Replaying Live-User Interactions in the Off-Line Evaluation of Critique-based Mobile Recommendations

Quang Nhat Nguyen
Free University of Bozen-Bolzano
Piazza Domenicani 3, 39100 Bolzano (BZ), Italy

Quang.NhatNguyen@unibz.it

Francesco Ricci
Free University of Bozen-Bolzano
Piazza Domenicani 3, 39100 Bolzano (BZ), Italy

fricci@unibz.it

ABSTRACT

Supporting conversational approaches in mobile recommender systems is challenging because of the inherent limitations of mobile devices and the dependence of produced recommendations on the context. In a previous work, we proposed a critique-based mobile recommendation approach and presented the results of a live users evaluation. Live-user evaluations are expensive and there we could not compare different system variants to check all our research hypotheses. In this paper, we present an innovative simulation methodology and its use in the comparison of different user-query representation approaches. Our simulation test procedure replays off-line, against different system variants, interactions recorded in the live-user evaluation. The results of the simulation tests show that the composite query representation, which employs both logical and similarity queries, does improve the recommendation performance over a representation using either a logical or a similarity query.

Categories and Subject Descriptors

H.5.2 [**Information Interfaces and Presentation**]: User Interfaces – *Graphical user interfaces (GUI), Evaluation/methodology*

General Terms

Design, Experimentation, Human Factors

Keywords

Critiquing, mobile recommender systems, query representation, simulation test

1. INTRODUCTION

When searching for products and services, e-commerce web sites users are often overwhelmed by the number of options to consider. Hence they need some system support to filter out irrelevant products, compare candidates, and select the best one(s). Recommender Systems (RSs) are decision support tools that solve this information overload problem, by suggesting products and services personalized to the user's needs and preferences at her particular context.

A research direction in the RSs field, that has received much attention, is conversational RSs. In these systems the user gets her desired items through a structured human-computer dialogue [13]. Critique-based RSs are conversational RSs which, at each interaction cycle, interleave the system's product proposals with the user's critiques to the proposed products [12, 2, 9, 5, 10, 6, 4, 7, 11, 3]. The user makes a critique to a recommended item when either a feature of the item does not satisfy the user or when she wants to emphasize that it is very important to her. A user's critique, for instance, may specify an unsatisfied preference, such as "I want a restaurant cheaper than this", or confirm an important preference, such as "I prefer to rent a room with private bathroom".

There have recently been an increasing number of mobile RSs introduced in the literature [14, 5, 16, 15, 11]. In practice, designing an effective and usable mobile RS requires the recommendation methodology to overcome obstacles typically present in the mobile usage environment (e.g., smaller screens and limited input modalities) and to be suitable for mobile users' behavior.

In our previous paper [11], we have presented a critique-based mobile RS that, to make user interaction simple and fast, supports a very limited input of explicit user preferences through system questions and is mostly based on critiques. When making critiques, the user also assigns the strength (i.e., wish or must) of the expressed preference, which helps the system correctly exploit the user's critique. To produce relevant recommendations, the system integrates both long-term and session-specific user preferences, employs a composite query representation, and exploits many sources of user related information [8]. The proposed recommendation methodology has been implemented in MobyRek, a mobile phone on-tour RS that assists on-the-go users in searching for travel products (restaurant). MobyRek was evaluated with real users with respect to: usability (i.e., functionality, efficiency, and convenience), recommendation quality, and overall user satisfaction [11]. The objective and subjective results of the on-line evaluation showed that our recommendation methodology is effective in supporting on-the-go users in making product choice decisions.

However, in the on-line evaluation we could not test different system variants that employ different user-query representation approaches. In particular, we would like to understand whether or not our composite user-query representation approach, which employs both logical and similarity query components, results in a better recommendation performance (interaction length and

percentage of successful sessions) against other approaches that employs an individual query representation based on either logical filtering or similarity-based retrieval. If being tested in the on-line evaluation, this check-experiment would have required test users to use and evaluate different system variants implementing the alternative query models. This would have required them to spend much more time and effort, and would have limited the number of variants that we could compare. This is a general problem for empirical system evaluation. Therefore, we decided to design a simulation methodology that could benefit from the interaction logs collected in the live-user test and replay such interactions with systems (slightly) different from that used in the live-user evaluation.

In this paper, we describe the proposed simulation test procedure and how it has been used to validate the hypothesized advantage of a hybrid (logical and similarity-based) query model. The simulation test procedure takes as input a dataset that comprises a historical series of recommendation sessions, where each session contains a historical sequence of user critiques. In other words, recommendation sessions, and user critiques in a session as well, are replayed in the simulation in the original order, instead of being simulated randomly. In summary, the paper makes the following contributions.

- A simulation test procedure that can be used for testing critique-based RSs in simulated environments, based on the replay of real recommendation sessions and the various ways of simulating user critiques.
- A number of simulation tests that show the advantage of the composite query representation (i.e., using both logical and similarity queries) over a single query representation (i.e., using either logical or similarity query) in term of the rate of successful recommendation sessions.

