Conversational Case-based Recommendations Exploiting a Structured Case Model

Quang Nhat Nguyen and Francesco Ricci

Free University of Bozen-Bolzano
{qnhatnguyen,fricci}@unibz.it

Abstract. There are case-based recommender systems that generate personalized recommendations for users exploiting the knowledge contained in past recommendation cases. These systems assume that the quality of a new recommendation depends on the quality of the recorded recommendation cases. In this paper, we present a case model exploited in a mobile critique-based recommender system that generates recommendations using the knowledge contained in previous recommendation cases. The proposed case model is capable of modeling evolving (conversational) recommendation sessions, capturing the recommendation context, supporting critique-based user-system conversations, and integrating both ephemeral and stable user preferences. In this paper, we evaluate the proposed case model through replaying real recommendation cases recorded in a previous live-user evaluation. We measure the impact of the various components of the case model on the system's recommendation performance. The experimental results show that the case components that model the user's contextual information, default preferences, and initial preferences, are the most important for mobile context-dependent recommendation.

1 Introduction

Product suggestions provided by recommender systems (RSs) are useful when users are overwhelmed by a large number of options to consider or when they do not have enough knowledge about a specific domain to make autonomous decisions [2].

Case-based recommender systems [3, 7] are knowledge-based RSs [5] that exploit case-based reasoning [1] to generate personalized recommendations. A case-based RS maintains a set of cases (i.e., a case base) of previously solved recommendation problems and their solutions. In many case-based RSs (e.g., [4, 8, 16, 9]), the authors assume that the case base is the product catalogue, i.e., the solutions of the recommendation problem, and the "problem" is the user's query that is essentially a partial description of her desired product. In this approach, the case does not store the user needs/preferences or the context that originates the recommendation problem, and the solution of the case is the case itself, i.e., the product recommended. Hence, this product-based case modeling approach does not capture the link between the problem, i.e., the user's preferences and the recommendation context, and the solution, i.e., the user's selected product, as in traditional CBR systems. Moreover,

here no learning takes place, since no new knowledge is stored after a problem is solved, and no adaptation of a previously solved case is done.

Besides the product-based case modeling approach, there have been a few attempts to exploit more extensively the CBR methodology in RSs. In the approach presented in [14], a case models a user's interaction with the system in the recommendation session, and consists of four components: the collaborative features, which are the user's general characteristics, wishes, constraints, and goals; the queries executed during the recommendation session, representing the user's additional preferences and constraints on products; the selected products; and the ratings (evaluations) given to the selected products. To build the recommendation list, the system first retrieves the products that satisfy the user's current query, and then ranks the retrieved products according to their similarity to the products selected in past similar recommendation cases. In another approach, discussed in [17], a case is composed of a user's query (the problem part) and the selected product (the solution part). Given a user's query, the system's construction of the recommendation list consists of two steps. First, the system builds the retrieval set that is a union of 1) the set of the products in the product catalogue most similar to the user's query and 2) the set of the products selected in the past cases most similar to the current case. Then, the system applies a weighted majority voting rule to the retrieval set, because different items in the retrieval set may refer to the same product, to build the recommendation list. Both these approaches show how cases can be generated and reused in the recommendation task.

In addition, we observe that many RSs, e.g., collaborative filtering [2], follow the single-shot recommendation strategy, where given a user's request for recommendations the system computes and shows to the user the recommendation list, and the session ends. Conversely, in conversational RSs a recommendation session does not terminate immediately after the first recommendation list is shown to the user, but it evolves in a dialogue where the system tries to elicit step-by-step the user's preferences to produce better recommendations [3]. In conversational RSs a recommendation case should record not only the user's query and her selected products but also the important information derived from the human-computer dialogue. So, for instance, in critique-based conversational RSs [4, 16, 8, 6, 10] the information contained in the user's critiques should also be stored in the recommendation case.

Moreover, most of the existing case-based RSs have been designed for Web users, not for mobile users who move from place to place and access the system using mobile devices such as PDAs or mobile phones. In mobile RSs the contextual information, such as the user's position and the time of her request, is important, and should be included in the case model and exploited in the recommendation process.

In a previous paper [15], we presented our critique-based recommendation approach and its implementation in MobyRek, a mobile case-based RS that helps mobile users find their desired travel products (restaurants). In that paper, we also presented a live-user evaluation of MobyRek, and the experimental results showed that our recommendation methodology is effective in supporting mobile users in making product selection decisions. We also showed in [12] that the composite query representation employed in our recommendation methodology results in a better recommendation quality over a simpler query representation using either a logical or

similarity query, and in [13] that the exploitation of both long-term and session-specific user preferences does improve recommendation performance compared to the exploitation of a single preferences component.

