Adapting the Interaction State Model in Conversational Recommender Systems

Tariq Mahmood
University of Trento
Trento, Italy
mahmood@dit.unitn.it

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

ABSTRACT

Conventional conversational recommender systems support interaction strategies that are hard-coded into the system in advance. In this context, Reinforcement Learning techniques have been proposed to learn an optimal, user-adapted interaction strategy, by encoding relevant information as features describing the state of the interaction. In this regard, a crucial problem is to select this subset of relevant features from a larger set, for any given recommendation task. In this paper, we tackle this issue of state features selection by proposing and exploiting two criteria for determining feature relevancy. Our results show that adding a feature might not always be beneficial, that the relevancy is influenced by the user behavior, and also by the numerical reinforcement signal which is exploited by the adaptive system for learning the optimal strategy. These results, obtained in off-line simulations and in a simplified scenario, were exploited to design an adaptive recommender system for an online travel planning application.

Categories and Subject Descriptors

H.4.2 [**Information Systems Applications**]: Types of Systems—*Decision Support*

General Terms

Design, Algorithms, Experimentation, Human Factors, Performance

Keywords

Conversational Recommender Systems, Reinforcement Learning, Adaptivity, State Features Selection

1. INTRODUCTION

Conversational recommender systems [2, 9, 18, 11] are interactive online applications which adopt the conversational style of real-life dialogues. They assist users in their information-seeking tasks by guiding them through complex product spaces. This approach departs from alternative recommendation technologies, such as collaborative filtering or content-based filtering, where the recommendations are computed once, and for all, when the user request a recommendation, and they cannot be revised either by the system or by the user [1]. Quite diversely, in a conversational system, at each dialogue stage, the system selects the conversation action, from a set of available system actions, that it considers more useful in that specific stage. For instance, it could query the user for her preferences or choose to present some information. The particular action selection is specified by the system's dialogue strategy. For instance, suppose that a conversational recommender is queried for a set of hotels matching some user preferences. Then, it could employ the following two strategies (among others): 1) query the user for her preferences in order to identify a small product subset, or 2) recommend a set of products, and exploit the user feedback (e.g., critiques) to refine future recommendations. Conventional conversational systems typically employ dialogue strategies that are hard-coded inside the system and are followed rigidly during a session [9, 18, 11].

A major limitation of this approach is that there could be numerous strategies for a given recommendation task. Furthermore, system designers might need to evaluate several strategies before they discover one which is "suitable" for the users. This is an infeasible process in terms of budget, time, and task force. Hence, in a previous paper [6], we have tackled these requirements by proposing a new type of a recommender system which is able to autonomously improve a hard-coded strategy in order to learn and adopt an optimal one. Our system learns the optimal strategy through Reinforcement Learning (RL) techniques, basically by executing actions through trial-and-error, while interacting with the users. In doing so, the system maximizes a reward function, that models of how much benefit the user is supposed to get from each interaction step. Hence, in our approach, the system is able to *decide autonomously* which action to execute at each stage of a recommendation session, and hence, is able to adopt different strategies during the session, until it discovers the optimal strategy. In [6], we have evaluated our approach by applying it within the NutKing travel recommender system, and specifically, within a particular type of interaction, i.e., the incremental request of product attributes, in order to further constrain a query to a product catalogue [9]. Here, the system should decide, whether to ask the user for additional product attributes in order to retrieve a small set of candidate products, or to let the user autonomously take this decision and manually change her query.

Our initial results showed that the recommender system can improve an initial policy that, in a rigid way, asks the user for additional preferences, only when the result set size of her query is greater than a given threshold value. In fact, we showed that different optimal policies are learned when the agent estimates in a different way the cost it incurs (through a negative reinforcement signal) at each non-terminal (non-goal) stage of the interaction. Specifically, for (negatively) large costs, the agent learns to take actions which are likely to reduce the number of interaction stages (in this case, the action of asking additional preferences), i.e., speed-up the process of reaching the user's goal state. On the other hand, for small costs, the agent doesn't attempt to speed things up; it learns actions which just guarantee that, in the long run, the user's goal would finally be achieved (i.e., in this case, the showing of the results to the user, and allowing her to manually modify her query). Furthermore, we showed that different policies are learned when the system models differently the willingness of the user (using the RL-based parameter γ) to continue the interaction. Specifically, for unwilling users, the system learns to speed-things up by requesting the user for additional attributes, and for willing users, it just allows the user to manually modify her query. Hence, we successfully validated our adaptive recommendation approach under two different (RL-based) parametric configurations.

In this paper, we extend our previous work and show that the learnt optimal policy is actually influenced by the information contained in the state representation, which is encoded as values of one or more *state features*. The problem is that an extensive information set, related to the state of the user and the state of the system activity, could result in a computationally infeasible policy learning problem and therefore, the selection of the *relevant* state features for a given recommendation task becomes crucial. In a previous paper [7], we have addressed several issues related to feature relevancy. Basically, we introduced two relevancy criteria, and used them to determine the relevancy of adding four different features to a *baseline* representation. For each feature we investigate its relevancy under different (simulated) user model behaviors. Our results indicated that the relevancy is influenced by changes in the user behavior. In this paper, we extend our work in [7], and, by exploiting a similar experimental approach (i.e., adding a set of features separately to a *baseline*), we investigate, for each added feature, it relevancy under different (simulated) user model behaviors and also under two different reward functions. Our results again indicate that adding a feature might not always be relevant for a given task. They also show that the relevancy is influenced both by changes in the user behavior and by changes in the reward model. In this paper, we execute also a *forward feature selection* procedure, i.e., we continue to add relevant features in order to determine the relevancy of adding the remaining ones. The results show, in this case, that no additional feature can further improve the optimal policy and that the baseline representation can be improved by just adding one single feature representing the frequency of the system's (attribute) suggestions, in order to constrain the user query. Practically these results guided the application of our recommendation approach in an online context, where we exploited our approach in order to improve an initial (fixed) policy, and learn an optimal policy for the Travel Planner (`TP`) tool of the Austrian tourism web portal (Austria.info), in the context of the $_{et}$*Packaging* project funded by Austrian Network for E-Tourism (ANET) [8].

In the rest of this paper, we will initially describe our adaptive recommendation model (Section 2). Then, in Section 3, we shall present our state model, i.e., our proposed five state feature sets (*state representations*), the five simulated user models and two reward models. Then, we shall detail our evaluation approach (Section 4), wherein we will explain our evaluation procedure (Section 4.1), and our two proposed criteria for determining state feature relevancy (Section 4.2), namely *Policy Diversity* and *Policy Value*. Later on, in Section 5, we shall present our results wherein we shall determine the relevancy of our proposed state representations under *Policy Diversity* (Section 5.1) and *Policy Value* (Section 5.2). Then, in Section 6, we will describe our evaluation procedure and results for the forward feature selection procedure. In Section 7, we will exploit our previous results in order to describe the evaluation setup, and some results, for our ongoing online experiment. Finally, we will discuss the related work in Section 8 and we will present our conclusions in Section 9.