The remainder of the paper is organized as follows. Section 2 discusses the product representation and the user preferences model, followed by the description of the recommendation process. The hypothesis on the user-query representation is posed and discussed in section 3. In section 4, we discuss the proposed simulation procedure, apply it to test the posed hypothesis, and discuss on the observed results. In section 5, we recall the off-line evaluation approaches introduced in previous research in critique-based RSs. Finally, the conclusions are given in Section 6.

2. RECOMMENDATION METHODOLOGY

In this section we shall briefly recall our proposed critique-based mobile recommendation methodology, first describing the product representation and the user preferences model and then presenting how the system produces personalized product recommendations for on-the-go users [11].

2.1 Product Representation and User Preferences Model

A product is represented as a feature vector $x= (x_1, x_2,..., x_n)$, where a feature value x_i can be numeric, nominal, or a set of nominal values. For instance, the representation of the restaurant x= (Trittico, 79, {pizzeria}, 10, {air-conditioned, parking}, {7, 1}, {credit-card}) means that the name (x_1) is Trittico, the distance from the user's position (x_2) is 79 m, the restaurant type

(x_3) is pizzeria, the average cost (x_4) is 10 euros, the characteristics (x_5) are air conditioned and parking, the days open (x_6) are Saturday and Sunday, and the accepted method of payment (x_7) is credit card.

To generate personalized recommendations, a recommender system needs a representation of the user's preferences. Preferences vary from user to user, and even from situation, i.e., context, to situation for the same user. In our approach, the user preferences model includes both contextual (e.g., space-time constraints) and product-feature (e.g., air conditioned) preferences, and incorporates both long-term (e.g., a preference on non-smoking room) and session-specific (e.g., a wish to eat a pizza) user preferences [8, 11]. Though the specification of initial preferences (i.e., at start-up) is supported, and optional, for users, session-specific preferences are acquired mainly through the user's critiques collected during the recommendation session.

In a recommendation session, the user's preferences are encoded in the system's user query representation which is used to compute the recommendation list. In our approach, the user query representation q consists of three components, $q= (Q_L, p, w)$.

- The logical query, $Q_L= (c_1 \wedge c_2 \wedge ... \wedge c_m)$, models the conditions that **must** be satisfied by every recommended product. The logical query is a conjunction of constraints, where each one (c_j) relates to a single feature.

- The favorite pattern, $p= (p_1, p_2, ..., p_n)$, models the conditions that the recommended products **should** match as closely as possible. The wish conditions (p_i, i=1..n) allows the system to make trade-offs.

- The feature importance weights vector, $w= (w_1, w_2, ..., w_n)$ models **how much important** to the user a feature is with respect to the others, where $w_i \in [0,1]$ is the importance weight of feature f_i. The system refers to the feature weights when it needs to make trade-offs or to find relaxation solutions for unsatisfiable (i.e., empty-result) logical queries.

For example, the query $<Q_L=(x_2 \leq 1000) \wedge (x_6 \supseteq \{7,1\})$; $p=(?,?,\{pizzeria\},?,?,?,?)$; $w=(0,0,0.4,0.6,0,0,0)>$ models a user who looks for restaurants within 1 km from her position that are open on Saturday and Sunday and prefers pizzeria restaurants. For the user the cost is most important, followed by the restaurant type, and he is indifferent to the other features.

2.2 The Recommendation Process

A recommendation session begins when a mobile user asks the system for a product suggestion, and it ends when the user selects a product or she quits the session with no selection. A recommendation session evolves in cycles. At a recommendation cycle, the system shows the recommended products (see Figure 1b) that the user can browse to see the product details and make critiques (see Figure 1c,d). After the user has expressed a critique, the critique is exploited by the system to compute a new recommendation list that is showed to the user in the next cycle. In an overview, a recommendation session is logically divided into three phases: initialization, interaction and adaptation, and retaining.

At start-up, the user is offered with three options for the search initialization (see Figure 1a).

- The *"No, use my profile"* option let the system automatically build the initial search query, exploiting the user's long-term preferences.
- The *"Let me specify"* option let the user explicitly specify initial preferences.
- The "Similar to" option let the user specify a known product as a favorite sample for the system's search initialization.

Figure 1: MobyRek user interface.
(a)-Options for initializing the search; (b)-The recommendation list; (c)-Critique on a numeric feature; (d): Critique on a nominal-set feature.

Having the user's selected an initialization option the system integrates the user's initial input and long-term preferences to build the initial search query and to initialize the case that models the current recommendation session [1]. This case representation, which is updated as the recommendation session evolves, consists of the following components.

- *SEL_AT_HOME*. The products selected previously that influence the current decision.
- *USER_CTX*. The user's contextual information (i.e., the current position and the time of the request).
- *USER_DEFT_PREF*. The user's default preferences (e.g., non-smoking room) stored in the mobile device's memory.
- *USER_INIT_SPEC*. The preferences that the user explicitly specified at start-up.
- *SYS_INIT_REP*. The system's initial representation of the user query.