In this paper, we present the case model employed in our mobile recommendation methodology, which extends the case modeling approaches presented in [14] and [17]. In our approach, a mobile recommendation session is modeled as a case that is built when a user requests a product suggestion, and it is incrementally updated throughout the conversational recommendation session. The CBR problem solving strategy is used to exploit (reuse) the knowledge contained in the past recommendation cases to build the user-query representation, and to adapt the current case to the user's critiques. In this paper, we also present off-line experiments aimed at evaluating the impact, on the system's recommendation performance, of the different case components. In particular, we compare the full case model with the other partial case models. These off-line experiments exploit the log data of real recommendation cases recorded in the previous live-user evaluation of MobyRek [15]. The experimental results show that the full case model does improve the system's recommendation performance over the partial ones, and that the case components that model the user's contextual information, default preferences, and initial preferences, are the most important for mobile context-aware recommendation. In summary, the paper makes the following contributions.

- A new and more comprehensive case model for the mobile conversational recommendation problem.
- A case-based approach for building and revising the user-query representation.
- A number of off-line experiments that show the impact of the different case components on the system's recommendation performance.

The rest of the paper is organized as follows. In Section 2, we present the product and the user-query representations, and discuss the recommendation process. Section 3 presents the proposed case model and discusses our case-based approach to building and revising the user query representation. In Section 4, we present the experimental evaluation of the proposed case model. Finally, Section 5 gives the conclusions and discusses future work.

2 The Recommendation Approach

In the proposed approach, a product is represented as a feature vector $x = (x_1, x_2, \ldots, x_n)$, where a feature value x_i may be numeric, nominal, or a set of nominal values. For instance, the restaurant x = ("Dolomiti", 1713, {pizza}, 10, {air-conditioned, smoking-room}, {Saturday, Sunday}, {credit-card}) has name x_1 = "Dolomiti", distance x_2 = 1,713 meters (from the user's position), type x_3 = {pizza}, average cost x_4 = 10 Euros, characteristics x_5 = {air-conditioned, smoking-room}, opening days x_6 = {Saturday, Sunday}, and payment method x_7 = {credit-card}.

In a recommendation session, the user-query representation encodes the user's preferences and is used by the system to compute the recommendation list. The user-query representation is managed in the case. (The case-based construction and

adaptation of the query representation are discussed later in Section 3.) The user query q consists of three components, $q = (q_l, p, w)$.

- The logical query (q_l) models the conditions that the recommended products must satisfy. The logical query is a conjunction of constraints, $q_l = (c_1 \wedge c_2 \wedge \ldots \wedge c_m)$.
- The favorite pattern (p), represented in the same vector space of the product representation, $p = (p_1, p_2, \ldots, p_n)$, models the conditions that the recommended products should match as closely as possible. These wish conditions allow the system to make trade-offs.
- The feature importance weights vector (w) models how much important each feature is for the user with respect to the others, $w = (w_1, w_2, \ldots, w_n)$, where $w_i \in [0,1]$ is the importance weight of feature f_i. The system exploits the feature importance weights when it makes trade-offs or when it needs to find query relaxation solutions [15].

For example, the query $q = (q_l = (x_2 \leq 2000) \wedge (x_6 \supseteq \{\text{Saturday, Sunday}\})$; $p = (?, ?, \{\text{pizza}\}, ?, ?, ?, ?)$; $w = (0, 0.4, 0.2, 0, 0, 0.4, 0)$) models a user who looks for restaurants within 2 km from her position that are open on Saturday and Sunday and prefers pizza restaurants. For the user the distance and the opening days are the most important, followed by the restaurant type, and she is indifferent to the other features.

In our approach, a case models a recommendation session starting when a user asks for a product recommendation (see Fig. 1a) and ends either when the user selects a product or when she quits the session with no product selection. A recommendation session evolves in cycles. In each recommendation cycle, the system shows the user a ranked list of recommended products (see Fig. 1b) that she can browse and criticize (see Fig. 1c), and the cycle ends when a new recommendation list is requested and shown. We note that users are supported to make critiques on all the product features, not just on distance and cost.

a) b) c)

Fig. 1. MobyRek user interface.
a) Search initialization options. b) The recommendation list. c) Critique on a numeric feature.

In a recommendation cycle, the current query representation, $q = (q_l, p, w)$, is used by the system to compute the recommendation list, i.e., the system first retrieves the

products that satisfy q_l and then ranks the retrieved products according to their similarity to (p, w). Only the k best products in this ranked list, i.e., those which satisfy q_l and are most similar to (p, w), are shown to the user as the recommendation result for the current cycle. In the retrieval phase, if no products in the catalog satisfy q_l, then the system automatically finds a minimum number of constraints that if discarded from q_l make it satisfiable. In this automatic relaxation process, the constraints involving less important features are considered before those involving more important ones. The discarded constraints are converted to wish conditions and put in p. Similarly, in case no products satisfy a critique stated as must, the critique is converted to a wish condition and put in p. (More details on the recommendation list computation are presented in [15]).