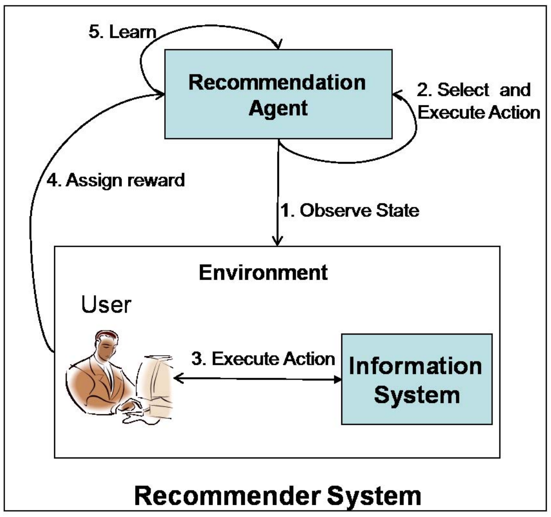
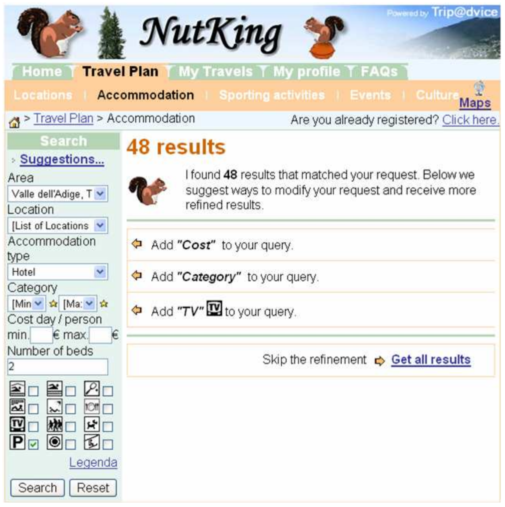


Figure 1: Adaptive Recommender Model

2. THE RECOMMENDATION AGENT

Our proposed recommendation model, shown in Figure 1, basically comprises two entities, namely the *Information System* (IS) and the *Recommendation Agent* (RA)) (we refer the reader to [6] for a more detailed description). The function of IS is to serve user requests like displaying a query result, showing most popular selections etc. The RA is the adaptive entity which actively assists the user by controlling the IS, and providing them (through the IS) the relevant information at appropriate stages of the interaction session. This process of assistance is dictated by the strategy of the Recommendation Agent. In our model, we learn the optimal strategy, i.e., the policy that maximizes the cumulative reward (more details will be provided in the next section), by using Reinforcement Learning (RL) techniques [13]. In the context of RL, we shall use the term *optimal policy* rather than optimal strategy, where a *policy* simply specifies how the strategy is implemented in terms of the system's actions. In order to learn the optimal policy, we consider the `RA` as a decision-making Agent, which executes, through the IS, some trial-and-error interactions with the user, who is part of the Agent's *Environment*.

Initially, the Agent has a pre-specified *initial policy* and the goal is to improve this initial policy in order to learn an optimal one. To this end, at each stage, and after the user has taken some action, the Agent observes the current situation, or *state*, of the system (Step 1 in Figure 1). The observed system state could include information about the current state of the user, the on-going interaction, or even about the state of the Agent itself. This is the information which the Agent needs in order to learn the optimal policy. In one or more system states, multiple system actions could be available to the Agent for execution. We label these states as the *System Decision Points* (SDPs). The Agent tries out (selects) different system actions during the interaction (Step 2). These actions are different



48 results

I found **48** results that matched your request. Below we suggest ways to modify your request and receive more refined results.

Add ***"Cost"*** to your query.

Add ***"Category"*** to your query.

Add ***"TV"*** to your query.

Skip the refinement **Get all results**

Figure 2: Query Tightening Suggestions

from the action specified by the Agent's initial policy. For each action execution (Step 3) (and the ensuing user response), the Agent receives a numerical reward (*reinforcement signal*) from the Environment (Step 4), informing it whether its previous action was acceptable for the user or not. The magnitude and the sign of this reward is determined by the real, or hypothesized, level of user acceptability. For instance, a large positive reward (+10) could be assigned for actions that are most favorable for the users, and a large negative reward could be assigned for least favorable ones (-10).

As the interaction proceeds, the Agent exploits the received rewards in order to learn to avoid taking unacceptable actions and to take acceptable ones (Step 5). The RL procedure guarantees that this process shall eventually lead the Agent to learn the optimal action at each SDP, i.e., learn the optimal policy. We note that in the *non-SDP* states, only a single action is available for execution, which we consider as the optimal action.

From the above discussion, it is apparent that the optimal policy is actually influenced by the information in the state representation, which is encoded as values of one or more *state features*. Typically, an extensive information set, concerning the state of the user and the system activity, could be identified for any recommendation task. Considering that it is computationally infeasible to learn policies with a large number of state features [13], the selection of the *relevant* state features for a given recommendation task becomes crucial. As we mentioned in the Introduction, in the rest of this paper, we are addressing several issues related to this problem of feature relevancy.

3. SYSTEM MODEL

In order to address the feature relevancy problem, we will now present our state model representation, the five (simulated) user behavior models, and the two numerical reward models we have used in the experiments. For our evaluation, we will use the scenario presented in [6]. Specifically, we simulate a user (details provided later on) who formulates a logical query to a product catalogue of a real recommender system (NutKing travel plan recommender system [9]), by constraining zero or more product features. If the query retrieves a large result set, i.e., greater than a certain threshold, the system's default policy is to suggest a set of product attributes for *tightening* the current query, i.e., the user can provide a preferred value for one of these attributes. For example, Figure 2 shows a

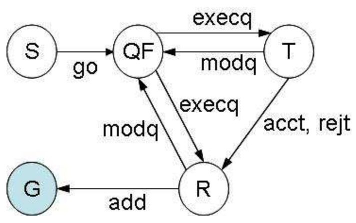


Figure 3: User Interaction Graph

State Feature	Discretized Feature Values
Page-UserAction (*PUA*)	{*S-go, QF-execq, T-acct, T-rejt, T-modq, R-modq, R-add, G*}
Current Result Size (*CRS*)	{*s=[0,20], m=]20,50], l=]50,100], vl=]100,INF]*}
Expected Result Size (*ERS*)	{*s=[0,20], m=]20,50], l=]50,100], vl=]100,INF]*}
Freq Tight Sugg(*FTSugg*)	{*s=[0,2], m=]2,4], l=]4,INF]*}
Number Int Stages(*NStages*)	{*s=[0,3], m=]3,6], l=]6,INF]*}
User Tighten Resp(*UserTResp*)	{*accept, mixed, reject*}

Table 1: State Features (*INF*=Infinity, s=small, m=medium, l=large, vl=very large)

screen shot of the NutKing recommender, where the user has constrained and executed a query (in the left-most column) and a large result size of forty eight (48) products has been retrieved (note that in NutKing, we set the threshold to 20, while for the current experiments, we have set it to 50). Hence the system has suggested the following three features to the user: "Cost/day of the hotel", "Hotel category" e.g. 3-star, 4-star etc, and "TV" i.e. whether the user would like to have a TV in her room (or not). We label this system action as *Suggest*. Otherwise, the system executes the query and shows all the results (possibly a large number) to the user. We label this system action as *Execute*. In this evaluation, we focus just on these two actions, as these are the only actions available in the states where a choice is to be made, i.e., in the System Decision Points (SDP). In fact, there are many other system actions in NutKing but the policy which specifies these actions is rigid.

3.1 State Representations

Given a state representation, the system's job is to learn the optimal policy, i.e., in each SDP (for each possible assignment of the state variables), decide whether to select *Suggest* or *Execute*. For our current study, we will select a baseline state representation (*Baseline*). Then we will add four different features separately to this baseline, and determine the relevancy of adding each new feature. Our selected state feature set and the corresponding discretized values, are shown in Table 1, and the alternative state models are shown in Table 2, where we define five state models $R = \{Baseline, Rep1, Rep2, Rep3, Rep4\}$. We say that R is our representation set.

