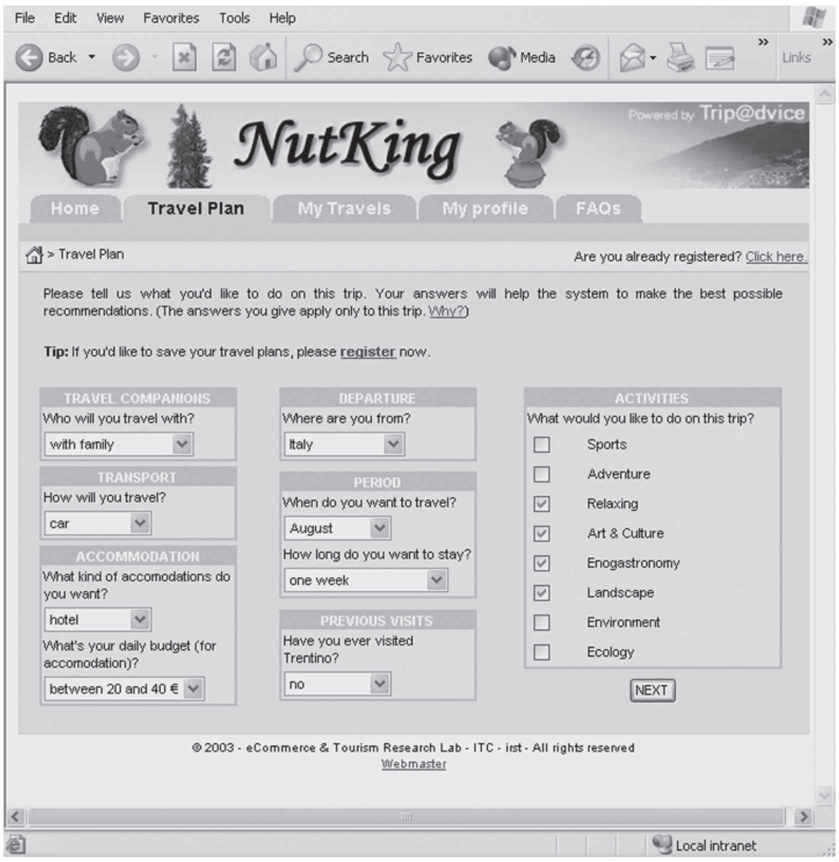
$$Sim(c, c') = exp(-d(c, c'))$$

## 8. Prototypes

### 8.1 NutKing

In the NutKing prototype, Trip@dvice supports the interaction mimicking a typical counselling session in a real travel agency. The user initially defines the major trip goals and constraints entering some preferences as shown in Fig. 6.7. These are general features of the proposed trip including travel companions, budget and means of transportation. These wishes are used both to recommend products, exploiting other users' trips, and to set some default constraints in successive query-forms. After this initial step, the user can start building his or her itinerary by searching for travel products, as required. This could be a destination, an accommodation, an event or an attraction (available catalogues).

Assume that the user is looking for a destination; the system offers the user a classical Query-By-Example interface (left side of Fig. 6.8), where he or she can set constraints on the product features. If the query retrieves too many products to be examined, Trip@dvice asks the user to provide some additional (alternative) constraints. Another possible situation may occur when no products satisfy the query; in this case the system proposes some alternative query's relaxations, which, if applied, would provide the user with a suitable result set (Fig. 6.8).



> Travel Plan

Are you already registered? Click here.

Please tell us what you'd like to do on this trip. Your answers will help the system to make the best possible recommendations. (The answers you give apply only to this trip. Why?)