Given a user's request asking for some product suggestions (see Fig. 1a), the system integrates the user's initial input and her long-term preferences to build a case that models the current recommendation session. Next, the system retrieves past recommendation cases similar to the current one, and uses the knowledge contained in these retrieved cases to update the current case. Then, the current case is iteratively adapted through a system-user dialogue that interleaves the system's recommendations with the user's critiques to the proposed products. When the session finishes, the current recommendation case is retained in the system's case base and the knowledge about the newly solved problem is stored. The case retention allows the system to exploit past recommendation cases in making recommendations for users in the future [1]. The model of the recommendation process is shown in Fig. 2.

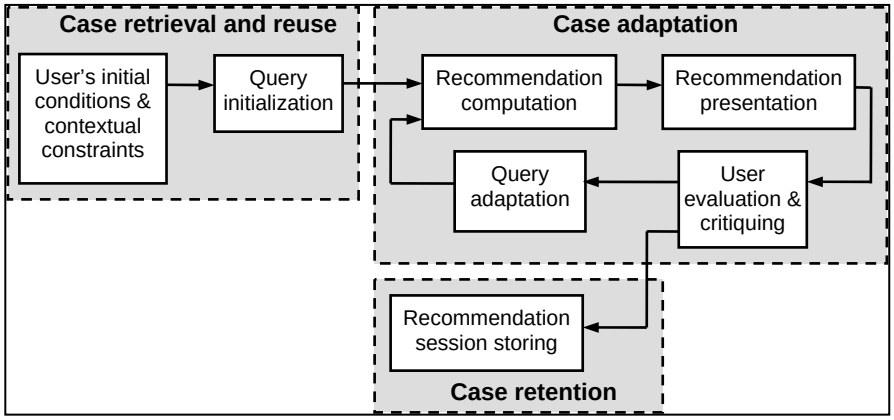

Fig. 2. A mobile recommendation case.

3 Case-based Construction of the User Query Representation

In this section, we present the case model and discuss how CBR is exploited in building and revising the user-query representation.

3.1 The Case Model

In the definition of the case model we considered the following requirements.

- Capture the problem definition, i.e., the knowledge necessary to compute personalized recommendations, and the solution (recommendations) of a mobile recommendation session.
- Support the critique-based conversational recommendation approach.
- Include the contextual information useful to solve a recommendation problem.
- Model both the user's ephemeral preferences (i.e., the user's query) and the user's stable ones.

Given a user's request for a product suggestion, the system exploits the user's initial input together with her stable preferences to initialize the case. In our approach, a recommendation case is modeled as:

C = (*SAH*, *CTX*, *UDP*, *UIS*, *IQR*, *CTZ*, *SEL*);

where:

- *SAH* stores the user's product selections (e.g., a selected hotel) made earlier (using a different web system, NutKing) and related to the current recommendation session. The idea is that if, for instance, users u1 and u2 selected the same hotel h before going to destination and user u2 selected also restaurant r; then the system would consider restaurant r to be a good product for user u1.
- *CTX* stores the contextual information of the current recommendation case. We note that in our tests (reported in Section 4) *CTX* stores only the user's position and the time of her recommendation request. The user's position is automatically detected by the system, or is approximated by the co-ordinates of a close landmark specified by her.
- *UDP* stores the preferences on product features that are explicitly set by the user as her default preference settings, i.e., $UDP = (u_1, u_2, \ldots, u_n)$, where u_i is a default favorite value set by the user. For example, the user may set a default preference of non-smoking room.
- *UIS* stores the conditions that are explicitly specified by the user at the beginning of the session, i.e., $UIS = (v_1, v_2, \ldots, v_n)$, where $v_i = (c_i, s_i)$, and c_i is an initial condition, and $s_i \in$ {wish, must}.
- *IQR* stores the system's initial user-query representation.
- *CTZ* stores the sequence of the critiques that the user makes in the session.
- *SEL* stores the user's product selection at the end of the (mobile) session.

Given a user's request for a product suggestion, together with all the other information contained in the case, the system builds the initial representation of the user query (discussed in the next section): $IQR = q^0 = (q_l^0, p^0, w^0)$, where q_l^0 is the system's initial representation of the user's must conditions, p^0 is the system's initial representation of the user's wish conditions, and w^0 is the system's initial representation of the user's feature importance weights.

After the system outputs a recommendation list the user can browse it and criticize any recommended product (see Fig. 1c). A sequence of critiques is modeled as:

CTZ = null | (*Ctz_Rec*)*;

Ctz_Rec = (*prdPos*, *recSize*, *prdID*, *fID*, *ctzOpt*, *ctzVal*);

where *Ctz_Rec* is a critique record, *prdPos* is the criticized product's position in the recommendation list in the current cycle, *recSize* is the number of the products in the recommendation list in the current cycle, *prdID* is the identity of the criticized product, *fID* is the identity of the criticized feature, *ctzOpt* is the critique operator applied, and *ctzVal* is the value of the criticized feature. An example of a critiques sequence is as follows.