We now present our rationale for selecting the state features in Table 1. The feature *PUA* (Page and User Action) depicts the combinations of the web pages (of the NutKing recommender) and the possible user actions in these pages (Figure 3). The five web pages are the welcome page *S*, the page *QF* in which the user formulates queries, the result page *R* in which the products are displayed, the page *T* where tightening is suggested (Figure 2), and *G* which depicts the addition of a product to the cart, i.e., the user's goal state. Furthermore, in *S*, a user can only go to the page *QF* (depicted

Baseline	{*PUA, CRS*}
Rep1	{*PUA, CRS, ERS*}
Rep2	{*PUA, CRS, FTSugg*}
Rep3	{*PUA, CRS, NStages*}
Rep4	{*PUA, CRS, UserTResp*}

Table 2: Different State Representations

by the combination *S-go*), wherein she executes her queries (*QF-exec*). In *T*, she can either accept a suggestion (*T-acct*), reject it and request to see all results (*T-rejt*), or modify her query manually (*T-modq*). Finally, in *R*, the user can either add a product (*R-add*), or manually modify her query (*R-modq*).

Also, *CRS* (Current Result Size) is the result set size of a user query. We included *PUA* and *CRS* in *Baseline* as these features provide the minimum information required by the system in order to decide between *Suggest* and *Execute*.

The feature *ERS* (Expected Result Size) is the system's estimation (based on statistics on the product catalogue) of the result set size if one of the suggested product attributes is actually exploited by the user for tightening her query. In fact, the product attributes are ranked according to an attribute selection method (see [9] for details on tightening). The state model that results by adding ERS to $Baseline$ is labeled as $Rep1$. This was used previously in [6], but then, we did not investigate the relevancy adding ERS, as we shall do now.

Moreover, in a previous online evaluation, we found that users were not too willing to respond to the action *Suggest*, i.e., they constrained a suggested feature only 26% of the time [9]. Thus, it is probably necessary for the system to know the frequency of times *Suggest* has been executed (feature *FTSugg*) and the general response of the user to *Suggest* (feature *UserTResp*), where *accept* implies that the user has always accepted a suggestion whenever *Suggest* has been executed, *reject* means that the user has never accepted any constraint suggestion, and *mixed* implies that the user has accepted tightening only a certain percentage of the time (see Table 1). Besides this, we believe that the number of session stages which have elapsed up to the current stage (feature *NStages*) might affect how the user responds to *Suggest*, e.g., users might accept suggestions earlier on but might get frustrated as the session prolongs.

3.2 User Models

In this section, we shall describe the five different user models used in our evaluation. These models differ in how the simulated user responds to the system action *Suggest*, which occurs in the following three cases (please refer to [6] for the full description of the PUA values and system actions):

- (Case 1) When *PUA=QF-execq*, i.e., when the user formulates and executes a query, how does she actually *constrain* the features in (or modify) her current query.
- (Case 2) When the system executes a query, whether the user will add the test item to the cart (*PUA=R-add*) or modify her query (*PUA=T-modq*).
- (Case 3) When the system suggests tightening, whether the user will decide to modify her current query (*PUA=T-modq*), or to accept tightening (*PUA=T-acct*), or to reject tightening (*PUA=T-rejt*).

Let us now describe these three situations. For Case 1, at the beginning of each simulated session, we sort the attributes of the test product, i.e., the product that the simulated user is supposed to search and add to her cart (see Section 4.1 for the details), according to their frequency of usage as observed in real-user interactions with NutKing [9]. This ordering simulates the ranking of the product attributes according to the users' interest in them. Then, we use this order to choose the first attribute to constrain in the initial query, and also to incrementally select the next attribute to constrain, when the (simulated) user must modify the current query (*PUA=QF-execq*).

In Case 2, the user will add the test item to the cart (*PUA=R-add*) if the test item is found in the top N items returned by the query (we set N=3). Otherwise the user will modify her current query (*PUA=R-modq*).

For Case 3, we model five user behavior models, which differ only in their response to *Suggest*: 1) a generic user model (*GUM*) which was used in [6]; 2) a willing user (*WillUM*), i.e., a user who is always willing to accept tightening; 3) a moderately-willing user (*ModwillUM*), i.e., a user who is willing to accept tightening only sometimes during her session; 4) an unwilling user (*UnwillUM*), i.e., a user who is never willing to accept tightening; and 5) all the above users (*AllUM*), i.e., a model which simulates the behavior of a population of users. We define the set $UM = \{GUM, WillUM, UnWillUM, ModwillUM, AllUM\}$.

1. ***GUM***: This model simulates a generic response behavior to *Suggest*: the user accepts tightening if any one of the suggested attributes has a *non-Null* value in the test item (i.e., the simulated user has a preference on that attribute), and this attribute is also the next preferred attribute of the user (according to the attribute usage order). If the user doesn't accept tightening and the result size is smaller than 50, i.e., when $CRS = small$ or when $CRS = medium$, then the user rejects it and executes the original query. In the remaining cases, i.e., when CRS is either $large$ or $verylarge$, and when acceptance cannot be simulated, the user does not accept tightening and autonomously modifies her query (as in Case 1) (*T-modq*).

2. ***WillUM***: The user accepts tightening if any of the suggested attributes has a *non-Null* value in the test item, *even if it is not her next preferred attribute*. This latter condition allows us to simulate the user's "willingness" to accept tightening (as compared to *GUM* where the user accepted tightening if the suggested attribute is also her next preferred one). If acceptance cannot be simulated, the user rejects tightening or manually modifies her query similarly to the corresponding behavior in *GUM*.

3. ***ModwillUM***: The user considers accepting tightening only 26% of the time that *Suggest* is executed during a session. Hence, *ModwillUM* simulates the real-user response to *Suggest* [9]. We make a random selection (from a uniform distribution) of the situation in which the user will accept tightening. If the user considers accepting tightening, acceptance is simulated similarly to the behavior in *WillUM*. If acceptance cannot be simulated in this case, or if the user does not consider accepting tightening (74% of the time), then the user either rejects tightening or manually modifies her query as in *GUM*.

4. ***UnwillUM***: The user never accepts tightening; if $CRS \in \{small, medium\}$, the user rejects tightening and executes

R1	*+5*	*The user adds a product to her cart*
	-0.05	*An interaction stage elapses*
R2	*+5*	*The user adds a product to her cart*
	+1	*A result page is shown to the user*
	0	*Otherwise*

Table 3: Reward Models

the query. Otherwise, if $CRS \in \{large, verylarge\}$, the user modifies her query as in *GUM*.

5. ***AllUM***: This model simulates the behavior of a population of users: each time *Suggest* is executed, we randomly select (from a uniform distribution) and simulate one from amongst the above four user behaviors.

3.3 Reward Models

In this section, we will present our two numerical reward models to be used in our evaluation, namely $R1$ and $R2$ (See Table 3).

In $R1$ we assign a large positive reward $(+5)$ to the Recommendation Agent (see Section 1) in the interaction stage where the user adds a product to her cart, as this is the primary goal of the interaction. Moreover, we assign a small negative reward (-0.05) to the Agent in all the stages where no cart addition occurs; the rationale is that until some product is selected, the Agent should be *punished* with a small negative reward, in order to convey that the user's goal has (as yet) not been achieved.

Conversely, in $R2$, besides assigning a large positive reward $(+5)$ for addition to cart, we also assign a small positive reward $(+1)$ to the stage in which a result page, showing product information, is displayed to the user. The rationale here is that we consider the displaying of a result page as one of the system's sub-goals, since it provides the necessary information for achieving the main goal, i.e., adding a product at the end of the session. In all the remaining stages, no reward (i.e., a reward equal to 0) is assigned to the Agent.