**Tip:** If you'd like to save your travel plans, please **register** now.

TRAVEL COMPANIONS
Who will you travel with? with family

TRANSPORT
How will you travel? car

ACCOMMODATION
What kind of accomodations do you want? hotel
What's your daily budget (for accomodation)? between 20 and 40 €

DEPARTURE
Where are you from? Italy

PERIOD
When do you want to travel? August
How long do you want to stay? one week

PREVIOUS VISITS
Have you ever visited Trentino? no

ACTIVITIES
What would you like to do on this trip?
- ☐ Sports
- ☐ Adventure
- ☑ Relaxing
- ☑ Art & Culture
- ☑ Enogastronomy
- ☑ Landscape
- ☐ Environment
- ☐ Ecology

NEXT

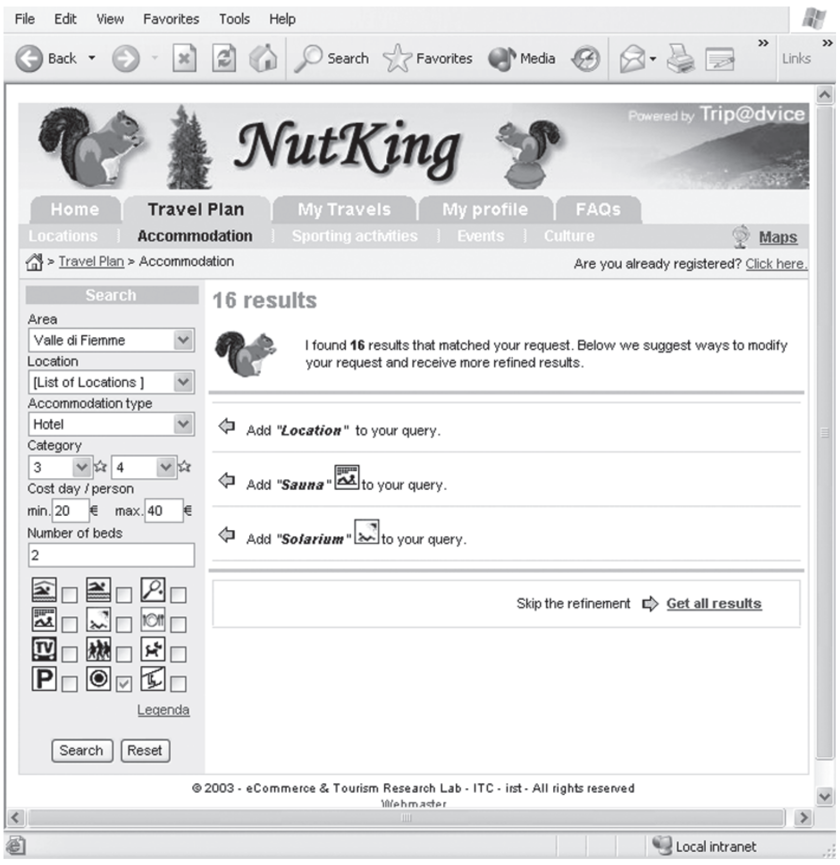
© 2003 - eCommerce & Tourism Research Lab - ITC - irst - All rights reserved
Webmaster

**Fig. 6.7.** Setting general travel wishes.

At the end of this interaction a list of recommended products is shown (Fig. 6.9). All of these satisfy the (explicit) user constraints, but furthermore the products most similar to those selected by other users with similar general wishes are ranked first. By clicking on the small trees icon on the right side of the product, the user can obtain an explanation of the recommendation. Then the user can add the selected product to his or her travel bag and proceed by adding other items to the trip. Finally, the user can print the complete itinerary in a brochure-like style.

## 8.2 DieToRecs

DieToRecs recommender system extends in many ways the NutKing prototype described above. In addition to the single-item iterative recommendation function, which has been illustrated in Section 8.1, DieToRecs includes the travel completion and seeking for inspiration functions. The seeking for


File Edit View Favorites Tools Help
Back Search Favorites Media Links
NutKing
Powered by Trip@dvice
Home Travel Plan My Travels My profile FAQs
Locations Accommodation Sporting activities Events Culture Maps
Travel Plan > Accommodation
Are you already registered? Click here.
Search
Legenda
Search Reset
Skip the refinement Get all results
Local intranet

Area: Valle di Fiemme
Location: [List of Locations ]
Accommodation type: Hotel
Category: 3 – 4
Cost day / person: min. 20 € max. 40 €
Number of beds: 2

**16 results**

I found **16** results that matched your request. Below we suggest ways to modify your request and receive more refined results.

Add ***"Location"*** to your query.