$(1, 20, Prd_4, f_2, Opt_{(<)}, 2000) \rightarrow$
$(2, 12, Prd_7, f_5, Opt_{(\sim\supseteq)}, \{parking, smoking\text{-}room\}) \rightarrow$
$(1, 12, Prd_3, f_4, Opt_{(\sim=)}, 15)$

In this example, the user first makes a critique of "must be nearer than 2 km", then she "wishes to have the parking and smoking-room characteristics", and finally she "wishes a price around 15 Euros".

A recommendation session ends either when the user selects a product or when she quits the session with no product selected. The former case is referred as a successful recommendation case whereas the latter a failed one. Hence, the final result (i.e., the solution part) of a mobile recommendation case is modeled as:

SEL = null | (*prdPos*, *recSize*, *prdID*, *postRating*);

where *prdPos* is the position of the selected product in the recommendation list in the final cycle, *recSize* is the number of the products in the recommendation list in the final cycle, *prdID* is the identity of the selected product, and *postRating* is the user's rating of the selected product (i.e., before visiting it) to indicate how close it is to her needs and preferences.

3.2 Exploiting Past Cases in Building the User Query Representation

The user-query representation initialization starts when the user requests the system for product recommendation and ends before the first recommendation list is produced and shown to her. In the initialization process, both the past information (e.g., past recommendation cases) and the current information (e.g., the user's explicit initial conditions) are exploited. At the end of the initialization process, the system builds the *IQR* case component, $IQR = q^0 = (q_l^0, p^0, w^0)$.

The discussions on how the system builds the initial logical query (q_l^0) and the initial feature weights vector (w^0) components are presented in [15, 11]. Basically, q_l^0 is built exploiting the user's space-time constraints and her initial conditions stated as must, and w^0 is built exploiting the history of the user's critiques expressed in her past recommendation sessions (i.e., recorded in the *CTZ* case component).

In this section, we focus the discussion on how the system initializes the favorite pattern component (p) exploiting the knowledge contained in past recommendation cases, together with the user's default preferences (stored in her mobile device's memory) and her initial conditions stated as wish. The initial favorite pattern component (p^0) is built in the three following steps.

- First, find the past recommendation case which 1) contains a product selection (i.e., the value of the *SEL* case component is not null) and 2) is the most similar to the current case.
- Then, take out the product selected in that most similar case (i.e., stored in the *SEL* case component).

- Finally, merge the three preferences sources (i.e., the product selected in the most similar case, the user's default preferences, and the user's initial conditions stated as wish) to build the initial favorite pattern component (p^0). (This merging step is discussed later.)

As discussed in the previous section, the case model consists of seven components. However, at the time when the user-query initialization takes place only the first four components of the current case are known.

$C^* = (SAH^*, CTX^*, UDP^*, UIS^*, ?, ?, ?)$

The most similar case is found by computing the dissimilarities of the past cases to the current one C^*. In particular, the dissimilarity of a past case C to the current one C^* is given by the following distance function.

$$d(C, C^*) = [1 / (w_{SAH} + w_{CTX} + w_{UDP} + w_{UIS})] . [w_{SAH} . d^{SAH}(C, C^*) + w_{CTX} . d^{CTX}(C, C^*) + w_{UDP} . d^{UDP}(C, C^*) + w_{UIS} . d^{UIS}(C, C^*)]; \quad (1)$$

where the weights w_{SAH}, w_{CTX}, w_{UDP}, and w_{UIS} model the relative importance of the case components *SAH*, *CTX*, *UDP*, and *UIS*, respectively. We note that the weights of these case components are fixed (predefined) in the tests discussed in Section 4.

The dissimilarity of a past case C to the current one C^* with respect to case component *COM* (i.e., either *SAH* or *CTX* or *UDP* or *UIS*) is given by:

$$d^{COM}(C, C^*) = \frac{\sum_{i=1}^{k} w_i^{COM} . d\left(COM_i(C), COM_i(C^*)\right)}{\sum_{i=1}^{k} w_i^{COM}} \quad (2)$$

where k is the number of the features of case component *COM*, $d(COM_i(C),COM_i(C^*))$ is the local dissimilarity function for the i-th feature of case component *COM*, and w_i^{COM} is the weight of the i-th feature of case component *COM*.

For the three case components *SAH*, *CTX*, and *UDP*, the local dissimilarity functions, $d(x_i, y_i)$, are defined for different feature types as follows.

- $d(x_i, y_i) = 1$, if x_i or y_i is undefined.
- For a numeric feature f_i, $d(x_i, y_i) = |x_i - y_i| / (max_i - min_i)$, where max_i and min_i are the maximum and minimum values of feature f_i.
- For a nominal feature f_i, $d(x_i, y_i)$ is equal to zero if $(x_i = y_i)$, and is equal to one if otherwise.
- For a nominal-set feature f_i, $d(x_i, y_i) = 1 - (|x_i \cap y_i| / |x_i \cup y_i|)$.

As discussed in Section 3.1, the *UIS* case component models the preferences that the user explicitly specifies at start-up. When specifying initial preferences, the user also indicates the strength of each initial preference; i.e., if an initial preference is a must or a wish. Therefore, the case similarity computation relative to the *UIS* case component must involve these strengths. In particular, the local dissimilarity functions, $d_{UIS}(x_i, y_i)$, used for the *UIS* case component are defined as follows.