- *USER_CTZ_SEQ*. The sequence of the critiques that the user expresses in the session.
- *SEL_ON_GO*. The product that the user selects at the end of the (on-the-go) session.

The system exploits multiple sources of the user-related knowledge to build the initial query representation (i.e., the *SYS_INIT_REP* case component), including a) the user's contextual information, b) the user's previous selections, c) the past selections of the users, and d) the user's stated initial preferences. Basically, the logical query Q_L is initialized using the user's space-time constraints and initial preferences stated as must. The favorite pattern p is initialized integrating the user's long-term preferences pattern and initial preferences stated as wish. The feature weights w are initialized exploiting the history of the user's interactions with the system. More details on the query representation initialization are presented in [8, 11].

The initial query representation is then used by the system to compute the first recommendation list. In the computation of a recommendation, the system first discards those products that do not satisfy the logical query component Q_L and then ranks the remaining products according to their similarity to (p, w) – the favorite pattern and the feature weights vector components. The more similar a product is to (p, w) the higher it appears in the ranked list. Similarity is computed as the difference between the maximum similarity value (=1) and the generalized weighted Euclidean distance between the product and (p, w). In the recommendation computation, if no products in the catalog satisfy the logical query Q_L the system finds a minimum number of constraints that if discarded from Q_L make it satisfiable. In this relaxation, a constraint involving a less important feature is considered for relaxation before another involving a more important one (see [11] for more details). The relaxed constraints are then converted to wish conditions and incorporated in the favorite pattern p.

When the recommended products list is showed to the user (see Figure 1b), three situations may occur. In the first one, the user is satisfied with a product and selects it; the session ends successfully. In the second situation, the user is not satisfied with the recommended products and quits; the session terminates unsuccessfully. In the third situation, the user is interested in a recommended product, but not completely satisfied with it. Hence, the user criticizes the product to further express his preferences. The user may, for example, make a critique to say that a product is too far from his position (see Figure 1c) or to state some interested features (see Figure 1d). When making a critique to a recommended product, the user also assigns the strength (i.e., must or wish) of the expressed preference. This help the system correctly exploit the user's critique. A critique stated as a must, incorporated in Q_L, makes the system zoom in a certain region of the product space. A critique stated as a wish, incorporated in p, makes the system refine the ranked list. After the user makes a critique, the system interprets and incorporates the critique in the query representation (see [11], for more details).

When a recommendation session ends, whether successfully or not, it is retained by the system as a case (see the case model discussed above). Hence, the system can exploit past
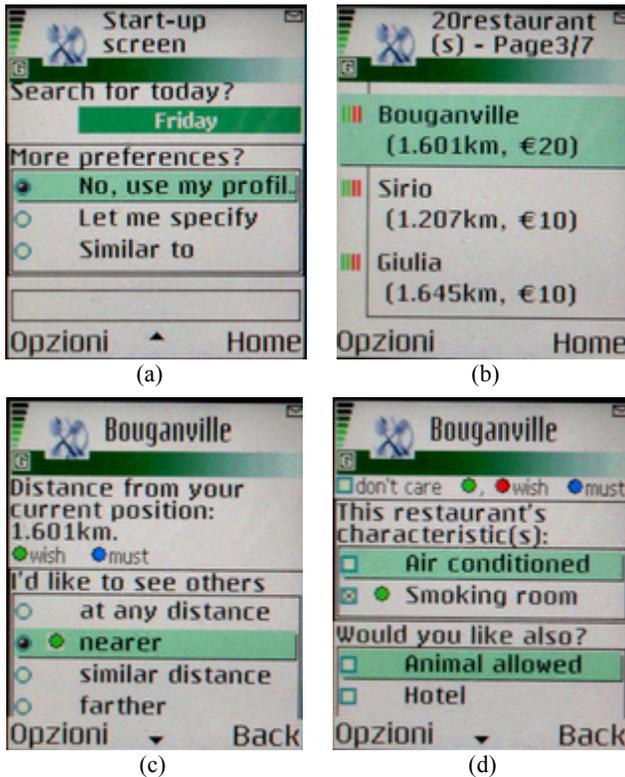
recommendation sessions in making new recommendations for the user, and other similar ones, in future [8, 11].

3. HYPOTHESIS ON THE USER QUERY REPRESENTATION

In traditional approaches, often employed in information retrieval systems, user's preferences are represented by a (usually conjunctive) set of logical constraints [4, 12]. The advantage of this representation approach is that the produced results are easy to understand for users, since only those products satisfying the logical constraints are retrieved and shown to the user. However, the major disadvantage of such representation is that it is very difficult to recover from failing queries (i.e., whose execution returns an empty result set). In such a case, the system needs to repair the failing query.