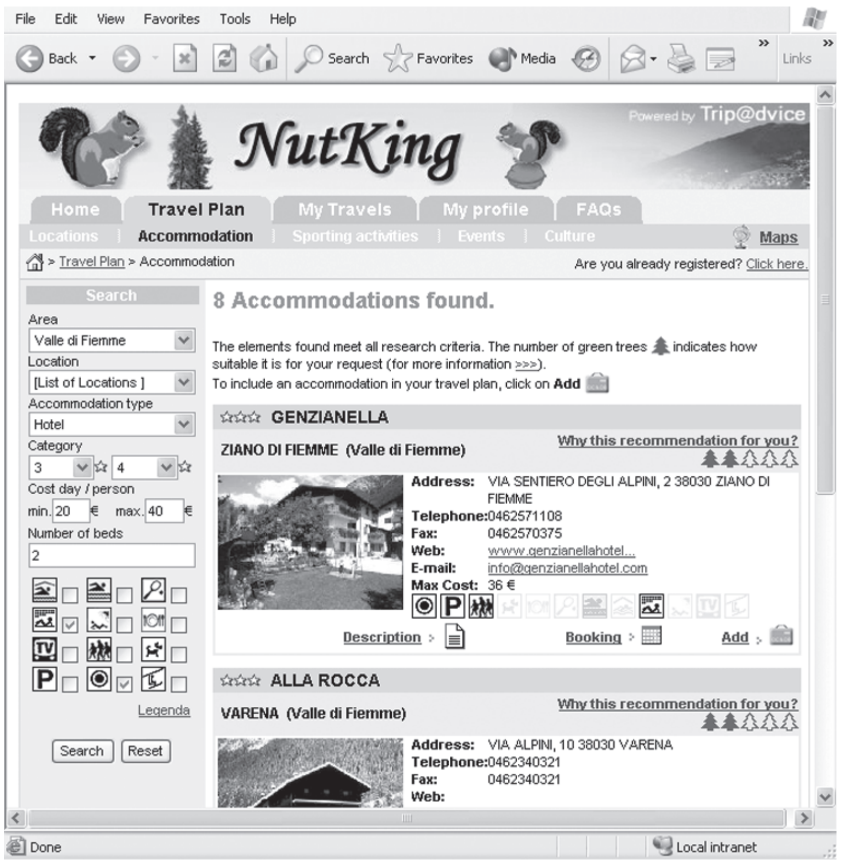
Add ***"Sauna"*** to your query.

Add ***"Solarium"*** to your query.

© 2003 - eCommerce & Tourism Research Lab - ITC - irst - All rights reserved
Webmaster

**Fig. 6.8.** Suggesting query relaxations.

inspiration recommendation function is designed for a tourist who is less constrained by precise needs and wants and is more prone to consider 'good proposals'. Hence, the selection process is more system-driven in the form of pictorial representations of former trips. In seeking for inspiration, users are not requested to specify general travel or travel item preferences to get some recommendations, but they immediately get some travel recommendations by the systems (see Fig. 6.10). The system proposes six complete travels, represented by images about the destination and the accommodation and the two most important and characterizing features. Users can then access detailed information about each proposal and the included tourist products, like details about the suggested accommodations, descriptions about the destinations and attractions. After examining the proposals or, just being inspired by the shown images, the users can get six new different proposals simply by expressing their interest about one of them; the system, exploiting the fact that the users are interested in that particular alternative, proposes six other alternatives that are more focused on the user wishes. In this way, the users can browse the catalogue space without explicitly specifying constraints and


File Edit View Favorites Tools Help
Back Search Favorites Media Links
NutKing
Powered by Trip@dvice
Home Travel Plan My Travels My profile FAQs
Locations Accommodation Sporting activities Events Culture Maps
Search
Legenda
Search Reset
Description Booking Add
Done
Local intranet

> Travel Plan > Accommodation
Are you already registered? Click here.