- $d_{UIS}(x_i, y_i) = 1$, if x_i or y_i is undefined.

- For a numeric feature f_i, $d_{UIS}(x_i, y_i)$ is equal to $d(x_i, y_i)$ if both x_i and y_i are must conditions or both are wish ones, and is equal to $[d(x_i, y_i)]^{\alpha}$ ($\alpha \in (0,1)$) if otherwise.
- For a nominal feature f_i, $d_{UIS}(x_i, y_i)$ is equal to zero if ($x_i = y_i$) and both x_i and y_i are must conditions or both are wish ones, and is equal to β ($\beta \in (0,1)$) if ($x_i = y_i$) and x_i and y_i have different strength, and is equal to one if ($x_i \neq y_i$).
- For a nominal-set feature f_i, $d_{UIS}(x_i, y_i) = (1/n_i)\sum_{j=1}^{n_i} d_{UIS}(x_{ij}, y_{ij})$, where n_i is the size (i.e., the number of the elements) of feature f_i.

The similarity of a past case C to the current one C^* is then computed as:

$$sim(C, C^*) = 1 - d(C, C^*) \quad (3)$$

Having found the case C^{sim} most similar to the current one C^*, the system merges the following preferences sources to build the initial favorite pattern (p^0).

- The user's initial conditions stated as wish stored in the *UIS* component of the current case C^*. (We note that the *UIS* component stores both the initial conditions stated as must and those stated as wish.)
- The user's default preferences (i.e., those kept in her mobile device's memory) stored in the *UDP* component of the current case C^*.
- The product selected in the most similar case C^{sim}, i.e., the solution part (*SEL*) of the most similar case C^{sim}.

The preference merging is done at the feature level, i.e., for each feature the preference expressed in the highest priority knowledge source overwrites that in the other sources. The order of the priority among the three knowledge sources is: the user's initial conditions, followed by the user's default preferences, and finally the product selected in the most similar case. For example, if the user specifies a wish condition on cost, then it is used to set the initial preference on cost (p_4^0); otherwise, the system sets p_4^0 by the preference on cost indicated in the user's default preferences. If no preference on cost indicated in the user's default preferences, the system sets p_4^0 by the cost of the product selected in the most similar case.

3.3 Adaptation of the User Query Representation through the User Critiquing

At the end of the initialization process, the system produces the initial user-query representation $q^0 = (q_l^0, p^0, w^0)$, i.e., the content of the *IQR* case component. However, this initial user-query representation may be far from the user's true preferences. To refine this initial representation, the user is involved in a dialogue where the system suggests some products and the user makes some critiques. The elicited preferences help the system to revise its current guess; i.e., adapt its current user-query representation. So, cycle by cycle, a more precise knowledge of the user's needs and preferences is obtained and the critiques are used to adapt the case.

When making a critique to a recommended product, the user indicates the strength of the preference (e.g., in Fig. 1c the user's critique is stated as a wish). A critique expressed as a must is incorporated in the logical query component (q_l), which makes the system focuses on a certain region of the product space, and the weight w_i of the criticized feature f_i is updated. A critique expressed as a wish is incorporated in the

favorite pattern component (p), which makes the system re-rank the recommendation list, and the weight w_i of the criticized feature f_i is updated. After the user makes a critique the system updates the *CTZ* case component incorporating (appending) the new critique.

We note that our critique-based approach is different from the existing ones in several aspects. First, in our approach users are supported to make critiques on feature-level, but not on item-level (like the preference-based approach in [8]). Second, in our approach, given a user's critique, only the user's preference expressed on the criticized feature, but not the values of the other features in the criticized item (as in [4, 8, 6, 10]), is used to adapt the query representation. The rationale is that, due to the peculiar characteristics of the mobile usage environment (e.g., small screens), it is not practical to assume, or require, that the user looks at all the recommended items, or all the features of an item, before making a critique. Third, in our approach, when making a critique, the user is supported to indicate the preference strength (i.e., wish or must) of the critique.

4 Experimental Evaluation

The proposed recommendation approach has been implemented in MobyRek – a mobile case-based RS that supports mobile users in the selection of their desired travel products (restaurants). The MobyRek system was validated with real users, employing a catalog of 84 restaurants [15]. The log data of this live-user evaluation consists of fifteen successful recommendation cases.

In a previous work [13], we introduced the two system variants, *sysMR* and *sysSS*, using the same composite query representation, but in building the initial user-query representation *sysMR* exploited the past recommendation cases while *sysSS* did not. We ran off-line experiments that, exploiting the log data of the real recommendation cases, compared the average position of the user selected product in the first recommendation list in the two variants. The experimental results showed that *sysMR* produced a better quality of the first recommendation list over *sysSS*, i.e., the average position of the selected product in *sysMR* was 23.19% higher than that in *sysSS* [13]. This initial comparative result proves the benefit of exploiting past recommendation cases for generating the first recommendation list. Here, we shall extend this result evaluating the impact of the different case components on the system's recommendation performance. In this evaluation, we compare five system variants exploiting different case components.