4. EVALUATION APPROACH

4.1 Evaluation Procedure

Basically, we carried out the evaluation by simulating a sequence of user-system interaction sessions or *trials*. For each specific session, we exploit a simulated user behavior model $um \in UM$ (see Section 3.2) in order to specify how the user responds to system actions during this session. Each trial is composed of some interaction stages, and simulates the user as incrementally modifying a query in order to finally add a product to her cart. We ran this procedure for a set of randomly selected products from the catalogue. In this simulation we performed a leave-one-in selection, i.e., for each trial we selected a product *t*, in which values have been specified for product attributes, i.e., $t = (v_1, ..., v_n)$. We call *t* as the *test item*, and we simulated a user as searching for this item. The values used by the user to modify her query, e.g., when a suggested feature is accepted are those of the test item. Note that not all the attributes in *t* have a specified value, i.e., some of these v_i may be *Null* (these are attributes for which the user has no preference).

We will now define our evaluation goal. Given our *Baseline* representation and the set *R* of alternative state representations (Section 3.1), our goal is to determine, whether each representation is more relevant, or *beneficial*, for the system as compared to the *Baseline* representation. This "benefit" is determined through two proposed criteria for determining feature relevancy (see the next section). Furthermore, we plan to investigate the relevancy by considering each representation $r \in R$ (Section 3.1) under each user model behavior $um \in UM$ (Section 3.2). The rationale is to investigate whether a representation, which is relevant for a particular group of users, is also relevant for some other group(s). The result is important because online user behavior is typically quite diverse. So, in order to select a relevant representation for a given recommendation scenario, we must determine how the relevancy is actually influenced by these different behaviors. Moreover, for each State Representation and User Model combination, we shall investigate relevancy under each reward model, i.e., $R1$ or $R2$ (see Section 3.3). To this end, we simulate the aforementioned evaluation procedure for each "State Representation - User Model - Reward Model" combination. The aim is to investigate whether feature relevancy for a given "State Representation - User Model" combination is influenced by changes in the reward model. The result is important because more than one reward model could be selected for a given policy-learning task.

In order to cater for these requirements, we will initially learn the optimal policy (OP) for all possible "State Representation - User Model - Reward Model" combinations. As we have a total of five representations (Table 2), five user models (Section 3.2) and two reward models (Table 3), we learn a total of $5 * 5 * 2 = 50$ optimal policies. While learning each policy, we set the Agent's initial policy as the default policy of NutKing, i.e., execute the query when $CRS \in \{small, medium\}$, and suggest tightening when $CRS \in \{large, verylarge\}$. For all the learnt policies, we are interested only in the SDP states, i.e., those where *PUA=QF-execq*, and for all the state combinations listed in the rest of the paper, we assume that *PUA=QF-execq*.

4.2 Proposed Relevancy Criteria

Our first relevancy criterion, called *Policy Diversity*, is based on a comparison of the actions prescribed by the optimal policies. Here, *for each User Model-Reward Model combination*, we compare the actions of the optimal policy (OP) for the *Baseline* representation, with the four optimal policies obtained under the alternative representations in *R*. We do this comparison for the states where *PUA=QF-exceq*, i.e., where the system must decide whether to execute the query or suggest tightening. In these states CRS may take the possible values $small$, $medium$, $large$, or $verylarge$. Let OP_{base} be the optimal policy for *Baseline* representation and OP_r the optimal policy for $r \in R$. Let us further suppose that for some user model $um \in UM$ and for some reward model $rm \in RM$, we compare OP_{base} with OP_r. Let us define CRS^r_{small} as the set of all states for the representation r where $PUA = QF - exceq$ and $CRS = small$ (we omit the PUA state feature because in this work we are interested only in the states where *PUA = QF-exceq*). A similar definition can be given for $CRS \in \{medium, large, verylarge\}$. Also, suppose that the optimal action for the single state in $CRS^{Baseline}_{small}$ (there is just one state since in the baseline there are only two features, PUA and CRS) is $Execute$ but for *one* of the states in CRS^r_{small} the optimal action under OP_r is $Suggest$. Based on this difference, we say that the representation r is *relevant* for $CRS = small$, as it is allowing the system to modify its optimal behavior by learning different optimal actions. More in general, we shall say that, for *Policy Diversity r* is

- *relevant* if it is relevant for one or more values of the variable *CRS*, and
- *irrelevant* if it is not relevant for any value of the variable *CRS*.

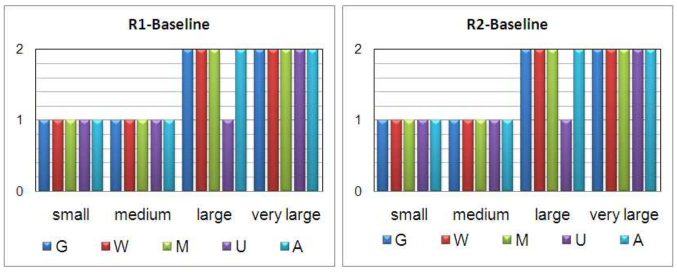


Figure 4: Learnt Optimal Policies for *Baseline* under reward model *R1* and *R2* (G=*GUM*, W=*WillUM*, M=*ModwillUM*, U=*UnwillUM*, A=*AllUM*), 1 = *Execute*, 2 = *Suggest*

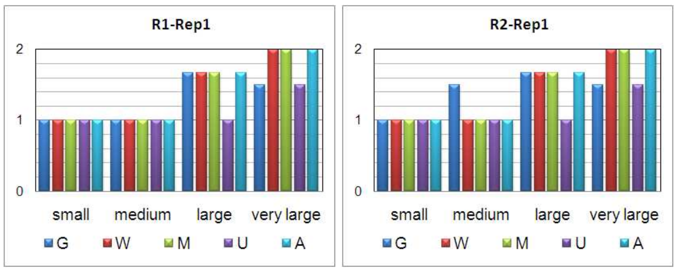


Figure 5: Learnt Optimal Policies for *Rep1* under reward model *R1* and *R2* (G=*GUM*, W=*WillUM*, M=*ModwillUM*, U=*UnwillUM*, A=*AllUM*, 1 = *Execute*, 2 = *Suggest*)

Our second relevancy criterion, called *Policy Value*, is based on an *evaluation* of the optimal policies, i.e., on determining the total reward which the system can accumulate while employing a particular OP. The rationale is that, for a given *User Model-Reward Model* combination and for a given representation $r \in R$, if OP_r allows the system to (significantly) accumulate more reward than OP_{base}, then r is a relevant representation.

We apply Policy Value by selecting a set of 300 items from the product catalogue. For each item, we simulate an interaction session and calculate the total reward that the system accumulates in that session. At the end, we compute the Average Cumulative Reward (*ACR*) for all the 300 sessions. When we evaluate OP_r which has been learnt under a user model $um \in UM$ and a reward model $rm \in RM$, we mean that during the session, the system takes actions according to OP_r with the user responses being generated through um, and the Agent being rewarded according to *rm*. For each *User Model - Reward Model combination*, we evaluate OP_{Base} and the four optimal policies obtained for each representation in R. We use the paired t-test in order to determine the significance of the estimated rewards. Let us define the *ACR* obtained after evaluating OP_r as ACR_r, and the *ACR* obtained after evaluating OP_{base} as ACR_{base}. Then, for *Policy Value*, we say that *r* is:

- *relevant* if ACR_r is significantly greater than ACR_{base}, and
- *irrelevant* if ACR_r is not significantly greater (or is significantly lesser) than ACR_{base}.