In other approaches based on similarity retrieval, which is typically used in instance-based learning and case-based reasoning systems, user's preferences are represented by a favorite feature vector [2, 5, 6, 7]. The advantage of this representation approach is that a similarity-based retrieval always returns a non-empty result set in the form of ranked products list. However, a recommender system using this representation approach may show a user products violating some of her preferences, since the retrieved products may have the features similar, but not necessarily identical, to those in the favorite pattern. This may decrease the user's confidence in the system. Moreover, this representation approach requires the system to scan all available products, which can be a heavy task, in terms of computational resource and time, if the size of the product database is large.

We hypothesize that a user query representation integrating both logical and similarity queries can better reflect the user's needs and preferences, and hence better support the recommendation process, benefiting from both individual representations. In particular, the logical query component allows the system to zoom in relevant and (much) smaller regions whereas the similarity one helps the system rank all the products in a relevant region according to their similarity to the user's favorite pattern. However, this hypothesis must be empirically validated since the additional complexity of a composite query model may cause negative impacts on the transparency of the recommendation process and on the efficiency of the implementation of the retrieval algorithm.

4. EXPERIMENTAL EVALUATION

As mentioned in section 1, the MobyRek system was initially evaluated with real users, and the log data of this evaluation, consisting of fifteen successful recommendation sessions, was recorded. In this section we describe how we have exploited the testers' recommendation sessions in off-line simulations to check the hypothesis posed in section 3. In particular, we have introduced three MobyRek variants employing different query representations.

- **repMR**. This variant represents the user query with a logical query and a similarity one, $q = <Q_L, p, w>$.
- **repLQ**. This variant represents the user query with a logical query, $q = <Q_L>$.

- **repFP**. This variant represented the user query with a favorite pattern, $q = <p, w>$.

In this section, we first describe the simulation test procedure together with the supported critique-simulation methods, and then discuss the results of the simulation tests that compare the three system variants regarding the recommendation performance.

4.1 Simulation Test Procedure

In the simulation tests, the three system variants "replayed" the testers' recommendation sessions (i.e., those recorded in the live users' evaluation). In particular, given a tester's original session in the simulation the system used the initial query representation to compute the first recommendation list, and then re-applied, one by one in the original order, the tester's critiques. In order to replay a recommendation session in a simulation, we had to define different ways of re-applying the user critiques. There are two major problems to consider. First, during a simulated session the system's recommendation list (at each cycle) could be different from that observed in the live user's session, and also the criticized product in the original session could be not present in the output list. Second, the user query representations employed by the two system variants *repLQ* and *repFP* are different from that employed by MobyRek in the live users evaluation. Because of these reasons, it is difficult to select a unique method for replaying the user critiques. Therefore, for each system variant we have tried different critique-simulation methods. We have finally compared the system variants, measuring for each variant the result obtained with the best critique-simulation methods (not to bias the evaluation towards any system variant).

To illustrate the different critique-simulation methods, let's assume a tester made the following critiques sequence in his recommendation session:

$$(1, 20, x^4, f_2, must_{(<)}, 2000) \rightarrow$$
$$(2, 12, x^7, f_5, wish_{(\supseteq)}, \{parking, smoking\text{-}room\}) \rightarrow$$
$$(1, 12, x^3, f_4, wish_{(=)}, 15);$$

and suppose that $l = (l_1, ..., l_n)$ is the product selected by the tester. In this example, the first critique was made to product x^4 (ranked first in a list of twenty products) on feature f_2. This is a must critique, and states that the distance from the user's position to the restaurant must be less than 2000 meters. The second is a wish critique to product x^7, which is ranked second in a twelve-product list. The critique is made on feature f_5, and says that *parking* and *smoking room* are preferred. The third is a wish critique to product x^3, which was ranked first in a list of twelve products. The critique is made on feature f_4, and expresses a preference for a price of about 15 Euros.

Five critique-simulation methods were tried in the simulation.

- **[rpt_ctz]** (abbr. for "*repeating critique*"). This method assumes that a user's critique is stable, and it is influenced more by the user's preferences than by the products shown. In particular, a tester's critiques are repeated exactly as in the original session. In other words, this critique-simulation method re-applies the critique even if the criticized product is not shown in the output list.

- **[val_sel]** (abbr. for "*values in the selected product*"). This method assumes that a user's critique is oriented by the user's selected product. In this method, for each critique the criticized feature's value in the selected product,

rather than the value in the criticized product, is used to adapt the user query. For instance, let's consider the first critique in the example above. Here, the criticized product in the original session has value 2000 for feature f_2. Using this simulation method, at the first cycle the value l_2, i.e., the value of the second feature in the product selected by the tester, which could be different from 2000, is used to adapt the user query.

- **[full_val_sel]** (abbr. for "*full values in the selected product*"). This method extends the *[val_sel]* one to measure how many cycles are still needed to locate the selected product if the critiques cannot bring the selected product in the output list. In this method, first the tester's critiques sequence is replayed by applying the *[val_sel]* method. If the selected product is not found in the output list, then the values in the selected product of the remaining features (i.e., those not being criticized during the tester's session) are used, feature by feature, to adapt the user query. Let's consider the example mentioned above. First, the value l_2 is used to adapt the user query. Next, the value l_5 is used. If after the third cycle (i.e., after the value l_4 is used to adapt the user query) the selected product l is still not found, then the other values l_i ($i \notin \{2,5,4\}$) are used, feature by feature at each cycle, to adapt the user query.