- **"Full case model":** the full case model is exploited.
- **"without *COM*":** not exploiting case component *COM*, where $COM \in \{SAH, CTX, UDP, UIS\}$.

Here we are also exploiting the log data of the recommendation cases recorded in the previous live-user evaluation of MobyRek. First we compare the performance of the five system variants with respect to the quality of the first recommendation list. Then, we compare the performance of these variants with respect to the quality of the whole (simulated) recommendation session.

4.1 Quality of the first recommendation list

In this first test, each variant incrementally **replayed the first recommendation cycle** of each tester's case. Here, "replayed" means that the initial user-query representation in the original case was recomputed, considering or not certain case components.

The test procedure, followed by all the five system variants, consists of three main steps (see [13] for another application of this evaluation methodology). First, for each tester's original case the system builds the initial user-query representation $q^0 = (q_l^0, p^0, w^0)$, as discussed in Section 3.2. Second, the system uses the initial query representation to compute the first recommendation list. We note that in this first test the system produces the full recommendation list, not top-*k* as discussed in Section 2. Third, the system checks in the first recommendation list the position of the product selected by the tester in her original case. For each system variant the average position of the selected product is computed over all the simulated cases. We assume that the best variant is the one achieving the highest average position of the selected product.

The performance of the five variants is shown in Fig. 3 with the "all cases" label. The results in Fig. 3 show that the variant exploiting the full case model outperformed the other variants exploiting a partial case model. In particular, the average position of the selected product in the first recommendation list was 15, 15.87, 19.73, 17.73, and 27 for "*Full case model*", "*without SAH*", "*without CTX*", "*without UDP*", and "*without UIS*", respectively (the lower the better, as 1 means the first position in the list). In Fig. 3, by comparing the "*Full case model*" variant with the others we can understand how much a case component influences the quality of the first recommendation list. As shown in Fig. 3, the *SAH* case component has a small impact, since the exclusion of this case component caused just a 5.8% increase of the average position of the selected product. However, each of the *CTX*, *UDP*, and *UIS* case components has a major impact, since its exclusion caused a larger increase of the average position of the selected product, i.e., 31.53%, 18.2%, and 80% for *CTX*, *UDP*, and *UIS*, respectively.

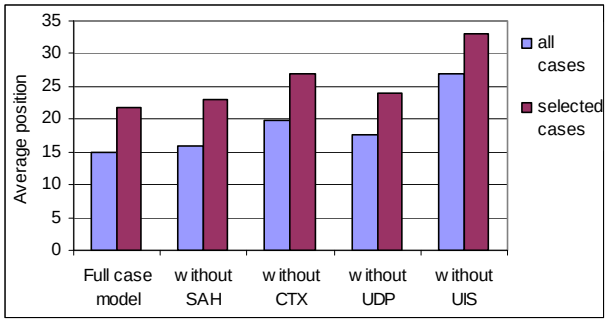

Fig. 3. The average position of the selected product in the first recommendation list.

Since in the live-user evaluation of MobyRek we used the full case model, one might conjecture that a product has been selected just because it appeared in the first screen of the first recommendation list, and therefore the comparison could be biased in favor of the "*Full case model*" variant. Hence, in order to obtain a possibly more

fair comparison, we also considered just the selected products that did not appear in the first screen (i.e., top 3, in the mobile phone interface) of the first recommendation list [13]. The performance, on this subset of simulated cases, of the five system variants is shown in Fig. 3 with the "selected cases" label. Even for these selected cases, the "*Full case model*" variant, which exploited the full case model, still achieved the best quality of the first recommendation list. In particular, the average position of the selected product in the first recommendation list was 21.7, 23, 27, 24.1, and 33 for "*Full case model*", "*without SAH*", "*without CTX*", "*without UDP*", and "*without UIS*", respectively. Hence, the conclusions made above are still true under this probably less biased evaluation.

4.2 Quality of the full recommendation session

In the second test, each system variant incrementally **replayed the whole sessions**. Here, "replayed" means that in a simulated recommendation case the system first re-computed the initial query representation, considering or not certain case components, and then re-applied, one by one in the original order, the user's critiques. To replay a real recommendation case, we had to define different ways of re-applying (i.e., simulating) user critiques. That is because the case models employed by the four variants "*without SAH*", "*without CTX*", "*without UDP*", and "*without UIS*" were different from that employed by MobyRek. Therefore, at each simulated cycle the recommendation list produced by a variant (hereafter called "the output list") can be different from that produced in the real session, and moreover the real criticized product can be absent from the output list. Hence, for each variant, we tried different critique-simulation methods, and measured its performance using the best (or average) result obtained over all these methods. In this way, we are trying to be unbiased, i.e., not in favor of any variant.