5. EXPERIMENTAL RESULTS

In this section, we shall present the results of some experiments aimed at determining feature relevancy under the two proposed criteria.

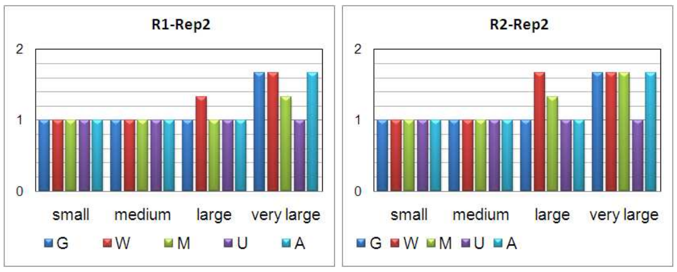


Figure 6: Learnt Optimal Policies for *Rep2* under reward model *R1* and *R2* (G=*GUM*, W=*WillUM*, M=*ModwillUM*, U=*UnwillUM*, A=*AllUM*), 1 = *Execute*, 2 = *Suggest*

5.1 Evaluation under Policy Diversity

In this section, we shall present and analyze our results for the Policy Diversity criteria. The overall result is that all the considered representations are relevant under all the possible user models. The representations that yield the most different policies under a combination of the user models are *Rep2* and *Rep3*.

To this end, we have represented our learnt optimal policies (50 in all) in Figures 4-8, which depict the policies, under both reward models $R1$ and $R2$, for the *Baseline*, *Rep1*, *Rep2*, *Rep3*, and *Rep4* representations respectively. Specifically, each figure comprises two graphs, for $R1$ and $R2$, and has been entitled accordingly, e.g., the graph *R1-Baseline* in Figure 4 represents the optimal policies learnt under *R1* for the *Baseline* representation. In this paper, we will refer to the graphs through their titles. Furthermore, in each graph, the x-axis represents the possible values of the CRS feature for different user models, while the y-axis represents the optimal policy action for that feature value and user model combination. Here the value 1 represents the action *Execute*, while the value 2 represents the action *Suggest*; the value 0 has no significance.

In order to facilitate the comparison of the optimal policies computed for the new representations (*Rep1*, ..., *Rep4*) with the *Baseline* itself, we have grouped the action value ($1 = Execute$ or $2 = Suggest$) prescribed by the optimal policy for all the states with a given value of the CRS feature. For instance, in Figure 5, a bar labelled with *very large* for the user model $G = GUM$ is the average of the actions prescribed by the optimal policy for all the states in *Rep1* where *CRS* = *very large* and the specific additional state feature introduced in *Rep1*, i.e., ERS, has some other value.

Moreover, we also calculate an overall indication of the *changes* in the optimal policy learnt for each representation in *R* under all the user model behaviors. This change is expressed as a percentage of the number of states with a different optimal action than that prescribed by the OP for *Baseline* state model. We calculate these changes under both *R1* and *R2* (Figure 9). Moreover, we recall that our learnt optimal policies seek to modify the Default Policy (DP) of Nutking that dictates executing the query for small and medium result sizes and suggesting tightening for large and very large ones.

Let us initially determine the relevancy of the representations in R under the reward model *R1*. We will initially consider the user models *GUM*, *WillUM*, *ModwillUM*, and *UnwillUM* only, and we will discuss our results for *AllUM* later on. For *Rep1*, we compare the graphs *R1-Baseline* (Figure 4) and *R1-Rep1* (Figure 5), and observe that under *GUM*, *Rep1* is relevant for states with $CRS = large$ because the numerical average value is less than 2, i.e., the value for *R1-Baseline*. This occurs because a different action, i.e., *Execute* is learnt for some state where $CRS = large$ and ERS is still large, and where suggesting tightening is not effective. Similarly, if, for a small result size bar ($CRS = small$) under a given

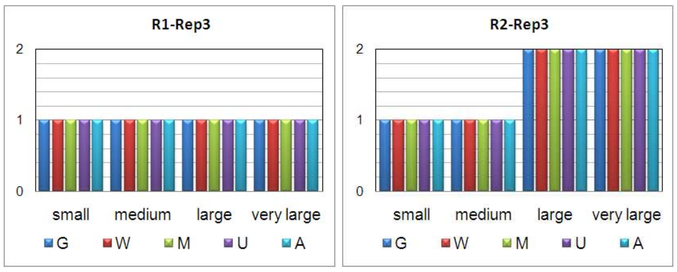


Figure 7: Learnt Optimal Policies for *Rep3* under reward model *R1* and *R2* (G=*GUM*, W=*WillUM*, M=*ModwillUM*, U=*UnwillUM*, A=*AllUM*), 1 = *Execute*, 2 = *Suggest*

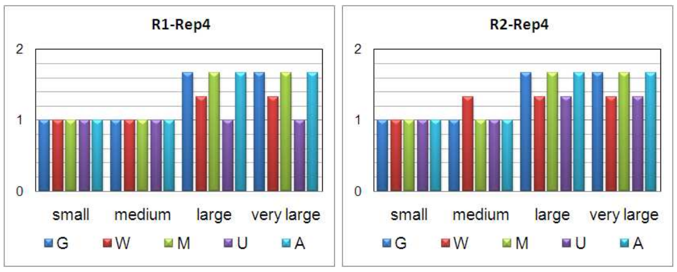


Figure 8: Learnt Optimal Policies for *Rep4* under reward model *R1* and *R2* (G=*GUM*, W=*WillUM*, M=*ModwillUM*, U=*UnwillUM*, A=*AllUM*), 1 = *Execute*, 2 = *Suggest*

representation, the average value is greater than 1, this implies that the representation is relevant since the optimal policy is different from that of the $Baseline$, i.e., not always prescribing to $Execute$ the query (value 1). We can observe that *Rep1* is also relevant for $CRS = verylarge$, since sometimes *Execute* is prescribed instead of *Suggest* (for user models GUM and $UnwillUM$). Hence, *Rep1* is relevant under *GUM*, with a policy change of 30% (see the graph entitled *Policy Change under R1* in Figure 9).

Similarly, we can see that *Rep1* is also relevant under *WillUM*, *ModwillUM*, and *UnwillUM* with policy changes of 10%, 10%, and 20% respectively. For *Rep1*, the greatest policy change occurs for *GUM*. We conclude that *Rep1* is relevant under all the user models.

For *Rep2*, we compare graphs *R1-Baseline* (Figure 4) and *R1-Rep2* (Figure 6), and observe that it is relevant under *GUM*, *WillUM*, *ModwillUM*, and *UnwillUM* with the corresponding policy change percentages being 33%, 25%, 42%, and 25% respectively. For *Rep2*, the greatest policy change occurs for *UnwillUM*.

For *Rep3*, comparing graphs *R1-Baseline* and *R1-Rep3* (Figure 7), we observe that it is also relevant for the aforementioned user models, with the corresponding policy change percentages being 50%, 50%, 50%, and 25% respectively. In this case, the greatest policy change occurs for *GUM*, *WillUM* and *ModwillUM*.

Similarly, for *Rep4*, we can observe in graphs *R1-Baseline* and *R1-Rep4* (Figure 8) that it is also relevant under the four considered user models with the corresponding policy change percentages being 17%, 33%, 33%, and 25% respectively. Here, the greatest policy change occurs for *WillUM* and *ModwillUM*. Thus, if we consider the Policy Diversity criteria, each representation in the set *R* is relevant under *GUM*, *WillUM*, *ModwillUM*, and *UnwillUM*. From Figure 9, we can see that the greatest policy change (from the DP) occurs for *Rep3* (50%), while the least change occurs for *Rep1* (10%).