- **[sim_ctz]** (abbr. for "*similar to the criticized product*"). In case the criticized product is not found in the output list, this method assumes that the user will likely make a similar critique to a product (in the output list) which is similar to the criticized one. This critique-simulation method is similar to that employed in [3]. In our method, for each of the tester's critiques if the real criticized product appears in the top N recommended products, then the critique is repeated. Otherwise, the product most similar to the one criticized in the original session is identified in the output list, and the value, in this (most similar) product, of the criticized feature in the original session is used to adapt the user query. Let's illustrate this method using again the above mentioned example. At the first cycle, if the product x^4 appears in the top N products of the output list, then the $must_{(<)}$ critique operator is applied to the product x^4 on feature f_2 with criticized value 2000. Otherwise, let's call x^* the product most similar to x^4 among the top N products of the output list; then, the value x_2^* (i.e., the value of feature f_2 in x^*) is used to adapt the user query.

- **[sim_sel]** (abbr. for "*similar to the selected product*"). In case the criticized product is not found in the output list, this method assumes that the user will likely make a similar critique to a product (in the output list) which is similar to the user's selected one. This critique-simulation method is similar to that introduced in [7]. In our method, for each of the tester's critiques if the actual criticized product appears in the top N products of the output list, then the critique is repeated. Otherwise, the product most similar to the one selected in the original session is identified, and the value, in this (most similar) product, of the feature criticized in the original session is used to adapt the user query. Let's use again the above mentioned

example to illustrate this method. At the first cycle, if the product x^4 appears in the top N products of the output list, then the $must_{(<)}$ critique operator is applied to the product x^4 on feature f_2 with criticized value 2000. Otherwise, let's call l^* the product most similar to l (i.e., the tester's selected product) amongst the top N products of the output list; then, the value l_2^* (i.e., the value of feature f_2 in l^*) is used to adapt the user query.

During a user's recommendation session, at each cycle the user's criticized product and the selected one provide the knowledge about the motivation and cause of the critique made at that cycle. Basically, in a replayed session the *[rpt_ctz]* and *[sim_ctz]* methods base the critique simulation on the products actually criticized by the test user or on those similar to them, whereas the remaining methods base the critique simulation on the tester's selected product or those similar to it. Table 1 describes the combinations of system variants and critique-simulation methods that we used.

Table 1: The ten combinations of the three system variants and different critique-simulation methods

		User query representation		
		repMR	repLQ	repFP
Critique-simulation method	[rpt_ctz]	√	√	√
	[val_sel]		√	√
	[full_val_sel]			√
	[sim_ctz]		√	√
	[sim_sel]		√	√

We note that in the simulation tests reported here we did not run each system variant with all five critique-simulation methods. The *repMR* system variant employed the same query representation approach that MobyRek had used in the on-line evaluation, and hence at each cycle in a simulated session the output list was the same as that produced in the original session. Therefore, for the *repMR* system variant repeating the critiques seem to be the most appropriate simulation method. We introduced the *[full_val_sel]* method, only for the *repFP* system variant, in order to support a previously proposed and popular simulation method [7] used in similarity-based retrieval systems, where all the feature values of the target item can be exploited to test the system's retrieval performance.

The three system variants, each employing a query representation approach and different critique-simulation methods, were evaluated by the simulation test procedure shown in Figure 2. In this procedure, the first input parameter is a set of successful sessions (i.e., those ending with an item selection) collected in the live users evaluation. The second input parameter is *view_size*, i.e., the number of products in a recommendation list that the simulated user is supposed to look at (i.e., the top N products). For example, *view_size* equal to 10 means that the simulated user is supposed to analyze only the top 10 products in every recommendation list.

At Step 1, given a tester's recommendation session the simulated initial query representation is built as follows.

- For the *repMR* system variant the simulated initial query is exactly the initial query in the original session, $q^0 = <Q_L^0, p^0, w^0>$ (i.e., taken from the *SYS_INIT_REP* case component of the original session).

- The *repLQ* variant constructs the simulated initial query, $q^0 = <Q_L^0>$, exploiting only the initial preferences that the tester had explicitly specified at the time of his request. The system first retrieves the *USER_INIT_SPEC* case component of the original session and then converts every initial wish condition to a must one, and finally incorporates all the must conditions in Q_L^0.

- The *repFP* system variant first retrieves the *SYS_INIT_REP* case component of the original session and then converts every initial must condition to a wish one, and finally incorporates all the wish conditions in p^0. The simulated initial feature weights vector w^0 is taken from the *SYS_INIT_REP* case component of the original session.