In a recommendation session, the cause and motivation for a critique made in a cycle may be inferred based on the product criticized at that cycle or the one selected at the end of the session. Hence, for each system variant, we tried the four critique-simulation methods listed below. (See [12] for more details and another application of these simulation methods)

- *Just repeat the critique*. This method assumes that a user's critique is influenced by her preferences, rather by the products shown. In this method, the critique is re-applied even if the criticized product does not appear in the output list.
- *Critique according to the selected product*. This method assumes that a user's critique is motivated by her selected product. Hence, for a simulated critique the value of the criticized feature in the selected product is used to adapt the user-query representation.
- *Critique according to the product similar to the criticized one*. In case the criticized product is not found in the output list, this method assumes that the user makes a similar critique to a product (in the output list) similar to the criticized one. In particular, if the criticized product is found in the top *N* of the output list, then the critique is repeated; otherwise, the product most similar to the criticized one is identified, and the value of the criticized feature in that product is used to adapt the user-query representation.

- *Critique according to the product similar to the selected one*. In case the criticized product is not found in the output list, this method assumes that the user makes a similar critique to a product (in the output list) similar to the selected one.

The simulation test procedure, followed by all the five system variants, consists of four main steps [12]. At Step 1, for each tester's original case the system builds the initial user-query representation (as discussed in Section 3.2), considering or not certain case components, and retrieves the list of the original critiques that are recorded in the *CTZ* case component. At Step 2, the system uses the current query representation to compute the output list (as discussed in Section 2). At Step 3, the system checks if one of the termination conditions is met; if not, the system proceeds to Step 4. The simulation of a tester's original case ends either when the selected product appears in the view window of the output list (a successfully simulated case) or when all the original critiques have been re-applied but the selected product is still not found (an unsuccessfully simulated case). In this test procedure, the view window models the number of products that the simulated user is supposed to look at. At Step 4, the system takes the next critique from the original critiques list, and simulates it using one of the four critique-simulation methods. The simulated critique is then used by the system to adapt the query representation (see Section 3.3), and the simulation process proceeds to the next cycle (at Step 2). Finally, for each system variant the number of successfully simulated cases and the average session length are measured.

Fig. 4 shows the results obtained for a view-window size of 5 items. For each system variant, Fig. 4 shows the best result among the four critique-simulation methods and the average result over these methods.

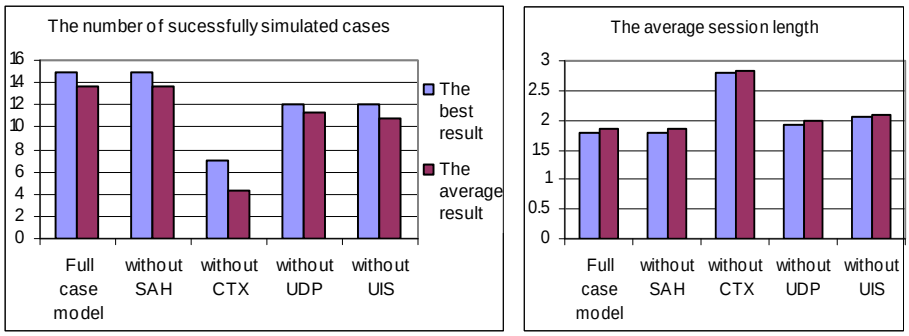

Fig. 4. The number of successfully simulated cases and the average session length, for the view-window size of 5.

We first look at the number of successfully simulated cases (success rate). When comparing on the best result, both the "*Full case model*" and "*without SAH*" variants achieved the same highest success rate. This confirms the result of the previous test where the average position of the selected product in the first recommendation list in the "*without SAH*" variant was just a bit lower (5.8%) than that in "*Full case model*". Also, the "*without SAH*" variant, applying the user critiques, achieved the same success rate as "*Full case model*". Conversely, the success rates of the three variants "*without CTX*", "*without UDP*", and "*without UIS*" were 53.33%, 20%, and 20%,

respectively, lower than that of "*Full case model*". Hence, these three variants achieved a lower quality of the first recommendation list (see Fig. 3) and a lower success rate. Looking at the average result we observe that "*Full case model*" and "*without SAH*" achieved the same success rate, but the success rates of "*without CTX*", "*without UDP*", and "*without UIS*" were lower than that of "*Full case model*". These comparative results show that the three case components *CTX*, *UDP*, and *UIS* are important for the mobile recommendation problem, since their exclusion from the case model causes a decrease of the system's recommendation performance.

We now look at the average session length (i.e., the average number of recommendation cycles). We note that in the simulated recommendation cases, the length of the dialogue was rather short, i.e., just 2-3 recommendation cycles. This average session length is much shorter than those of Web critique-based RSs [16, 8, 6, 10], and it is due to the mobile usage context, i.e., mobile users tend to spend less time and effort than Web users do in searching for some information or products. As shown in Fig. 4, the three variants "*without SAH*", "*without UDP*", and "*without UIS*" took the average session lengths approximately to that taken by "*Full case model*". However, the "*without CTX*" variant consumed a longer (55.56%) average session length than "*Full case model*". In fact, the exclusion of the *CTX* component from the case model caused not only a poor success rate but also a longer session length.