Although all the representations in *R* are relevant, Figures 5-8 show that their relevancy is being influenced by the particular value of *CRS*, e.g., under *ModwillUM*, *Rep1* is relevant only for states where $CRS = large$, while under *UnwillUM*, *Rep1* is relevant only for states where $CRS = verylarge$. This proves that *given a particular reward model, the relevancy of a state representation could be influenced by changes in the user behavior*. Furthermore, for a given representation, different user models give a different optimal policy, modifying the original policy DP by different percentages, i.e., different policies are being learnt for different classes of users.

In order to cater for these facts, for a given policy-learning task, it is best to consider the user behavior for a population of users, i.e., under *AllUM*, rather than for a particular class of users. Then, under *AllUM*, all the representations in *R* are again relevant: *Rep1* and *Rep2* are relevant for $CRS = large$, while *Rep3* and *Rep4* are relevant for $CRS \in \{large, verylarge\}$. This implies that for our user population, it is beneficial for the system to consider any representation $r \in R$ for the policy-learning task, instead of the *Baseline* representation. The corresponding policy change percentages under *Rep1*, *Rep2*, *Rep3* and *Rep4* are 10%, 33%, 50%, and 17% respectively (Figure 9), with the greatest policy change, similarly to other user models, again occurring for *Rep3* (50%), and the least occurring for *Rep1* (10%).

Let us now determine the relevancy of the representations in *R* under the reward model *R2*. We will again initially consider the user models *GUM*, *WillUM*, *ModwillUM*, and *UnwillUM* only. For *Rep1*, we compare the graphs *R2-Baseline* (Figure 4) and *R2-Rep1* Figure 5 and observe that it is relevant under the aforementioned user models, with the corresponding policy change percentages being 40%, 10%, 10%, and 20% respectively (the graph labelled *Policy Change under R2* in Figure 9). We recall that the corresponding change under *GUM* for *R1* was 30%. In fact, under *R1*, *Rep1* was relevant only for large and very large result sizes, while under *R2*, *Rep1* is also relevant for medium result sizes.

This proves that *given a particular user behavior, the relevancy of a state representation could be influenced by changes in the reward model*. For *Rep2*, we compare the graphs *R2-Baseline* and *R2-Rep2*, and observe that it is relevant under our selected user models with the corresponding percentages being 33%, 17%, 25%, and 25% respectively (Figure 9). In this case, the greatest policy change occurs for *GUM* (similarly to *R1*).

For *Rep3*, we compare the graphs *R2-Baseline* and *R2-Rep3*, and observe that it is relevant just under *UnwillUM* (percentage of change is 25%), while for *R1*, *Rep3* was relevant under all of our selected user models. Thus, this again proves that the relevancy is influenced by changes in the reward model.

For *Rep4*, we compare the graphs *R2-Baseline* and *R2-Rep4*, and see that it is relevant under all the user models, specifically for medium (under *WillUM*), large and very large result sizes (under all the user models). These results show that different optimal policies are being learnt for different user models, and also that these policies are being learnt in a different manner for different user models. This again motivates the application of *AllUM* user model for determining feature relevancy. Under *AllUM*, *Rep1*, *Rep2* and *Rep4* are relevant, with the corresponding percentages being 10%, 33%, and 17% respectively, with the greatest policy change occurring for *Rep2*, and the least occurring for *Rep1*. This implies that for our user population, under *R2*, it is beneficial for the system to consider representations *Rep1*, *Rep2* and *Rep4* for the policy-learning task, instead of the *Baseline* representation.

Having determined relevancy of the set *R*, we will now mention some analytical results concerning the learnt policies in Figures 4-8. We note that under *R2*, the optimal policy for *Rep3* (Figure 7)

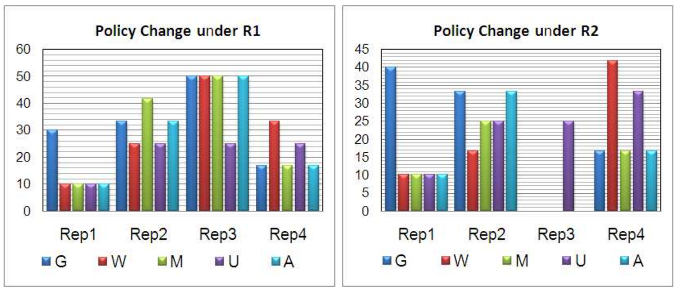


Figure 9: Optimal policy changes (in %) for *Rep1*, *Rep2*, *Rep3* and *Rep4*, under *R1* and *R2*, G=*GUM*, W=*WillUM*, M=*ModwillUM*, U=*UnwillUM*, A=*AllUM*

UM	*RM*	Different State Representations				
		Baseline	*Rep1*	*Rep2*	*Rep3*	*Rep4*
GUM	*R1*	4.515	**4.555**	**4.562**	**4.564**	**4.523**
	R2	6.223	**7.733**	**8.363**	6.223	**6.303**
WillUM	*R1*	4.569	**4.572**	4.563	4.564	*4.543*
	R2	6.757	**6.910**	**8.363**	6.757	**7.880**
ModwillUM	*R1*	4.535	4.539	**4.563**	**4.564**	4.539
	R2	6.043	6.120	**8.363**	6.020	*5.963*
UnwillUM	*R1*	4.529	**4.553**	**4.564**	**4.564**	**4.564**
	R2	6.037	**7.760**	**8.363**	*5.523*	**8.363**
AllUM	*R1*	4.532	**4.545**	**4.563**	**4.564**	4.534
	R2	6.200	**6.283**	**8.363**	5.523	**6.227**

Table 4: Average cumulative rewards under different "User Model-Reward Model-State Representation" combinations (*UM*=User Model,*RM*=Reward Model)

under all user models is exactly DP. This is also true, under both *R1* and *R2*, for *Baseline* under the models *GUM*, *WillUM*, *ModwillUM*, and *AllUM*. This shows that the DP (not explicitly optimized) could also be optimal for certain classes of users, hence justifying our initial selection of DP. Furthermore, for each representation $r \in R$, DP doesn't change much for small result sizes. In fact, for $CRS = small$, it doesn't change at all, and for $CRS = medium$, it changes only twice, i.e., for *Rep2* and *Rep4* under *R2*. This proves that, for our particular interaction scenario, it is best to execute the query for small result sizes (< 50).

Furthermore, we note from Figure 7 that for *Rep3* under *R1*, the DP is modified for large and very large result sizes but, differently from *Rep1* and *Rep2*, there is a 100% policy change with respect to DP for all the user models, i.e., the system learns to always execute the query for large and very large result sizes. This shows that under *R1*, *Rep3* provides a more diverse set of information to the system as compared to *Rep1* and *Rep2*, because the system modifies DP to a greater degree for large result sizes. Also, we note from Figure 7 that equal optimal policies are being learnt under all the user models, under both the reward models.

Moreover, we observe from Figure 8 that, under *R1*, the optimal policy for *GUM*, *WillUM*, *ModwillUM*, and *AllUM* sometimes dictates executing the query for large and very large result sizes. Analyzing the full policy description, we observed that this occurs only for states where the response of the user to tightening suggestions ($UserTResp$) is either $mixed$ or $reject$. This means that the system learns not to suggest tightening to users who are moderately-willing to accept it, or who always reject it. However, for $UserTResp = accept$, the system learns to always suggest tightening. This proves that our system is intelligently modifying its behavior by catering for the willingness of the (simulated) user to respond to tightening suggestions.