Area: Valle di Fiemme
Location: [List of Locations ]
Accommodation type: Hotel
Category: 3 – 4
Cost day / person: min. 20 € max. 40 €
Number of beds: 2

8 Accommodations found.

The elements found meet all research criteria. The number of green trees indicates how suitable it is for your request (for more information >>>).
To include an accommodation in your travel plan, click on Add

GENZIANELLA
ZIANO DI FIEMME (Valle di Fiemme)
Why this recommendation for you?
Address: VIA SENTIERO DEGLI ALPINI, 2 38030 ZIANO DI FIEMME
Telephone: 0462571108
Fax: 0462570375
Web: www.genzianellahotel...
E-mail: info@genzianellahotel.com
Max Cost: 36 €

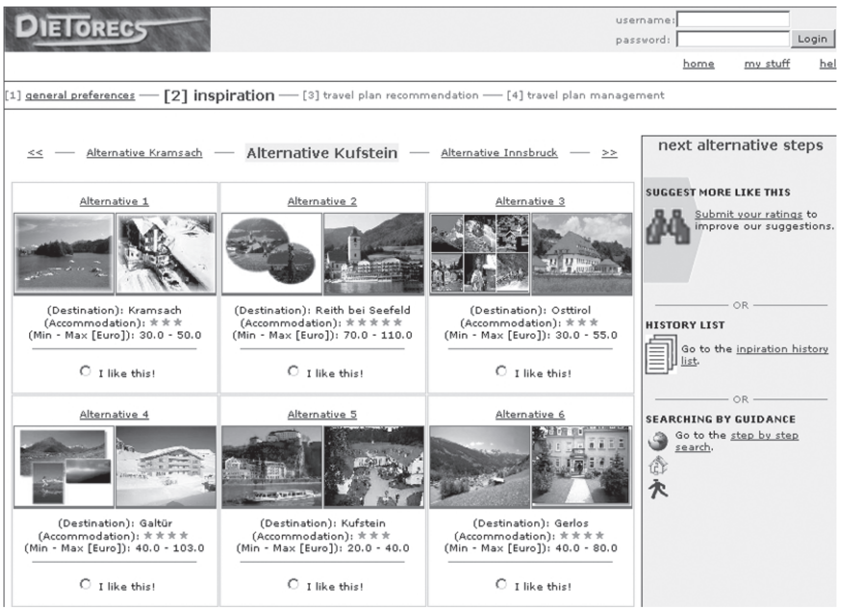
ALLA ROCCA
VARENA (Valle di Fiemme)
Why this recommendation for you?
Address: VIA ALPINI, 10 38030 VARENA
Telephone: 0462340321
Fax: 0462340321
Web:

**Fig. 6.9.** Showing the final recommendation.

wishes. When the users find the proposal that suits their needs, they can add the suggested products to their personal travel bag.

## 9. Discussion and Conclusion

In this chapter we have presented Trip@dvice, an approach to products and services recommendation that exploits in a novel way the classical CBR learning loop. Trip@dvice integrates CBR and interactive query management to support a number of decision styles. The proposed methodology can help the user to specify a query that filters out unwanted products in electronic catalogues (content-based) and can as well serve less structured search process that are mostly driven by browsing options rather than imposing restrictions on options. The approach has been applied to a couple of web travel applications and it has been empirically evaluated with real users. We have shown that the proposed approach: (i) reduces the number of user queries; (ii) reduces the number of browsed products; (iii) increases user satisfaction;


DIETORECS
username:
password:
Login
home
my stuff
hel
[1] general preferences — [2] inspiration — [3] travel plan recommendation — [4] travel plan management
<< — Alternative Kramsach — Alternative Kufstein — Alternative Innsbruck — >>
next alternative steps
SUGGEST MORE LIKE THIS
Submit your ratings to improve our suggestions.
OR
HISTORY LIST
Go to the inpiration history list.
OR
SEARCHING BY GUIDANCE
Go to the step by step search.
I like this!