5 Conclusions and Future Work

In this paper, we have described a case-based approach for modeling mobile context-aware recommendation problems and solutions. The proposed case model is capable of modeling evolving recommendation sessions, capturing the recommendation context, supporting critique-based user-system conversations, and integrating both ephemeral and stable user preferences. We have discussed the exploitation of the proposed case model to build and revise the user query representation. We have illustrated an experimental evaluation aimed at testing the impact of different case components on the system's recommendation performance. The experimental results showed that the exploitation of the full case model results in a better recommendation, hence proving the correctness of our design choices, and that the case components that model the user's contextual information, default preferences, and initial preferences, play the most important role in the mobile recommendation problem.

In the proposed approach, when searching the case base for similar cases, the system uses a fixed set of pre-defined importance weights for the case components (see Equation 1). However, a case component may be very important in a recommendation situation, but less important in another one. Hence, in the future we want to define an appropriate approach for learning and adapting the importance weights of the case components for a given user in a particular recommendation situation. Also, in our approach the *CTX* case component contains only the user's position and the time of the request. In fact, the recommendation contextual information may include not only spatial-temporal information but also other situational information such as if the user goes to the restaurant alone or with a friend,

on a date or for a casual dinner, etc. We plan to deal with this extension of the contexts exploitation in a future project.

References

1. Aamodt, A., Plaza, E.: Case-based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. AI Communications 7(1), 39–59 (1994)
2. Adomavicius, G., Tuzhilin, A.: Toward the next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. IEEE Trans. Knowledge and Data Engineering 17(6), 734–749 (2005)
3. Bridge, D., Göker, M., McGinty, L., Smyth, B.: Case-based Recommender Systems. Knowledge Engineering Review 20(3), 315–320 (2005)
4. Burke, R.: Interactive Critiquing for Catalog Navigation in E-Commerce. Artificial Intelligence Review 18(3-4), 245–267 (2002)
5. Burke, R.: Hybrid Web Recommender Systems. In: Brusilovsky, P., Kobsa, A., Nejdl, W. (eds.) The Adaptive Web: Methods and Strategies of Web Personalization, pp. 377–408. Springer (2007)
6. Chen, L., Pu, P.: Preference-based Organization Interface: Aiding User Critiques in Recommender Systems. In: 11th International Conference on User Modeling, pp. 77–86. Springer (2007)
7. Lorenzi, F., Ricci, F.: Case-based Recommender Systems: A Unifying View. In: Mobasher, B., Anand, S. (eds.) Intelligent Techniques for Web Personalization, pp. 89–113. Springer (2005)
8. McGinty, L., Smyth, B.: Adaptive Selection: An Analysis of Critiquing and Preference-based Feedback in Conversational Recommender Systems. International Journal of Electronic Commerce 11(2), 35–57 (2006)
9. McSherry, D.: Completeness Criteria for Retrieval in Recommender Systems. In: 8th European Conference on Case-Based Reasoning, pp. 9–29. Springer (2006)
10. McSherry, D., Aha, D.W.: Mixed-Initiative Relaxation of Constraints in Critiquing Dialogues. In: 7th International Conference on Case-Based Reasoning, pp. 107–121. Springer (2007)
11. Nguyen, Q.N., Ricci, F.: User Preferences Initialization and Integration in Critique-Based Mobile Recommender Systems. In: 5th International Workshop on Artificial Intelligence in Mobile Systems, pp. 71–78. Universitat des Saarlandes Press (2004)
12. Nguyen, Q.N., Ricci, F.: Replaying Live-User Interactions in the Off-Line Evaluation of Critique-based Mobile Recommendations. In: Recommender Systems 2007, pp. 81-88. ACM Press (2007)
13. Nguyen, Q.N., Ricci, F.: Long-Term and Session-Specific User Preferences in a Mobile Recommender System. In: 2008 International Conference on Intelligent User Interfaces, pp. 381–384. ACM Press (2008)
14. Ricci, F., Venturini, A., Cavada, D., Mirzadeh, N., Blaas, D., Nones, M.: Product Recommendation with Interactive Query Management and Twofold Similarity. In: 5th International Conference on Case-Based Reasoning, pp. 479–493. Springer (2003)
15. Ricci, F., Nguyen, Q.N.: Acquiring and Revising Preferences in a Critique-based Mobile Recommender System. IEEE Intelligent Systems 22(3), 22–29 (2007)
16. Shimazu, H.: Expertclerk: A Conversational Case-based Reasoning Tool for Developing Salesclerk Agents in E-Commerce Webshops. Artificial Intelligence Review 18 (3-4), 223–244 (2002)
17. Stahl, A.: Combining Case-Based and Similarity-Based Product Recommendation. In: 8th European Conference on Case-Based Reasoning, pp. 355–369. Springer (2006)