Input
  S: a set of successful sessions
  view_size: a view-window size
Output
  The number of successfully simulated sessions
  The average recommendation length

$num\_of\_succ \leftarrow 0$;
$total\_cycles \leftarrow 0$;

For each session s_k in *S*

  $cycles\_per\_session \leftarrow 0$;
  Step1. Build the initial query representation.
         Build the list of critiques, *CtzList(s_k)*, for session s_k.
  Step2. Produce the recommendation list.
         $cycles\_per\_session \leftarrow (cycles\_per\_session + 1)$;
  Step3. Check if the termination condition for session s_k is satisfied. If true, go to Step5; otherwise, go to Step4.
  Step4. Take the next critique from *CtzList(s_k)*.
         Simulate the critique.
         Adapt the user query representation to the simulated critique.
         Go to Step2.
  Step5. $total\_cycles \leftarrow (total\_cycles + cycles\_per\_session)$;
         If the selected product in session s_k appears in the view window of the recommendation list, then:
             $num\_of\_succ \leftarrow (num\_of\_succ + 1)$.
End of For
Return {num_of_succ, $total\_cycles/|S|$}

Figure 2: The simulation test procedure.

At Step 1, given a tester's recommendation session the simulated critiques list is retrieved from the *USER_CTZ_SEQ* case component of the original session. For the *[full_val_sel]* critique-simulation method, the simulated critiques list is constructed by concatenating the tester's critiques sequence and the values in the selected product for the features that had not been criticized in the tester's session.

At Step 2, the simulated recommendation list is produced as follows.
- For the *repMR* variant the computation of the simulated recommendation list consists of the two phases, filtering by Q_L and ranking by similarity to $<p, w>$, as discussed in subsection 2.2.

- The *repLQ* variant produces the simulated recommendation list by filtering the products by the logical query Q_L. If the execution of Q_L returns an empty result set, then no relaxation is performed and the simulated session is failing. The products satisfying Q_L were included in the simulated recommendation list in the same order as they were stored in the database.

- The *repFP* system variant produces the simulated recommendation list by ranking the products by similarity to $<p, w>$.

A simulated recommendation session terminates, at Step 3, if:
- the selected product appears in the view window of the simulated recommendation list (i.e., a success); or

- all the critiques in the simulated critiques list have been exploited, but the selected product is still not found (i.e., a failure); or

- the simulated recommendation list at the current cycle is empty (i.e., a failure). Note that in the *repLQ* variant automatic constraint relaxation is not supported because it would require to change a logical failing constraint into a favorite pattern feature.

At Step 4, one of the five critique-simulation methods discussed above is applied to simulate a critique. The simulated critique is then used by the system to adapt the query representation. The adaptation rules depend on 1) what query representation approach is used and 2) what critique-simulation method is applied.

For the *repMR* system variant, the adaptation rules are exactly as defined in section 2.2. In particular, a critique stated as a must causes an update of the respective logical constraint in Q_L, whereas a critique stated as a wish is used to update the respective favorite element in p.

For the *repLQ* system variant.
- If the *[rpt_ctz]* critique-simulation method is applied, a must critique makes an update of the respective logical constraint in Q_L, whereas a critique stated as a wish is converted to a must condition and then incorporated in Q_L.
- If one of the *[val_sel]*, *[sim_ctz]*, and *[sim_sel]* critique-simulation methods is applied, the criticized feature's value in the base product is used to create a new (feature type dependent) constraint which is then incorporated in Q_L. A base product may be the selected one, or the product most similar to the one criticized in the original session, or the product most similar to the selected one, depending on what critique-simulation method is applied.

For the *repFP* system variant.
- If the *[rpt_ctz]* critique-simulation method is applied, a wish critique makes an update of the respective favorite

element in *p*, whereas a must critique is converted to a wish condition and then incorporated in *p*.

- For the remaining critique-simulation methods, the criticized feature's value in the base product is used to update the respective favorite element in *p*.

4.2 Test Results and Discussion

An analysis of the logs of the testers' recommendation sessions showed that they made their critiques to, and selected, products ranked among the first eleven positions in the recommendation lists. Therefore, we tested the ten combinations shown in Table 1 for eleven view-window sizes, from 1 to 11. We present here the results obtained for the view-window size of 5, due to paper space limit and also because most of the testers limited their browsing to the top 5 items in recommendation lists.

In Figure 3, we first look at the success rates of the system variants. The results show that *repMR[rpt_ctz]* has success rate 36.36% higher than *repLQ[rpt_ctz]*. *repLQ[val_sel]* tends to have the highest success rate for the *repLQ* variant, since this combination replays the critiques using the criticized features' values in the selected product. When compared to *repLQ[val_sel]*, *repMR[rpt_ctz]* is still 36.36% higher with respect to the success rate. The main reason why the *repLQ* query-representation approach has a low success rate is because of the "failing queries" problem. When this happens, the simulated session is considered to fail.
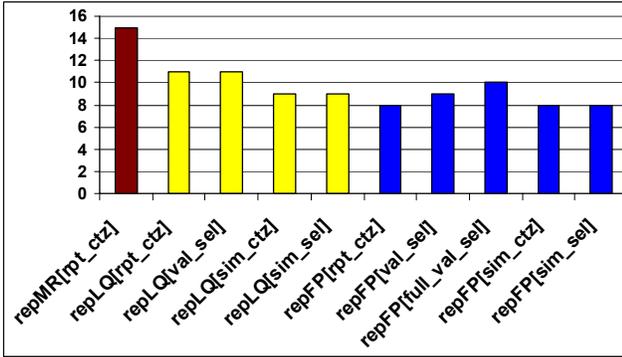