Alternative 1
(Destination): Kramsach
(Accommodation): ★★★
(Min - Max [Euro]): 30.0 - 50.0

Alternative 2
(Destination): Reith bei Seefeld
(Accommodation): ★★★★★
(Min - Max [Euro]): 70.0 - 110.0

Alternative 3
(Destination): Osttirol
(Accommodation): ★★★
(Min - Max [Euro]): 30.0 - 55.0

Alternative 4
(Destination): Galtür
(Accommodation): ★★★★
(Min - Max [Euro]): 40.0 - 103.0

Alternative 5
(Destination): Kufstein
(Accommodation): ★★★
(Min - Max [Euro]): 20.0 - 40.0

Alternative 6
(Destination): Gerlos
(Accommodation): ★★★★
(Min - Max [Euro]): 40.0 - 80.0

**Fig. 6.10.** Seeking for inspiration user interface in DieToRecs.

(iv) produces good recommendations; and that (v) the selected items are found first on the ranked list.

We have already talked about the general flow of the CBR cycle in Section 2 and explained why the classical approach has some limitations when applied to complex products recommendation. To tackle these problems we designed the Trip@dvice methodology. This yields a change in the CBR loop, which is described below. The description here refers to the single-item recommendation only. A similar description would be necessary for the other decision styles, but it is here omitted for lack of space.

The left part of Fig. 6.11, i.e. (a), covers the mentioned classical CBR cycle, plus an extra step '6. Iterate', which is particular to our approach. The boxes are the points where we have introduced some changes to the classical framework. The separation between the left and right parts of Fig. 6.11, i.e. (a) and (b), underlines one of the main differences between the Trip@dvice methodology and the basic CBR cycle.

The travel items selected in a recommendation session, i.e. the items that form the travel cart, are contained in electronic catalogues (products database). The case base provides information about good bundling of products and is therefore used for acquiring this knowledge and for ranking items selected in the catalogues as we have explained throughout this document. The catalogues are exploited for obtaining up-to-date information about currently available products and services.

- **Retrieve.** Retrieval is done considering also similarities between subcomponents of cases. Section 7 explains how we obtain the set of retrieved

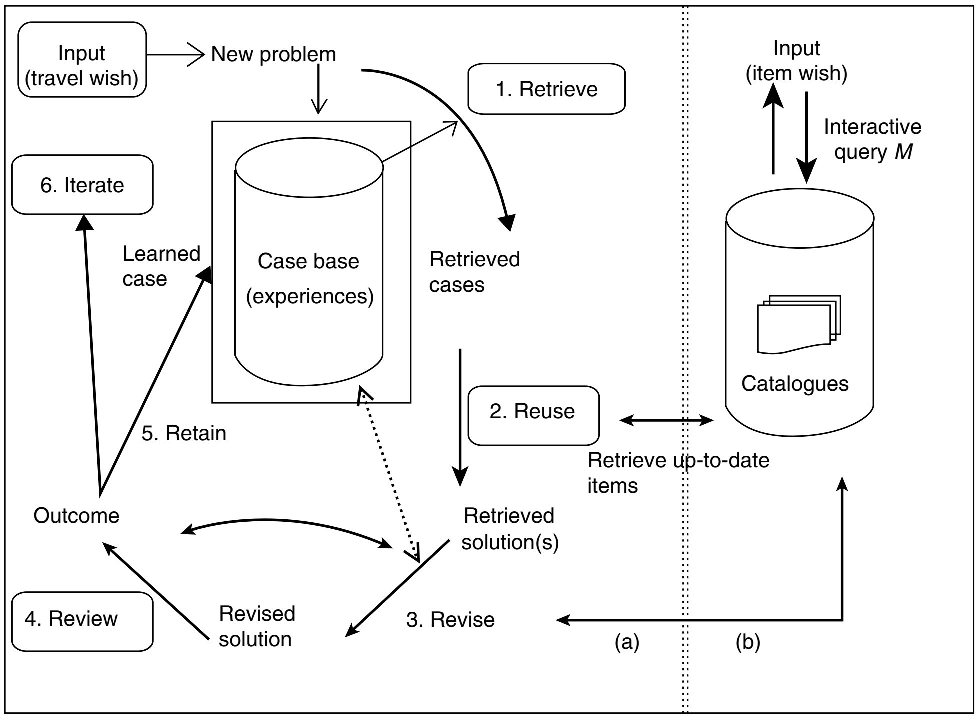


**Fig. 6.11.** Trip@dvice learning cycle.

cases that are called 'Reference Set' to stress their importance in all the recommendation stages.