5.2 Evaluation under Policy Value

In this section, we will present our results for the Policy Value criteria. The overall result of these experiments is showing that, under this criteria as well, there are differences in the optimal policies according to the representation and the user model. Moreover, it emerges that *Rep2* is producing a larger average cumulative reward and therefore should be preferred for this particular recommender.

Table 4 shows the *AvgCumRwd* values obtained for each policy evaluation, i.e., for each of the 50 optimal policies (Figures 2-6). All the rewards for the representations in *R* which are significantly greater than their corresponding *Baseline* reward are shown in bold, while the rewards which are significantly less have been italicized.

Let us first determine the relevancy of representations in *R* under *R1*. As we did for Policy Diversity, we will initially consider *GUM*, *WillUM*, *ModwillUM*, and *UnwillUM* only. Under *R1*, *Rep1* is relevant for *GUM*, *WillUM* and *UnwillUM*, while it is irrelevant for *ModwillUM*. Furthermore, *Rep2* and *Rep3* are relevant for *GUM*, *ModwillUM* and *UnwillUM*, while they are irrelevant for *WillUM*. Also, *Rep4* is relevant for *GUM* and *UnwillUM*, while it is irrelevant for *WillUM* and *ModwillUM*. In fact, the reward accumulated for *WillUM* is significantly less than that for *Baseline*. These results clearly show that, *given a reward model, the relevancy of a state representation could be influenced by changes in the user behavior*. Hence, it is imperative to determine relevancy under the user population model *AllUM*. For *AllUM*, *Rep1*, *Rep2*,and *Rep3* are all relevant, while *Rep4* is irrelevant. We note that this result is different than the corresponding results for Policy Diversity, where all the representation in *R* were relevant under *AllUM*.

Let us now determine the relevancy of the representations in *R* under *R2*. Under *R2*, similarly to *R1*, *Rep1* is relevant for *GUM*, *WillUM* and *UnwillUM*, while it is irrelevant for *ModwillUM*. Thus, using a different reward model doesn't affect the relevancy of *Rep1*. Moreover, *Rep2* is relevant for all of our selected user models. Furthermore, the relevancy result for *Rep3* and *Rep4* are also different than the corresponding results for *R1*, e.g., *Rep3* is irrelevant for *GUM*, while *Rep4* is relevant under *WillUM*. Also, the reward accumulated for *ModwillUM* is significantly less than that for *Baseline*. These results demonstrate (similarly to the results for Policy Diversity) that *given a user model behavior, the relevancy of a state representation could be influenced by changes in the reward model*. Furthermore, the relevancy of a given representation is being again influenced by changes in the user behavior, e.g., *Rep4* is relevant for *GUM*, while it is irrelevant for *ModwillUM*. Thus, we again determine relevancy of *R* for *AllUM*. For *AllUM*, *Rep1*, *Rep2*,and *Rep4* are all relevant, while only *Rep3* is irrelevant. Interestingly, this result is exactly the same as the corresponding result for Policy Diversity.

6. FORWARD FEATURE SELECTION

In this section, we will extend our evaluation by performing a *Forward Feature Selection* (FFS) procedure [5], i.e., we will exploit our previous results in order to select a relevant representation as our new baseline representation. Then, we will attempt to determine the relevancy of adding each of the remaining three features to the new baseline. We will continue this procedure until, 1) all of the features have been added to baseline (i.e., all the features turn out to be relevant), or 2) at some stage, none of the remaining features turn out to be relevant. Our aim of performing FFS is to investigate which of the above two conditions occurs in our feature selection procedure.

In order to select our new baseline, we will consider the relevancy results under Policy Value. The rationale is that Policy Value outputs the actual numerical reward which the system can accumulate while employing an OP, and these rewards are a much better indicator of the suitability of an OP (and hence, of the relevancy of a new state representation) than simply a comparison of the different actions learnt under the OP of the new representation (Policy Diversity). Furthermore, for sake of simplicity, in this paper we will use just the *AllUM* user model as it represents a combination of various kinds of user behaviors. In these experiments we focussed on the *R2* reward model. Similar experiments can be conducted using $R1$ as well. We will select *Rep2* as our new baseline representation, as it accumulates the maximum amount of reward (8.363) as compared to other representations and seems to be the best representation under the considered conditions. The goal is to determine the relevancy of the representations $\{$*PUA, CRS, FTSugg, ERS*$\}$ (*Rep5*), $\{$*PUA, CRS, FTSugg, NStages*$\}$ (*Rep6*), and $\{$*PUA, CRS, FTSugg, UserTResp*$\}$ (*Rep7*). The *AvgCumRwd* values obtained for *Rep5*, *Rep6* and *Rep7* are 7.627, 6.371 and 6.646 respectively, i.e., none of these representations are relevant for further inclusion and the feature selection procedure terminates prematurely. So the conclusion of this experiments is reaffirming that *Rep2* is a meaningful state representation for the current recommendation problem.

7. LIVE-USERS EXPERIMENTS

In our previous experiments, we have investigated the issue of feature relevancy, under different reward models and user model behaviors. Although our results are encouraging, they were obtained under a limited setting, i.e., there is only one situation in which the system must learn to make a decision, and the system's action set comprises only two actions *Execute* and *Suggest*. Moreover, we have carried out this evaluation *off-line*, i.e., in simulated interactions. Furthermore, our results show that, for our recommendation policy-learning task, the behavior of the user population does influence the optimal policy. Hence, a meaningful policy for the recommender agent must be obtained only in an online setting with real users. Also, in order to justify our recommendation approach for e-Commerce systems, it is imperative to learn and validate the optimal policy in an online experiment.

In order to address these limitations, we have applied our approach within the Travel Planner (TP) tool of the Austrian tourism web portal (Austria.info), in the context of the $_{et}$*Packaging* project funded by Austrian Network for E-Tourism (ANET). Here, we are evaluating our approach with real users and under a more complex setting, i.e., the system should learn an increased number of decisions, and under a more detailed action set. The default policy of TP is exactly the default policy of NutKing, and our aim is to improve this default policy in order to learn an optimal one [8]. For this policy-learning task, we have proposed four system decision points, which support interactive decisions such as, whether to query the user for her preferences earlier on in the interaction or later on, or whether to push the user to add some product to her cart or not etc. The total number of possible states is 41287680, but when the users evaluated our system, the number of states which actually occurred were 5261, out of which the system had to learn the optimal action for 739 states (corresponding to the situations at the four SDPs). Moreover, the total number of possible system actions is 30.

In this evaluation, we have used the aforementioned results on feature relevancy in order to select a part of our state representation. Specifically, we have considered our results for Policy Value, under the reward model *R2* and the user model *AllUM*.

Hence, in our state representation, we include the variable *User Action*, which is similar to *PUA*, in that it depicts the various actions which the user could take while interacting with the system. We also include the variable *Current Result Size* (similar to *CRS*). We don't directly consider *FTSugg*, but rather we consider the variable *UserTResp* because it also implicitly provides information about *FTSugg* to the system, i.e., the frequency of times tightening has been suggested is used to determine whether users accept or reject these suggestions.

Furthermore, in our experiment, the system also supports query *relaxation*, i.e., when the user's query retrieves too many results, the system suggests a set of features in order to unconstrain the query, and hence, retrieve some results. Hence, we have also included the variable *UserRelaxResponse* in our state representation, which depicts the user's response to relaxation suggestions, and encodes the useful information about the number of times relaxation has been suggested during the interaction. Yet another related system action is the automatic relaxation (AutoRelax) of the query by our system. In this context, we also include the variable *UserAutoRelaxResponse* in our representation.