Figure 3: Comparison on the number of successfully simulated sessions, for the view-window size of 5.
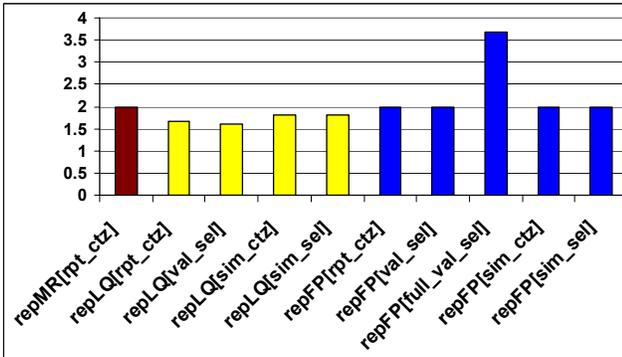

Figure 4: Comparison on the average recommendation length, for the view-window size of 5.

The important advantage of the *repLQ* representation approach is that if it can converge to the selected product, the convergence is fast (i.e., the recommendation length is short). This can be observed in Figure 4, by comparing *repMR[rpt_ctz]* and *repLQ[rpt_ctz]* with respect to the average recommendation length. In particular, the average recommendation length of *repMR[rpt_ctz]* is 19.76% longer than that by *repLQ[rpt_ctz]*. This compensates for the increment in the success rate. Similar results can be observed when comparing *repMR[rpt_ctz]* to *repLQ[sim_ctz]* and *repLQ[sim_sel]*. We note that the simulated recommendation dialogues are rather simple, since the selected items can be found on average within two recommendation cycles. This average recommendation length is (much) shorter than those found in previous evaluations of web critique-based RSs [3, 6, 7, 12]. The reason of that difference is that on mobile devices users tend to spend (much) less time and effort when searching for desired items.

Comparing now the success rates of *repMR* and *repFP* (i.e., the variant represents the user preferences with only a similarity query), one can see that *repMR[rpt_ctz]* achieves a much higher success rate over *repFP[rpt_ctz]*, i.e., 87.5% higher. When comparing *repMR[rpt_ctz]* to *repFP[val_sel]*, *repFP[sim_ctz]*, and *repFP[sim_sel]*, similar large increments in the success rate are observed.

Amongst the replaying methods used for *repFP* (see Table 1), *repFP[full_val_sel]* tends to have the highest success rate, since it

- simulates the critiques using the criticized features' values in the selected product, and
- uses the remaining features' values in the selected product even after all the critiques have been exploited (i.e., simulated).

When compared to *repFP[full_val_sel]*, *repMR[rpt_ctz]* has still a success rate 50% higher. Furthermore, in Figure 4 we can see that *repMR[rpt_ctz]* has a shorter average recommendation length compared to *repFP[full_val_sel]*, and an equal average length compared to the remaining *repFP*-based combinations.

The reason why the replaying methods using the *repFP* query representation approach have lower success rates is because in many simulated sessions the system was not able to push the selected product towards the top of the recommendation list. In other words, the *repFP*-based combinations sometimes fail to isolate the selected product from the others. With just very few (e.g., one or two) wish critiques the system using the *repFP* query representation may fail in differentiating the selected product from the others.

In summary, if the *repLQ* system converges at the selected product, the convergence is fast. However, in many cases the system faces the "failing queries" problem that causes a recommendation failure (and this cannot be repaired as we mentioned above because only logical constraints are supported). On the other hand, the system using the *repFP* query representation approach has no failing queries, but often it makes longer recommendation sessions (i.e., more critiquing cycles) and in some cases it may fail at isolating the selected product from the others, or may not be able to push it up in the recommendation list. The *repMR* system, which integrates both logical and similarity queries, has the benefits of the two individual query representation approaches (logical and similarity) and seems to be

immune from their main disadvantages. The *repMR* system, therefore, should always converge at the selected product, and the convergence is rather fast.

5. RELATED WORK

Though a number of critique-based recommendation approaches have been introduced in the literature, only some of them have been evaluated with real users [5, 6, 9, 10] or by simulation tests [3, 7, 12]. Previous off-line tests, except the one presented in [3], have focused mainly on the recommendation length (i.e., the number of cycles, or the number of products presented). In previous research, such as [7], it was assumed an unlimited number of critiques that can be simulated in a session, that is, critiques are simulated until the target item is found in the output list. Hence, a simulated session never fails. But, in practice (mobile) users are often impatient, and rarely express many critiques in a recommendation session.