- **Reuse and Revise.** We do reuse the previously experienced cases, but in a different fashion. Mainly we use the past experiences to get the required knowledge for aggregations of travel items. Instead of reusing exactly the same recommendations as in the past (solutions to the previously stated problems), we always recommend up-to-date items to the user. Therefore we deliver to the user already revised (updated) recommendations.
- **Review.** The last decision is always made by the user. Trip@dvice users are always allowed to reconfigure the recommended bundles. Using the experienced successful cases, the system opens a new door to the user. Having adapted and updated versions of these templates, the user may replace, add or remove one (or more) of the items in the recommended travel. In this view, we manage the Review step in a mixed-initiative way, in collaboration with the user.
- **Retain and Iterate.** When the user accepts the outcome (the final version of the recommendation delivered to the user), this action forms a new learned case, i.e. a complex problem and its solution to be used for further recommendations, which is the point at which a standard CBR cycle completes the loop. But we need to underline that Trip@dvice CBR cycle

may not be completed at this stage, if the user does not think that the travel bag is completed yet.
- **Input.** The first remark here is that in Trip@dvice, the user input is gathered in a structural way (Travel Wish and Item Wish). One can say that the 'Input' is the problem description for a classical CBR application. But, as the 'Iterate' step states, the underlined loop in Fig. 6.11 may take place repeatedly. Each time the user starts a new loop, what he already has in his travel bag defines a part of the new problem. Therefore, the notion of complete travel bag makes the problem description (i.e. input) a little fuzzy and a cloudy concept.

Our framework extends the basic CBR model presented in Section 2. First, the scoring and ranking mechanism is determined not only by the user input conditions and the product catalogues but also on the memory of past interaction sessions (case base). Therefore, referring to the example quoted above, we can rank and differentiate even the hotels that satisfy the query conditions ('cost = 100 and location = Rome') by measuring the similarity of the hotels that satisfy the user query with those chosen in the past for similar travels. In CABATA all the hotels that logically satisfy the query conditions have maximal similarity with the query and therefore are not differentiated in the recommendation.

The second major aspect that differentiates our system from standard CBR recommender systems refers to the case structure itself. Our case base is made up of complex structures of more elementary components (travel products). This case base is used both when the elementary components retrieved from a catalogue must be ranked and when similarity-based retrieval is performed at the case level itself. Standard CBR recommenders only perform this last function, i.e. similarity-based retrieval on the case base.

That points out a third element, i.e. the usage of both a case base and catalogues. The case base provides complex knowledge about travel products bundling and user navigation preferences, whereas the catalogues of products, accessed as external systems, provide up-to-date information on the available travel products.

A number of open issues are still to be addressed. We would like to mention first the adaptation of the similarity metric. This depends on a number of parameters (weights) and it is applied in a range of recommendation functions and, more internally, in steps of the recommendation process. General knowledge can be exploited to design a default weighting scheme but a finer tuning is necessary to improve the recommendation quality. Another major issue is related to the case base management. Currently the methodology has been tested with small case bases. An operational system that includes Trip@dvice will produce a large case base in a few days of proper usage. Hence, the problem of discarding unnecessary cases will become more and more relevant. Even if this is not a new problem in CBR, the customization of general techniques to the peculiar case structure and CBR application poses interesting and unsolved issues.

## Chapter Summary

In this chapter we provided the methodological foundations and the rationale for approaching travel destination recommendation as a problem-solving activity. The CBR was presented both as a cognitive plausible approach and as an integrated paradigm to build advisory systems. We then described an integrated solution called Trip@dvice that employs CBR. Finally, we discussed the most important findings of various validation activities and future research.

## Acknowledgements

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). The authors would like to thank all the partners of the DieToRecs (http://DieToRecs.itc.it/) project for their original ideas and comments on the Trip@dvice methodology.

### Author Queries

AQ1 Please specify whether it is Burke 2000a or b.

AQ2 In Fig. 6.2, "HOTEL where category = 3 AND parking = T AND cost < 40" has been changed to "HOTEL where category = 3 AND parking = T AND cost ≤ 40" as per text (<40 has been changed to ≤40). Please check.