In total, our representation comprises of *twelve* variables (we shall not enlist them here in detail). We have used this representation in order to learn an optimal policy for the Travel Planner (TP) tool, and we have successfully validated our policy with real users (a more detailed account of these experiments will be the topic of a forthcoming paper). For instance, one important result is that the optimal policy is able to assist the users in acquiring their goals in a lesser number of interaction stages, and by viewing a lesser number of result pages, as compared to the system's behavior under one or more non-optimal (default) policy.

8. RELATED WORK

Reinforcement Learning has been applied previously within the domain of recommender systems in order to learn an optimal recommendation strategy, but with limited results, and with quite a different model of the recommendation process. In [10], the authors model the state as the set of products previously bought by the user, and where the system action is aimed at computing the next set of recommendations. Quite similarly, in [15] and [14], the state model basically comprises the set of pages previously viewed by the user, and the goal is to determine the next best page (set of products) to recommend to the user. A similar approach has also been applied in [16] where the state comprises the current page which the user is viewing, and the goal is to recommend the next best link on this page. Another related work has been carried out by [4], where the state is either the current URL (page) of the user, or the current product selected by the user, and the goal is to intelligently identify the best recommendation algorithm among some competitive alternatives for personalizing the recommendations for some user. Furthermore, in [19], the authors determine the presentation order of future recommendations by exploiting the current user feedback as a reward. In all of these works, the goal is similar to that of a classical recommender system i.e., to learn what products to recommend next (or which recommendation page to show next. Hence, a similar technology (Reinforcement Learning) is exploited differently as compared to our approach, where the goal is to decide which action to choose in a more diverse set of possible system moves, in order to intelligently and interactively assist the users in acquiring their goals.

The problem of determining feature relevancy has been addressed in the domain of dialogue systems. In [17], the authors exploit the Policy Diversity criteria to determine the relevancy of five state representations. Also, [3] adopt a similar criteria to Policy Value in order to prove the relevancy of adding two dialogue features to a baseline representation. Moreover, [12] determine which feature values are optimal for different points in the dialogue, and show that the number of information and distress attributes impact the state. To the best of our knowledge, our work is the first attempt in solving the feature relevancy problem for RL-based conversational recommender systems. We note that our results for Policy Diversity and Policy Value are different (for say, the user population model *AllUM*). In our point of view, more research needs to be done in order to better understand which relevancy criteria could be suitable for a given policy-learning task.

9. CONCLUSIONS AND FUTURE WORK

In this paper, we have investigated the issue of determining state feature relevancy in conversational recommender systems, and we have shown that adding a new feature is not always beneficial, and that the relevancy is influenced by changes in the user behavior, and also by changes in the reward model. Moreover, we have shown how these results have been exploited in a live-users experiment where we considered real-user behavior and we learned the optimal policy for a more complex set of system actions and for a much larger state space. As stated previously, our online state representation comprises twelve state variables, and we have learnt the optimal policy using all of these features.

For any recommendation policy-learning problem, a set of potential features could be available for inclusion in the state representation. The goal of our feature selection procedure is to determine the relevancy of adding these features, until some stopping criteria is met, i.e., until all of the features have been added or, at some iteration of the feature selection procedure, none of the remaining features can be added, i.e., the procedure terminates prematurely. Our future work would concentrate on investigating this issue. As an initial step, we have carried out a *forward feature selection* (FFS) procedure, by exploiting the relevancy results under a particular system configuration, and have shown that, for our recommendation task, the procedure terminates prematurely, with the addition of only one feature to *baseline*.

We are now working to determine whether all of the identified features adopted for the online experiment are relevant for learning the policy. For this, we plan to execute a backward feature selection (BFS) procedure on the current state model. Specifically, we will exploit the Policy Value criteria, and we will initially calculate the *AvgCumRwd* value for the current baseline representation. Then, we will remove one feature, say *f*, from it and for the new representation, we will calculate its *AvgCumRwd* value and compare it with the *AvgCumRwd* value for the baseline. If the new value is larger, we will consider *f* as irrelevant, and will select the new representation as our new baseline. In this way, we will continue our backward feature selection until the removal of a feature leads to a lesser *AvgCumRwd* value.

10. REFERENCES

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, 2005.

[2] D. Bridge, M. Göker, L. McGinty, and B. Smyth. Case-based recommender systems. *The Knowledge Engineering review*, 20(3):315–320, 2006.

[3] M. Frampton and O. Lemon. Learning more effective dialogue strategies using limited dialogue move features. In *ACL '06*, 2006.

[4] N. Golovin and E. Rahm. Reinforcement learning architecture for web recommendations. In *International Conference on Information Technology: Coding and Computing (ITCC'04), Volume 1, April 5-7, 2004, Las Vegas, Nevada, USA*, pages 398–402, 2004.

[5] R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

[6] T. Mahmood and F. Ricci. Learning and adaptivity in interactive recommender systems. In *ICEC '07: Proceedings of the ninth international conference on Electronic commerce*, pages 75–84, 2007.

[7] T. Mahmood and F. Ricci. Towards learning user-adaptive state models in a conversational recommender system. In *ABIS'07: Proceedings 15th Workshop on Adaptivity and User Modeling in Interactive Systems*, Halle, Germany, 2007.

[8] T. Mahmood, F. Ricci, A. Venturini, and W. Höpken. Adaptive recommender systems for travel planning. In W. H. Peter OConnor and U. Gretzel, editors, *Information and Communication Technologies in Tourism 2008, proceedings of ENTER 2008 International Conference*, pages 1–11, Innsbruck, 2008. Springer.

[9] N. Mirzadeh and F. Ricci. Cooperative query rewriting for decision making support and recommender systems. *Applied Artificial Intelligence*, 21:1–38, 2007.

[10] G. Shani, D. Heckerman, and R. I. Brafman. An mdp-based recommender system. *Journal of Machine Learning Research*, 6:1265–1295, 2005.

[11] H. Shimazu. ExpertClerk: Navigating shoppers buying process with the combination of asking and proposing. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001*, Seattle, Washington, USA, 2001.

[12] S. P. Singh, M. J. Kearns, D. J. Litman, and M. A. Walker. Reinforcement learning for spoken dialogue systems. In *NIPS Conference, Colorado, USA*, 1999.

[13] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[14] N. Taghipour and A. Kardan. A hybrid web recommender system based on q-learning. In *Proceedings of the 2008 ACM Symposium on Applied Computing (SAC), Fortaleza, Ceara, Brazil, March 16-20, 2008*, pages 1164–1168, 2008.

[15] N. Taghipour, A. Kardan, and S. S. Ghidary. Usage-based web recommendations: a reinforcement learning approach. In *Proceedings of the 2007 ACM Conference on Recommender Systems, RecSys 2007, Minneapolis, MN, USA, October 19-20, 2007*, pages 113–120, 2007.

[16] S. ten Hagen, M. van Someren, and V. Hollink. Exploration/exploitation in adaptive recommender systems. In *Proceedings of the European Symposium on Intelligent Technologies, Hybrid Systems and their implementation on Smart Adaptive Systems*, Oulu, Finland, 2003.

[17] J. R. Tetreault and D. J. Litman. Using reinforcement learning to build a better model of dialogue state. In *EACL*, 2006.

[18] C. A. Thompson, M. H. Goker, and P. Langley. A personalized system for conversational recommendations. *Artificial Intelligence Research*, 21:393–428, 2004.

[19] Y. Z. Wei, L. Moreau, and N. R. Jennings. Learning users' interests in a market-based recommender system. In *IDEAL*, pages 833–840, 2004.