Simulating user critiques, which plays a very important role in off-line evaluations of critique-based RSs, has been done in a random way. In particular, at a recommendation cycle the criticized product is randomly selected [12] or is the product in the recommendation list that is most similar to the target one [7], and also the criticized feature is randomly selected [7]. Furthermore, no off-line evaluations of previous critique-based RSs, except the one presented in [3], have exploited a dataset of real users' recommendation sessions; they have mainly exploited the product (e.g., wine, PC, or travel) catalogs, where each product is used to generate the initial query and identify the target.

6. CONCLUSIONS

In a previously conducted live users evaluation we proved the effectiveness of our critique-based recommendation methodology for supporting on-the-go users in decision making of product selection. In this paper, we report on an experimental study aimed at comparing different user-query representation approaches. In particular, we compared a composite query representation approach, which employs both logical and similarity queries, with two representations, each one using only one of the two queries. To perform this comparative evaluation, we defined a simulation procedure, and used it to replay the testers' recommendation sessions recorded in the live users evaluation. The results of the simulation tests showed that the query representation using both logical and similarity queries resulted in a better recommendation quality over a simpler query representation. The proposed simulation procedure replays real recommendation sessions and simulates user critiques in different ways. This simulation procedure could be useful for other researchers who want to test their critique-based RSs off-line.

As future work we want to introduce some query repair approach for the *repLQ*-based variant, so that we can relax failing queries in the off-line evaluations of *repLQ*. Moreover, the presented empirical evaluation assumes a re-application of real users' critiques in their historical (original) order. Since different products shown may cause users to make different critiques, we plan to test other replaying strategies that will prioritize the critiquing according to user preferences.

7. REFERENCES

[1] D. Bridge, M. Göker, L. McGinty, and B. Smyth. Case-based recommender systems. *Knowl. Eng. Rev.*, 20(3): 315–320, 2006.

[2] R. Burke. Interactive critiquing for catalog navigation in e-commerce. *Artif. Intell. Rev.*, 18(3-4): 245–267, 2002.

[3] L. Chen and P. Pu. Preference-based organization interface: Aiding user critiques in recommender systems. In *Proc. 11th Int'l Conf. User Modeling*, pp. 77–86, 2007.

[4] Z. Jiang, W. Wang, and I. Benbasat. Multimedia-based interactive advising technology for online consumer decision support. *Communications of the ACM*, 48(9): 92–98, 2005.

[5] C. Y. Kim, J. K. Lee, Y. H. Cho, and D. H. Kim. Viscors: A visual-content recommender for the mobile Web. *IEEE Intell. Syst.*, 19(6): 32–39, 2004.

[6] K. McCarthy, L. McGinty, B. Smyth, and J. Reilly. A live-user evaluation of incremental dynamic critiquing. In *Proc. 6th Int'l Conf. Case-Based Reasoning*, pp. 339–352, 2005.

[7] L. McGinty and B. Smyth. Adaptive selection: An analysis of critiquing and preference-based feedback in conversational recommender systems. *Int'l. J. Electronic Commerce*, 11(2): 35–57, 2006.

[8] Q. N. Nguyen and F. Ricci. User preferences initialization and integration in critique-based mobile recommender systems. In *Proc. 5th Int'l Workshop Artif. Intell. in Mobile Syst.*, pp. 71–78, 2004.

[9] P. Pu and P. Kumar. Evaluating example-based search tools. In *Proc. 5th ACM Conf. Electronic Commerce*, pp. 208–217, 2004.

[10] P. Pu and L. Chen. Integrating tradeoff support in product search tools for e-commerce sites. In *Proc. 6th ACM Conf. Electronic Commerce*, pp. 269–278, 2005.

[11] F. Ricci and Q. N. Nguyen. Acquiring and revising preferences in a critique-based mobile recommender system. *IEEE Intell. Syst.*, 22(3): 22–29, 2007.

[12] H. Shimazu. ExpertClerk: Navigating shoppers buying process with the combination of asking and proposing. In *Proc. 17th Int'l Joint Conf. Artif. Intell.*, pp. 1443–1450, 2001.

[13] C. A. Thompson, M. H. Göker, and P. Langley. A personalized system for conversational recommendations. *Artif. Intell. Research*, 21: 393–428, 2004.

[14] M. van Setten, S. Pokraev, and J. Koolwaaij. Context-aware recommendations in the mobile tourist application COMPASS. In *Proc. 3rd Int'l Conf. Adaptive Hypermedia and Adaptive Web-Based Syst.*, pp. 235–244, 2004.

[15] Z. Yu, X. Zhou, D. Zhang, C.-Y. Chin, X. Wang, and J. Men. Supporting context-aware media recommendations for smart phones. *IEEE Pervasive Comp.*, 5(3): 68–75, 2006.

[16] B. Zhou, S. C. Hui, and K. Chang. Enhancing mobile Web access using intelligent recommendations. *IEEE Intell. Syst.*, 21(1): 28–34, 2006.